XBinder

XML Schema Compiler
Version 2.9
C Runtime
Reference Manual

Objective Systems, Inc. — November 2023

Copyright Notice

Author’s Contact Information

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

Contents

Chapter 1

C Common Runtime Library Functions

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

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 Bit String Functions

Bit string functions allow bits to be set, cleared, or tested in arbitrarily sized byte arrays.

Macros

- #define OSRTBYTEARRAYSIZE(numbits) ((numbits+7)/8)

  This macro is used to calculate the byte array size required to hold the given number of bits.

Functions

- EXTERNRT OSUINT32 rtxGetBitCount (OSUINT32 value)

  This function returns the minimum size of the bit field required to hold the given integer value.
- EXTERNRT int rtxSetBit (OSOCTET ∗pBits, OSSIZE numbits, OSSIZE bitIndex)

  This function sets the specified zero-counted bit in the bit string.
- EXTERNRT 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.
- EXTERNRT int rtxClearBit (OSOCTET ∗pBits, OSSIZE numbits, OSSIZE bitIndex)

  This function clears the specified zero-counted bit in the bit string.
- EXTERNRT OSBOOL rtxTestBit (const OSOCTET ∗pBits, OSSIZE numbits, OSSIZE bitIndex)

  This function tests the specified zero-counted bit in the bit string.
- EXTERNRT OSSIZE rtxLastBitSet (const OSOCTET ∗pBits, OSSIZE numbits)

  This function returns the zero-counted index of the last bit set in a bit string.
- EXTERNRT 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.
- EXTERNRT int rtxZeroUnusedBits (OSOCTET ∗pBits, OSSIZE numbits)

  This function zeros unused bits at the end of the given bit string.
- EXTERNRT int rtxCheckUnusedBitsZero (const OSOCTET ∗pBits, OSSIZE numbits)

  This function checks to see if unused bits at the end of the given bit string are zero.

5.1.1 Detailed Description

Bit string functions allow bits to be set, cleared, or tested in arbitrarily sized byte arrays.

5.1.2 Function Documentation

5.1.2.1 rtxCheckBitBounds()

```
EXTERNRT 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.

5.1.2.2 rtxCheckUnusedBitsZero()

```
EXTERNRT int rtxCheckUnusedBitsZero (
             const OSOCTET * pBits,
             OSSIZE numbits )
```

This function checks to see if unused bits at the end of the given bit string are zero.

Parameters

pBits	Pointer to the octets of the bit string.
numbits	The number of bits in the bit string.

Returns

Zero if the operation is successful, or a negative status code if the operation fails. RTERR_INVBINS will be returned if bits at end are not zero.

5.1.2.3 rtxClearBit()

```
EXTERNRT 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 the bit. If bit was previously set, the return value is positive. If bit was not previously set, the return value is zero. Otherwise, return value is an error code:

- RTERR_OUTOFBND = bitIndex is out of bounds

5.1.2.4 rtxGetBitCount()

```
EXTERNRT 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.

5.1.2.5 rtxLastBitSet()

```
EXTERNRT 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. OSNULLINDEX if no bit is set.

5.1.2.6 rtxSetBit()

```
EXTERNRT 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 the bit. If bit was previously set, the return value is positive. If bit was not previously set, the return value is zero. Otherwise, return value is an error code:

- RTERR_OUTOFBND = bitIndex is out of bounds

5.1.2.7 rtxSetBitFlags()

```
EXTERNRT 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.

5.1.2.8 rtxTestBit()

```
EXTERNRT 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.

5.1.2.9 rtxZeroUnusedBits()

```
EXTERNRT int rtxZeroUnusedBits (
             OSOCTET * pBits,
             OSSIZE numbits )
```

This function zeros unused bits at the end of the given bit string.

Parameters

pBits	Pointer to the octets of the bit string.
numbits	The number of bits in the bit string.

Returns

Zero if the operation is successful, or a negative status code if the operation fails.

5.2 Context Message Buffer Read/Write Functions

The context message buffer can be written to, or read from, using various functions.

Classes

- struct _OSRTBufLocDescr

  Buffer location descriptor.

Typedefs

- typedef struct _OSRTBufLocDescr OSRTBufLocDescr

  Buffer location descriptor.

Functions

- EXTERNRT int rtxCheckOutputBuffer (OSCTXT ∗pctxt, size_t nbytes)

  This function checks to ensure that the output buffer has sufficient space to hold an additional nbytes full bytes (if there is a partially filled byte, it is treated as though full).
- EXTERNRT OSBOOL rtxIsOutputBufferFlushable (OSCTXT ∗pctxt)

  This function returns true if the context buffer can be flushed to a stream by calling rtxFlushOutputBuffer.
- EXTERNRT int rtxFlushOutputBuffer (OSCTXT ∗pctxt)

  This function flushes the buffer to a stream.
- EXTERNRT int rtxExpandOutputBuffer (OSCTXT ∗pctxt, size_t nbytes)

  This function attempts to ensure the output buffer has at least the given number of bytes available.
- EXTERNRT int rtxCheckInputBuffer (OSCTXT ∗pctxt, size_t nbytes)

  Ensures the given number of bytes are available in the context buffer.
- EXTERNRT int rtxLoadInputBuffer (OSCTXT ∗pctxt, OSSIZE nbytes)

  This is for meant for internal use by the runtime.
- EXTERNRT int rtxPeekByte (OSCTXT ∗pctxt, OSOCTET ∗pbyte)

  This function peeks at the next byte of input, if there is one before EOF.
- EXTERNRT int rtxPeekBytes (OSCTXT ∗pctxt, OSOCTET ∗pdata, OSSIZE bufsize, OSSIZE nocts, OSSIZE ∗pactual)

  This function peeks at the next nocts bytes of input, peeking at fewer bytes if EOF is encountered first.
- EXTERNRT int rtxReadBytesSafe (OSCTXT ∗pctxt, OSOCTET ∗buffer, size_t bufsize, size_t nocts)

  This function safely reads bytes from the currently open stream or memory buffer into the given static buffer.
- EXTERNRT int rtxReadBytes (OSCTXT ∗pctxt, OSOCTET ∗pdata, size_t nocts)

  This function reads bytes from the currently open stream or memory buffer.
- EXTERNRT int rtxReadBytesDynamic (OSCTXT ∗pctxt, OSOCTET ∗∗ppdata, size_t nocts, OSBOOL ∗pMem↩ Alloc)

  This function reads bytes from the currently open stream or memory buffer.
- EXTERNRT int rtxWriteBytes (OSCTXT ∗pctxt, const OSOCTET ∗pdata, size_t nocts)

  This function writes bytes to the currently open stream or memory buffer.
- EXTERNRT int rtxWriteIndent (OSCTXT ∗pctxt)

  This function writes a newline followed by indentation whitespace to the buffer.

- EXTERNRT void rtxIndentDecr (OSCTXT ∗pctxt)

  This decreases the indentation level set in the given context by updating the indent member.
- EXTERNRT void rtxIndentIncr (OSCTXT ∗pctxt)

  This increases the indentation level set in the given context by updating the indent member.
- EXTERNRT void rtxIndentReset (OSCTXT ∗pctxt)

  This resets the indentation level in the given context to zero.
- EXTERNRT size_t rtxGetIndentLevels (OSCTXT ∗pctxt)

  This returns the number of levels of indentation set in the given context.
- EXTERNRT OSBOOL rtxCanonicalSort (OSOCTET ∗refPoint, OSRTSList ∗pList, OSBOOL normal)

  Sort a list of buffer locations, referring to component encodings, by comparing the referenced encodings as octet strings.
- EXTERNRT int rtxEncCanonicalSort (OSCTXT ∗pctxt, OSCTXT ∗pMemCtxt, OSRTSList ∗pList)

  Encode the encodings held in pMemCtxt into pctxt, first sorting them as required for canonical BER (and other encoding rules) by X.690 11.6.
- EXTERNRT void rtxGetBufLocDescr (OSCTXT ∗pctxt, OSRTBufLocDescr ∗pDescr)

  Set the buffer location description's offset (pDescr->offset) to the current position in pCtxt's buffer.
- EXTERNRT void rtxAddBufLocDescr (OSCTXT ∗pctxt, OSRTSList ∗pElemList, OSRTBufLocDescr ∗pDescr)

  Create a new Asn1BufLocDescr for an element just encoded and append it to pElemList.

5.2.1 Detailed Description

The context message buffer can be written to, or read from, using various functions.

5.2.2 Function Documentation

5.2.2.1 rtxAddBufLocDescr()

```
EXTERNRT void rtxAddBufLocDescr (
             OSCTXT * pctxt,
             OSRTSList * pElemList,
             OSRTBufLocDescr * pDescr )
```

Create a new Asn1BufLocDescr for an element just encoded and append it to pElemList.

Parameters

pctxt	Pointer to context where data has been encoded.
pElemList	List of Asn1BufLocDescr to which a new entry will be added.
pDescr	Pointer to Asn1BufLocDescr whose offset indicates the start of the element just encoded. The new Asn1BufLocDescr that is added will have the same offset and will have numocts determined by this offset and pctxt's current buffer position.

5.2.2.2 rtxCanonicalSort()

```
EXTERNRT OSBOOL rtxCanonicalSort (
            OSOCTET * refPoint,
            OSRTSList * pList,
            OSBOOL normal )
```

Sort a list of buffer locations, referring to component encodings, by comparing the referenced encodings as octet strings.

The sorting can be used with canonical-BER (CER), distinguished-BER (DER), and canonical-XER (XER).

Encoding into the buffer may be done as a normal encoding (start to end) or as a reverse encoding (end to start). This affects the parameters as described below.

Parameters

refPoint	Reference point in the buffer for the buffer locations. For normal encoding, refPoint is the start of the buffer; for reverse encoding, refPoint is the end of the buffer.
pList	List of OSRTBufLocDescr, each of which locates the start of an encoded component. The offsets for the locations are relative to refPoint. If normal is TRUE, this function orders the list from least to greatest. Otherwise, it is ordered from greatest to least.
normal	TRUE for normal encoding; FALSE for reverse encoding. This tells the function whether to add or substract offsets from refPoint to locate the component encodings and also how to order the list.

Returns

TRUE if any changes to pList were made; FALSE otherwise (meaning the list was already in the desired order).

5.2.2.3 rtxCheckInputBuffer()

```
EXTERNRT int rtxCheckInputBuffer (
            OSCTXT * pctxt,
            size_t nbytes )
```

Ensures the given number of bytes are available in the context buffer.

If a stream is attached to the context and is being buffered by the context buffer, this may read from the stream to fill the buffer. Thus this function may alter the context buffer byteIndex and size. Any bytes read from the stream will be sent to the capture buffer, if there is one.

Returns

0 on success, or less than zero on failure. RTERR_ENDOFBUF or RTERR_ENDOFFILE (depending on whether the source is a stream or not) is returned if the requested number of bytes are not available in the input.

5.2.2.4 rtxCheckOutputBuffer()

```
EXTERNRT int rtxCheckOutputBuffer (
             OSCTXT * pctxt,
             size_t nbytes )
```

This function checks to ensure that the output buffer has sufficient space to hold an additional nbytes full bytes (if there is a partially filled byte, it is treated as though full).

Dynamic buffers are resized if the check fails, while static buffers induce a buffer overflow error. This function may return RTERR_NOMEM if reallocating the dynamic buffer fails.

Parameters

pctxt	Pointer to a context structure.
nbytes	The requested capacity for the buffer.

Returns

0 on success, or less than zero on failure.

5.2.2.5 rtxEncCanonicalSort()

```
EXTERNRT int rtxEncCanonicalSort (
             OSCTXT * pctxt,
             OSCTXT * pMemCtxt,
             OSRTSList * pList )
```

Encode the encodings held in pMemCtxt into pctxt, first sorting them as required for canonical BER (and other encoding rules) by X.690 11.6.

Parameters

pctxt	Pointer to context structure into which the sorted encodings should be encoded.
pMemCtxt	Pointer to context structure which holds the unsorted encodings.
pList	List of Asn1BufLocDescr, each of which locates an encoding in pMemCtxt's buffer, the whole being the encodings that are to be sorted.

5.2.2.6 rtxExpandOutputBuffer()

```
EXTERNRT int rtxExpandOutputBuffer (
             OSCTXT * pctxt,
             size_t nbytes )
```

This function attempts to ensure the output buffer has at least the given number of bytes available.

A partially full byte in the buffer counts as being available. Dynamic buffers are resized, if necessary, while static buffers induce a buffer overflow error. This function may return RTERR_NOMEM if reallocating a dynamic buffer fails.

Parameters

pctxt	Pointer to a context structure.
nbytes	The requested capacity for the buffer.

Returns

0 on success, or less than zero on failure.

5.2.2.7 rtxFlushOutputBuffer()

```
EXTERNRT int rtxFlushOutputBuffer (
             OSCTXT * pctxt )
```

This function flushes the buffer to a stream.

This function MUST only be called if rtxIsOutputBufferFlushable(pctxt) returns TRUE; the behavior is otherwise undefined. After a successful call, pctxt->buffer.byteIndex == 0. Note that pctxt->buffer.bitOffset is not changed - if there was a partial byte in the buffer before this call, there will be a partial by afterward.

Parameters

pctxt	Pointer to a context structure.

Returns

0 on success, or less than zero on failure.

5.2.2.8 rtxGetIndentLevels()

```
EXTERNRT size_t rtxGetIndentLevels (
             OSCTXT * pctxt )
```

This returns the number of levels of indentation set in the given context.

Currently, the indentation level in the context affects rtxPrintToStreamIndent (when used with a non-null OSCTXT), rtxWriteIndent, and rtJsonEncIndent, meaning it currently affects print-to-stream output, Abstract Syntax Notation output, and JSON (JER) output.

Parameters

pctxt	Pointer to context block structure.

5.2.2.9 rtxIndentDecr()

```
EXTERNRT void rtxIndentDecr (
             OSCTXT * pctxt )
```

This decreases the indentation level set in the given context by updating the indent member.

See also

rtxGetIndentLevels

Parameters

pctxt	Pointer to context block structure.

5.2.2.10 rtxIndentIncr()

```
EXTERNRT void rtxIndentIncr (
             OSCTXT * pctxt )
```

This increases the indentation level set in the given context by updating the indent member.

See also

rtxGetIndentLevels

Parameters

pctxt	Pointer to context block structure.

5.2.2.11 rtxIndentReset()

```
EXTERNRT void rtxIndentReset (
             OSCTXT * pctxt )
```

This resets the indentation level in the given context to zero.

See also

rtxGetIndentLevels

Parameters

pctxt	Pointer to context block structure.

5.2.2.12 rtxIsOutputBufferFlushable()

```
EXTERNRT OSBOOL rtxIsOutputBufferFlushable (
            OSCTXT * pctxt )
```

This function returns true if the context buffer can be flushed to a stream by calling rtxFlushOutputBuffer.

This function is used to determine whether it is safe to call rtxFlushOutputBuffer.

Parameters

pctxt	Pointer to a context structure.

Returns

TRUE if the buffer can be flushed; FALSE is not.

5.2.2.13 rtxLoadInputBuffer()

```
EXTERNRT int rtxLoadInputBuffer (
            OSCTXT * pctxt,
            OSSIZE nbytes )
```

This is for meant for internal use by the runtime.

Read at least as many bytes from the context's input stream into the context buffer as necessary to make nbytes of data available in the context buffer.

Upon return, pctxt.buffer.byteIndex + nbytes <= pctxt.buffer.size OR EOF has been reached OR an error has been logged and is being returned.

If the context buffer has not been created, this will create it. If the context buffer needs to be made larger, this will enlarge it or else log, and return, an error.

Parameters

pctxt	A context with an attached stream using the context's buffer as a buffer for the stream.

Returns

0 or or negative error. EOF is not considered an error.

5.2.2.14 rtxPeekByte()

```
EXTERNRT int rtxPeekByte (
            OSCTXT * pctxt,
            OSOCTET * pbyte )
```

This function peeks at the next byte of input, if there is one before EOF.

A return of 1 ensures the peeked byte can be skipped by incrementing pctxt->buffer.byteIndex.

Parameters

pbyte	Receives the value of the peeked byte.

Returns

0 if there was no byte to peek at. 1 if there was a byte to peek at. <0 if there was an error.

5.2.2.15 rtxPeekBytes()

```
EXTERNRT int rtxPeekBytes (
            OSCTXT * pctxt,
            OSOCTET * pdata,
            OSSIZE bufsize,
            OSSIZE nocts,
            OSSIZE * pactual )
```

This function peeks at the next nocts bytes of input, peeking at fewer bytes if EOF is encountered first.

A returned value (in *pactual) of n (>=0) ensures that n (or fewer) peeked bytes can be skipped by adding n (or less) to pctxt->buffer.byteIndex.

Parameters

pdata	Receives the value of the peeked bytes.
bufsize	Size of pdata. If less than nocts, nocts will be adjusted accordingly.
nocts	The number of bytes to peek.
pactual	Receives the actual number of bytes peeked.

Returns

0 for success <0 if there was an error.

5.2.2.16 rtxReadBytes()

```
EXTERNRT int rtxReadBytes (
             OSCTXT * pctxt,
             OSOCTET * pdata,
             size_t nocts )
```

This function reads bytes from the currently open stream or memory buffer.

Parameters

pctxt	Pointer to a context structure.
pdata	Pointer to byte array where bytes are to be copied.
nocts	Number of bytes (octets) to read.

Returns

Status of the operation: 0 if success, negative value if error.

5.2.2.17 rtxReadBytesDynamic()

```
EXTERNRT int rtxReadBytesDynamic (
             OSCTXT * pctxt,
             OSOCTET ** ppdata,
             size_t nocts,
             OSBOOL * pMemAlloc )
```

This function reads bytes from the currently open stream or memory buffer.

In this case the function MAY allocate memory to hold the read bytes. It will only do this if the requested number of bytes will not fit in the context buffer; othwerwise, a pointer to a location in the context buffer is returned. If memory was allocated, it should be freed using rtxMemFreePtr.

Parameters

pctxt	Pointer to a context structure.
ppdata	Pointer to byte buffer pointer.
nocts	Number of bytes (octets) to read.
pMemAlloc	Pointer to boolean value which is set to true if memory was allocated to hold requested bytes.

Returns

Status of the operation: 0 if success, negative value if error.

5.2.2.18 rtxReadBytesSafe()

```
EXTERNRT int rtxReadBytesSafe (
            OSCTXT * pctxt,
            OSOCTET * buffer,
            size_t bufsize,
            size_t nocts )
```

This function safely reads bytes from the currently open stream or memory buffer into the given static buffer.

This function is preferred over `rtxReadBytes` because it will detect buffer overflow.

Parameters

pctxt	Pointer to a context structure.
buffer	Static buffer into which bytes are to be read.
bufsize	Size of the static buffer.
nocts	Number of bytes (octets) to read.

Returns

Status of the operation: 0 if success, negative value if error.

5.2.2.19 rtxWriteBytes()

```
EXTERNRT int rtxWriteBytes (
            OSCTXT * pctxt,
            const OSOCTET * pdata,
            size_t nocts )
```

This function writes bytes to the currently open stream or memory buffer.

Parameters

pctxt	Pointer to a context structure.
pdata	Pointer to location where bytes are to be copied.
nocts	Number of bytes to read.

Returns

I/O byte count.

5.2.2.20 rtxWriteIndent()

```
EXTERNRT int rtxWriteIndent (
             OSCTXT * pctxt )
```

This function writes a newline followed by indentation whitespace to the buffer.

The amount of indentation to add is determined by the indent member variable in the context structure.

If context flag OSNOWHITESPACE is set, this function will do nothing.

Parameters

pctxt	Pointer to context block structure.

Returns

Completion status of operation:

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

5.3 Character string functions

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

Functions

- EXTERNRT int rtxStricmp (const char ∗str1, const char ∗str2)

  This is an implementation of the non-standard stricmp function.
- EXTERNRT int rtxStrnicmp (const char ∗str1, const char ∗str2, size_t count)

  This is an implementation of the non-standard stricmp function.
- EXTERNRT char ∗ rtxStrcat (char ∗dest, size_t bufsiz, const char ∗src)

  This function concatanates the given string onto the string buffer.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxStrcpy (char ∗dest, size_t bufsiz, const char ∗src)

  This function copies a null-terminated string to a target buffer.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxStrdup (OSCTXT ∗pctxt, const char ∗src)

  This function creates a duplicate copy of a null-terminated string.
- EXTERNRT char ∗ rtxStrndup (OSCTXT ∗pctxt, const char ∗src, OSSIZE nchars)

  This function creates a duplicate copy of up to the given number of characters in a string.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxStrTrimEnd (char ∗s)

  This function trims whitespace from the end of a string.
- EXTERNRT int rtxValidateConstrainedStr (OSCTXT ∗pctxt, const char ∗pvalue, const char ∗pCharSet)

  This function will validate a constrained character string by checking to see if all the characters in the given string are contained within the

  ### constraining character set.
- EXTERNRT int rtxIntToCharStr (OSINT32 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts a signed 32-bit integer into a character string.
- EXTERNRT int rtxUIntToCharStr (OSUINT32 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts an unsigned 32-bit integer into a character string.
- EXTERNRT int rtxInt64ToCharStr (OSINT64 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts a signed 64-bit integer into a character string.
- EXTERNRT int rtxUInt64ToCharStr (OSUINT64 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts an unsigned 64-bit integer into a character string.
- EXTERNRT 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.

- EXTERNRT 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.
- EXTERNRT int rtxHexCharsToBin (const char ∗hexstr, size_t nchars, OSOCTET ∗binbuf, size_t bufsize)

  This function converts the given hex string to binary.
- EXTERNRT int rtxCharStrToInt (const char ∗cstr, OSINT32 ∗pvalue)

  This function converts the given character string to a signed 32-bit integer value.
- EXTERNRT int rtxCharStrnToInt (const char ∗cstr, OSSIZE ndigits, OSINT32 ∗pvalue)

  This function converts up to the given number of digits from the given character string to a signed 32-bit integer value.
- EXTERNRT int rtxCharStrToInt8 (const char ∗cstr, OSINT8 ∗pvalue)

  This function converts the given character string to a signed 8-bit integer value.
- EXTERNRT int rtxCharStrToInt16 (const char ∗cstr, OSINT16 ∗pvalue)

  This function converts the given character string to a signed 16-bit integer value.
- EXTERNRT int rtxCharStrToInt64 (const char ∗cstr, OSINT64 ∗pvalue)

  This function converts the given character string to a signed 64-bit integer value.
- EXTERNRT int rtxCharStrToUInt (const char ∗cstr, OSUINT32 ∗pvalue)

  This function converts the given character string to an unsigned 32-bit integer value.
- EXTERNRT int rtxCharStrToUInt8 (const char ∗cstr, OSUINT8 ∗pvalue)

  This function converts the given character string to an unsigned 8-bit integer value.
- EXTERNRT int rtxCharStrToUInt16 (const char ∗cstr, OSUINT16 ∗pvalue)

  This function converts the given character string to an unsigned 16-bit integer value.
- EXTERNRT int rtxCharStrToUInt64 (const char ∗cstr, OSUINT64 ∗pvalue)

  This function converts the given character string to an unsigned 64-bit integer value.

5.3.1 Detailed Description

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

5.3.2 Function Documentation

5.3.2.1 rtxCharStrnToInt()

```
EXTERNRT int rtxCharStrnToInt (
             const char * cstr,
             OSSIZE ndigits,
             OSINT32 * pvalue )
```

This function converts up to the given number of digits from the given character string to a signed 32-bit integer value.

It consumes all leading whitespace before the digits start. It then comsumes digits until either the number of digits is reached or a non-digit character is encountered.

Parameters

cstr	Character string to convert.
ndigits	Number of digits to convert.
pvalue	Pointer to integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.2 rtxCharStrToInt()

```
EXTERNRT int rtxCharStrToInt (
             const char * cstr,
             OSINT32 * pvalue )
```

This function converts the given character string to a signed 32-bit 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.

5.3.2.3 rtxCharStrToInt16()

```
EXTERNRT int rtxCharStrToInt16 (
             const char * cstr,
             OSINT16 * pvalue )
```

This function converts the given character string to a signed 16-bit 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 16-bit integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.4 rtxCharStrToInt64()

```
EXTERNRT int rtxCharStrToInt64 (
             const char * cstr,
             OSINT64 * pvalue )
```

This function converts the given character string to a signed 64-bit 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 64-bit integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.5 rtxCharStrToInt8()

```
EXTERNRT int rtxCharStrToInt8 (
             const char * cstr,
             OSINT8 * pvalue )
```

This function converts the given character string to a signed 8-bit 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 8-bit integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.6 rtxCharStrToUInt()

```
EXTERNRT int rtxCharStrToUInt (
             const char * cstr,
             OSUINT32 * pvalue )
```

This function converts the given character string to an unsigned 32-bit 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 32-bit unsigned integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.7 rtxCharStrToUInt16()

```
EXTERNRT int rtxCharStrToUInt16 (
             const char * cstr,
             OSUINT16 * pvalue )
```

This function converts the given character string to an unsigned 16-bit 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 16-bit unsigned integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.8 rtxCharStrToUInt64()

```
EXTERNRT int rtxCharStrToUInt64 (
             const char * cstr,
             OSUINT64 * pvalue )
```

This function converts the given character string to an unsigned 64-bit 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 64-bit unsigned integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.9 rtxCharStrToUInt8()

```
EXTERNRT int rtxCharStrToUInt8 (
             const char * cstr,
             OSUINT8 * pvalue )
```

This function converts the given character string to an unsigned 8-bit 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 8-bit unsigned integer value to receive converted data.

Returns

Number of bytes or negative status value if fail.

5.3.2.10 rtxHexCharsToBin()

```
EXTERNRT 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.

5.3.2.11 rtxHexCharsToBinCount()

```
EXTERNRT 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.

5.3.2.12 rtxInt64ToCharStr()

```
EXTERNRT 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.

5.3.2.13 rtxIntToCharStr()

```
EXTERNRT 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.

5.3.2.14 rtxSizeToCharStr()

```
EXTERNRT 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.

5.3.2.15 rtxStrcat()

```
EXTERNRT 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.

5.3.2.16 rtxStrcpy()

```
EXTERNRT 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.

5.3.2.17 rtxStrdup()

```
EXTERNRT 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.

5.3.2.18 rtxStrDynJoin()

```
EXTERNRT 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.

5.3.2.19 rtxStricmp()

```
EXTERNRT 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.

5.3.2.20 rtxStrJoin()

```
EXTERNRT 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.

5.3.2.21 rtxStrncat()

```
EXTERNRT 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.

5.3.2.22 rtxStrncpy()

```
EXTERNRT 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 target buffer is too short to hold the null terminator, the last character is overwritten and a null pointer is returned; the destination buffer can still be examined in this case.

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.

5.3.2.23 rtxStrndup()

```
EXTERNRT char* rtxStrndup (
             OSCTXT * pctxt,
             const char * src,
             OSSIZE nchars )
```

This function creates a duplicate copy of up to the given number of characters in a string.

The string does not need to be null-terminated. 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 `strndup` 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.

5.3.2.24 rtxStrnicmp()

```
EXTERNRT int rtxStrnicmp (
             const char * str1,
             const char * str2,
             size_t count )
```

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.
count	Number of characters to compare, at most.

Returns

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

5.3.2.25 rtxStrTrimEnd()

```
EXTERNRT char* rtxStrTrimEnd (
             char * s )
```

This function trims whitespace from the end of a string.

Parameters

s	Pointer to string to be trimmed.

Returns

Point to string s.

5.3.2.26 rtxUInt64ToCharStr()

```
EXTERNRT 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.

5.3.2.27 rtxUIntToCharStr()

```
EXTERNRT 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.

5.3.2.28 rtxValidateConstrainedStr()

```
EXTERNRT int rtxValidateConstrainedStr (
             OSCTXT * pctxt,
             const char * pvalue,
             const char * pCharSet )
```

This function will validate a constrained character string by checking to see if all the characters in the given string are contained within the

constraining character set.

Parameters

pctxt	Pointer to a context block structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue	A pointer to a null-terminated C character string to be validated.
pCharSet	A pointer to a null-terminated C character string containing the character set to validate against.

Returns

Status of the validation attempt. A negative status value will be returned if validation is not successful.

5.4 Context Management Functions

Context initialization functions handle the allocation, initialization, and destruction of context variables (variables of type OSCTXT).

Classes

- struct OSRTErrLocn

  Run-time error location structure.
- struct OSRTErrInfo

  Run-time error information structure.
- struct OSRTBuffer

  Run-time message buffer structure.
- struct OSRTBufSave

  Structure to save the current message buffer state.
- struct OSBufferIndex

  This structure can be used as an index into the buffer.
- struct OSCTXT

  Run-time context structure.

Macros

- #define OSNOWHITESPACE 0x00400000 /∗ Turn off indentation whitesapce ∗/

  Turn off unnecessary whitespace in text output.
- #define rtxCtxtGetMsgPtr(pctxt) (pctxt)->buffer.data

  This macro returns the start address of an encoded message.
- #define rtxCtxtGetMsgLen(pctxt) (pctxt)->buffer.byteIndex

  This macro returns the length of an encoded message.
- #define rtxCtxtTestFlag(pctxt, mask) (((pctxt)->flags & mask) != 0)

  This macro tests if the given bit flag is set in the context.
- #define rtxCtxtPeekElemName(pctxt)

  This macro returns the last element name from the context stack.
- #define rtxByteAlign(pctxt)

  This macro will byte-align the context buffer.
- #define rtxCtxtSetProtocolVersion(pctxt, value) (pctxt)->version = value

  This macro sets the protocol version in the context.

Typedefs

- 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.
- 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.
- typedef void(∗ OSFreeCtxtGlobalPtr) (struct OSCTXT ∗pctxt)

  OSRTFreeCtxtGlobalPtr is a pointer to a memory free function.

Functions

- EXTERNRT int rtxInitContext (OSCTXT ∗pctxt)

    This function initializes an OSCTXT block.
- EXTERNRT int rtxInitContextExt (OSCTXT ∗pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, O↩ SFreeFunc free_func)

    This function initializes an OSCTXT block.
- EXTERNRT int rtxInitThreadContext (OSCTXT ∗pctxt, const OSCTXT ∗pSrcCtxt)

    This function initializes a context for use in a thread.
- EXTERNRT int rtxInitContextUsingKey (OSCTXT ∗pctxt, const OSOCTET ∗key, OSSIZE keylen)

    This function initializes a context using a run-time key.
- EXTERNRT int rtxInitContextBuffer (OSCTXT ∗pctxt, OSOCTET ∗bufaddr, OSSIZE bufsiz)

    This function assigns a message buffer to a context block.
- EXTERNRT int rtxCtxtSetBufPtr (OSCTXT ∗pctxt, OSOCTET ∗bufaddr, OSSIZE bufsiz)

    This function is used to set the internal buffer pointer for in-memory encoding or decoding.
- EXTERNRT OSSIZE rtxCtxtGetBitOffset (OSCTXT ∗pctxt)

    This function returns the total bit offset to the current element in the context buffer.
- EXTERNRT int rtxCtxtSetBitOffset (OSCTXT ∗pctxt, OSSIZE offset)

    This function sets the bit offset in the context to the given value.
- EXTERNRT OSSIZE rtxCtxtGetIOByteCount (OSCTXT ∗pctxt)

    This function returns the count of bytes either written to a stream or memory buffer.
- EXTERNRT int rtxCheckContext (OSCTXT ∗pctxt)

    This function verifies that the given context structure is initialized and ready for use.
- EXTERNRT void rtxFreeContext (OSCTXT ∗pctxt)

    This function frees all dynamic memory associated with a context.
- EXTERNRT void rtxCopyContext (OSCTXT ∗pdest, OSCTXT ∗psrc)

    This function creates a copy of a context structure.
- EXTERNRT void rtxCtxtSetFlag (OSCTXT ∗pctxt, OSUINT32 mask)

    This function is used to set a processing flag within the context structure.
- EXTERNRT void rtxCtxtClearFlag (OSCTXT ∗pctxt, OSUINT32 mask)

    This function is used to clear a processing flag within the context structure.
- EXTERNRT 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.
- EXTERNRT int rtxCtxtPushElemName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗elemName)

    This function is used to push an element name onto the context element name stack.
- EXTERNRT int rtxCtxtPushElemNameCopy (OSCTXT ∗pctxt, const OSUTF8CHAR ∗elemName)

    This function is used to push a copy of the given element name onto the context element name stack.
- EXTERNRT int rtxCtxtPushTypeName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗typeName)

    This function is used to push a type name onto the context element name stack.
- EXTERNRT OSBOOL rtxCtxtPopArrayElemName (OSCTXT ∗pctxt)

    This function pops the last element name from the context stack.
- EXTERNRT const OSUTF8CHAR ∗ rtxCtxtPopElemName (OSCTXT ∗pctxt)

    This function pops the last element name from the context stack.
- EXTERNRT void rtxCtxtPopElemNameCopy (OSCTXT ∗pctxt)

    This function pops the last element name from the context stack and frees the associated memory.
- EXTERNRT const OSUTF8CHAR ∗ rtxCtxtPopTypeName (OSCTXT ∗pctxt)

    This function pops the type name from the context stack.

- EXTERNRT OSBOOL rtxCtxtContainerHasRemBits (OSCTXT ∗pctxt)

  Return true iff there are bits remaining to be decoded in the current length-constrained container, which is possibly the outer PDU.
- EXTERNRT OSBOOL rtxCtxtContainerEnd (OSCTXT ∗pctxt)

  Return true if we are at the end of container - neither having more bits remaining nor having overrun it; otherwise return false.
- EXTERNRT OSSIZE rtxCtxtGetContainerRemBits (OSCTXT ∗pctxt)

  Return the number of bits remaining to be decoded in the current length-constrained container, which is possibly the outer PDU.
- EXTERNRT int rtxCtxtPushContainerBytes (OSCTXT ∗pctxt, OSSIZE bytes)

  Notify the runtime layer of the start of decoding of a length-constrained container of a given length.
- EXTERNRT int rtxCtxtPushContainerBits (OSCTXT ∗pctxt, OSSIZE bits)

  Notify the runtime layer of the start of decoding of a length-constrained container of a given length.
- EXTERNRT void rtxCtxtPopContainer (OSCTXT ∗pctxt)

  Notify the runtime layer of the end of decoding of a length-constrained container of the given length.
- EXTERNRT void rtxCtxtPopAllContainers (OSCTXT ∗pctxt)

  Pop all containers from the container stack.
- EXTERNRT void rtxMemHeapSetFlags (OSCTXT ∗pctxt, OSUINT32 flags)

  This function sets flags to a heap.
- EXTERNRT void rtxMemHeapClearFlags (OSCTXT ∗pctxt, OSUINT32 flags)

  This function clears memory heap flags.
- EXTERNRT int rtxCtxtMarkBitPos (OSCTXT ∗pctxt, OSSIZE ∗ppos)

  This function saves the current bit position in a message buffer.
- EXTERNRT int rtxCtxtResetToBitPos (OSCTXT ∗pctxt, OSSIZE pos)

  This function resets a message buffer back to the given bit position.
- EXTERNRT int rtxMarkPos (OSCTXT ∗pctxt, OSSIZE ∗ppos)

  This function saves the current position in a message buffer or stream.
- EXTERNRT int rtxResetToPos (OSCTXT ∗pctxt, OSSIZE pos)

  This function resets a message buffer or stream back to the given position.
- EXTERNRT const char ∗ rtxCtxtGetExpDateStr (OSCTXT ∗pctxt, char ∗buf, OSSIZE bufsiz)

  This function will get the license expiration date for a time-limited license.
- EXTERNRT void rtxLicenseClose (void)

  Finish with current license and free internal resources.

5.4.1 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.

5.4.2 Macro Definition Documentation

5.4.2.1 OSNOWHITESPACE

```
#define OSNOWHITESPACE 0x00400000 /* Turn off indentation whitesapce */
```

Turn off unnecessary whitespace in text output.

Currently, this affects Abstract Value Notation and JSON (JER) output.

Definition at line 150 of file rtxContext.h.

5.4.2.2 rtxCtxtGetMsgLen

```
#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.

Definition at line 469 of file rtxContext.h.

5.4.2.3 rtxCtxtGetMsgPtr

```
#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.

Definition at line 458 of file rtxContext.h.

5.4.2.4 rtxCtxtPeekElemName

```
#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.

Definition at line 667 of file rtxContext.h.

5.4.2.5 rtxCtxtSetProtocolVersion

```
#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.

Definition at line 861 of file rtxContext.h.

5.4.2.6 rtxCtxtTestFlag

```
#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

Definition at line 554 of file rtxContext.h.

5.4.3 Typedef Documentation

5.4.3.1 OSFreeCtxtGlobalPtr

```
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.

Definition at line 176 of file rtxContext.h.

5.4.4 Function Documentation

5.4.4.1 rtxCheckContext()

```
EXTERNRT 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

5.4.4.2 rtxCopyContext()

```
EXTERNRT 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.

5.4.4.3 rtxCtxtClearFlag()

```
EXTERNRT 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.

5.4.4.4 rtxCtxtContainerEnd()

```
EXTERNRT OSBOOL rtxCtxtContainerEnd (
            OSCTXT * pctxt )
```

Return true if we are at the end of container - neither having more bits remaining nor having overrun it; otherwise return false.

Overflowing the container should only be possible when a container length has been pushed onto the container stack, since the runtime doesn't allow going beyond the end of the buffer.

See also rtxCtxtPushContainer(Bits|Bytes)/rtxCtxtPopContainer

Parameters

pctxt	Pointer to context structure.

5.4.4.5 rtxCtxtContainerHasRemBits()

```
EXTERNRT OSBOOL rtxCtxtContainerHasRemBits (
            OSCTXT * pctxt )
```

Return true iff there are bits remaining to be decoded in the current length-constrained container, which is possibly the outer PDU.

See also rtxCtxtPushContainer(Bits|Bytes)/rtxCtxtPopContainer

Parameters

pctxt	Pointer to context structure.

5.4.4.6 rtxCtxtGetBitOffset()

```
EXTERNRT 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.

5.4.4.7 rtxCtxtGetContainerRemBits()

```
EXTERNRT OSSIZE rtxCtxtGetContainerRemBits (
            OSCTXT * pctxt )
```

Return the number of bits remaining to be decoded in the current length-constrained container, which is possibly the outer PDU.

See also rtxCtxtPushContainer(Bits|Bytes)/rtxCtxtPopContainer

Parameters

pctxt	Pointer to context structure.

5.4.4.8 rtxCtxtGetExpDateStr()

```
EXTERNRT const char* rtxCtxtGetExpDateStr (
            OSCTXT * pctxt,
            char * buf,
            OSSIZE bufsiz )
```

This function will get the license expiration date for a time-limited license.

Parameters

pctxt	Pointer to a context block.
buf	Buffer to receive date string.
bufsiz	Size of the buffer.

Returns

Pointer to character string if successful (will be point to buf) or null if no expiration date was found.

5.4.4.9 rtxCtxtGetIOByteCount()

```
EXTERNRT 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.

5.4.4.10 rtxCtxtMarkBitPos()

```
EXTERNRT int rtxCtxtMarkBitPos (
            OSCTXT * pctxt,
            OSSIZE * ppos )
```

This function saves the current bit position in a message buffer.

Parameters

pctxt	Pointer to a context block.
ppos	Pointer to saved position.

Returns

Completion status of operation:

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

5.4.4.11 rtxCtxtPopAllContainers()

```
EXTERNRT void rtxCtxtPopAllContainers (
            OSCTXT * pctxt )
```

Pop all containers from the container stack.

This is useful for clearing the stack when an error has occured. It is invoked automatically by rtxErrReset.

Parameters

pctxt	Pointer to context structure.

5.4.4.12 rtxCtxtPopArrayElemName()

```
EXTERNRT 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.

5.4.4.13 rtxCtxtPopContainer()

```
EXTERNRT void rtxCtxtPopContainer (
            OSCTXT * pctxt )
```

Notify the runtime layer of the end of decoding of a length-constrained container of the given length.

This method should be called when the final bit to be decoded has been decoded.

This pops an entry off of pctxt->containerEndIndex

Parameters

pctxt	Pointer to context structure.

5.4.4.14 rtxCtxtPopElemName()

```
EXTERNRT 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.

5.4.4.15 rtxCtxtPopElemNameCopy()

```
EXTERNRT void rtxCtxtPopElemNameCopy (
             OSCTXT * pctxt )
```

This function pops the last element name from the context stack and frees the associated memory.

It is assumed it was added to the stack using the rtxCtxtPushElemNameCopy function.

Parameters

pctxt	Pointer to a context structure.

5.4.4.16 rtxCtxtPopTypeName()

```
EXTERNRT 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.

5.4.4.17 rtxCtxtPushArrayElemName()

```
EXTERNRT 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.

5.4.4.18 rtxCtxtPushContainerBits()

```
EXTERNRT int rtxCtxtPushContainerBits (
            OSCTXT * pctxt,
            OSSIZE bits )
```

Notify the runtime layer of the start of decoding of a length-constrained container of a given length.

This method should be called when the next bit to be decoded is the first bit of the length-constrained content.

This pushes an entry onto pctxt->containerEndIndex.

Parameters

pctxt	Pointer to context structure.
bits	Number of bits in the length-constrained container.

Returns

Completion status of operation:

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

5.4.4.19 rtxCtxtPushContainerBytes()

```
EXTERNRT int rtxCtxtPushContainerBytes (
            OSCTXT * pctxt,
            OSSIZE bytes )
```

Notify the runtime layer of the start of decoding of a length-constrained container of a given length.

This method should be called when the next bit to be decoded is the first bit of the length-constrained content.

This pushes an entry onto pctxt->containerEndIndex.

Parameters

pctxt	Pointer to context structure.
bytes	Number of bytes of the length-constrained container.

Returns

Completion status of operation:

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

5.4.4.20 rtxCtxtPushElemName()

```
EXTERNRT 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 list element fails.

5.4.4.21 rtxCtxtPushElemNameCopy()

```
EXTERNRT int rtxCtxtPushElemNameCopy (
            OSCTXT * pctxt,
            const OSUTF8CHAR * elemName )
```

This function is used to push a copy of the given element name onto the context element name stack.

A copy of the element name is made using context memory management. The name should be popped using the rtxCtxtPopElemNameCopy function to ensure memory is freed.

Parameters

pctxt	Pointer to a context structure.
elemName	Name of element to be pushed on stack. A copy of the name is made.

Returns

Completion status of operation:

- 0 = success,
- RTERR_NOMEM if mem alloc for name or list element fails.

5.4.4.22 rtxCtxtPushTypeName()

```
EXTERNRT 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.

5.4.4.23 rtxCtxtResetToBitPos()

```
EXTERNRT int rtxCtxtResetToBitPos (
             OSCTXT * pctxt,
             OSSIZE pos )
```

This function resets a message buffer back to the given bit position.

Parameters

pctxt	Pointer to a context block.
pos	Context position.

Returns

Completion status of operation:

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

5.4.4.24 rtxCtxtSetBitOffset()

```
EXTERNRT 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

5.4.4.25 rtxCtxtSetBufPtr()

```
EXTERNRT 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).

5.4.4.26 rtxCtxtSetFlag()

```
EXTERNRT 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.

5.4.4.27 rtxFreeContext()

```
EXTERNRT 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.

5.4.4.28 rtxInitContext()

```
EXTERNRT 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.

NOTE: This as part of the common runtime, this is not intended to be called directly by user code; it does not initialize application-specific data structures (e.g. those used for ASN.1). If you are a user reading this, you may instead be interested in rtInitContext.

Parameters

pctxt	Pointer to the context structure variable to be initialized.

Returns

Completion status of operation:

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

5.4.4.29 rtxInitContextBuffer()

```
EXTERNRT 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.

5.4.4.30 rtxInitContextExt()

```
EXTERNRT 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.

5.4.4.31 rtxInitContextUsingKey()

```
EXTERNRT int rtxInitContextUsingKey (
             OSCTXT * pctxt,
             const OSOCTET * key,
             OSSIZE keylen )
```

This function initializes a context using a run-time key.

This form is required for evaluation and limited distribution software. The compiler will generate a macro for rtXmlInitContext in the rtkey.h file that will invoke this function with the generated run-time key.

Parameters

pctxt	The pointer to the context structure variable to be initialized.
key	Key data generated by ASN1C compiler.
keylen	Key data field length.

Returns

Completion status of operation:

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

5.4.4.32 rtxInitThreadContext()

```
EXTERNRT 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.

5.4.4.33 rtxLicenseClose()

```
EXTERNRT void rtxLicenseClose (
             void  )
```

Finish with current license and free internal resources.

To avoid crashing your application:

- Do not call this during an exit handler function registered with atexit().
- Do not call this when another thread might concurrently trigger a license check by invoking methods in the asn1c runtime.

5.4.4.34 rtxMarkPos()

```
EXTERNRT int rtxMarkPos (
             OSCTXT * pctxt,
             OSSIZE * ppos )
```

This function saves the current position in a message buffer or stream.

Note that this saves the byte offset only and does not consider the bit position. Therefore, it may not be suitable for use with the packed encoding rules (see rtxMarkBitPos).

Parameters

pctxt	Pointer to a context block.
ppos	Pointer to saved position.

Returns

Completion status of operation:

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

5.4.4.35 rtxMemHeapClearFlags()

```
EXTERNRT 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

5.4.4.36 rtxMemHeapSetFlags()

```
EXTERNRT 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.

5.4.4.37 rtxResetToPos()

```
EXTERNRT 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.

5.5 Date/time conversion functions

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

Functions

- EXTERNRT 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).
- EXTERNRT 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]).
- EXTERNRT 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]).
- EXTERNRT int rtxGYearToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian year value to a string (CCYY).
- EXTERNRT int rtxGYearMonthToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian year and month value to a string (CCYY-MM).
- EXTERNRT int rtxGMonthToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian month value to a string (MM).
- EXTERNRT int rtxGMonthDayToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian month and day value to a string (MM-DD).
- EXTERNRT int rtxGDayToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian day value to a string (DD).
- EXTERNRT int rtxGetCurrDateTime (OSNumDateTime ∗pvalue)

  This function reads the system date and time and stores the value in the given OSNumDateTime structure variable.
- EXTERNRT int rtxGetCurrDateTimeString (char ∗buffer, OSSIZE bufsize, OSBOOL local)

  This function reads the current system date and time and returns it as a formatted string.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxCmpDateTime (const OSNumDateTime ∗pvalue1, const OSNumDateTime ∗pvalue2)

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

- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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 OSNumDate↩ Time to the decoded value.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR ∗buf, OSUINT32 bufsize)

  This function converts millisecs to a duration string with format "PnYnMnDTnHnMnS".
- EXTERNRT int rtxDurationToMSecs (OSUTF8CHAR ∗buf, OSUINT32 bufsize, OSINT32 ∗msecs)

  This function converts a duration string to milliseconds.
- EXTERNRT int rtxSetDateTime (OSNumDateTime ∗pvalue, struct tm ∗timeStruct)

  This function converts a structure of type 'struct tm' to an OSNumDateTime structure.
- EXTERNRT int rtxGetGMTime (struct tm ∗pvalue, time_t timeMs)

  This function gets GM time.
- EXTERNRT int rtxGetLocalTime (struct tm ∗pvalue, time_t timeMs)

  This function gets local time.
- EXTERNRT int rtxSetLocalDateTime (OSNumDateTime ∗pvalue, time_t timeMs)

  This function converts a local date and time value to an OSNumDateTime structure.
- EXTERNRT int rtxSetUtcDateTime (OSNumDateTime ∗pvalue, time_t timeMs)

  This function converts a UTC date and time value to an OSNumDateTime structure.
- EXTERNRT 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.
- EXTERNRT OSBOOL rtxDateIsValid (const OSNumDateTime ∗pvalue)

  This function verifies that date members (year, month, day, timezone) of the OSNumDateTime structure contains valid values.
- EXTERNRT OSBOOL rtxTimeIsValid (const OSNumDateTime ∗pvalue)

This function verifies that time members (hour, minute, second, timezone) of the OSNumDateTime structure contains valid values.

- EXTERNRT OSBOOL rtxDateTimeIsValid (const OSNumDateTime ∗pvalue)

  This function verifies that all members of the OSNumDateTime structure contains valid values.
- EXTERNRT int rtxAscTime (char ∗buffer, OSSIZE bufsize, struct tm ∗pvalue)

  This function returns a string representation of the given date/time structure.

5.5.1 Detailed Description

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

5.5.2 Function Documentation

5.5.2.1 rtxAscTime()

```
EXTERNRT int rtxAscTime (
             char ∗ buffer,
             OSSIZE bufsize,
             struct tm ∗ pvalue )
```

This function returns a string representation of the given date/time structure.

It is similar to the std::asctime function except uses the secure version of asctime (asctime_s) for Windows.

Parameters

buffer	Character array into which to write characetr data.
bufsize	Size of the character buffer.
pvalue	Pointer value to convert.

Returns

Status of the conversion operation.

5.5.2.2 rtxCmpDate()

```
EXTERNRT 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.

5.5.2.3 rtxCmpDate2()

```
EXTERNRT 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.

5.5.2.4 rtxCmpDateTime()

```
EXTERNRT 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.

5.5.2.5 rtxCmpDateTime2()

```
EXTERNRT 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.

5.5.2.6 rtxCmpTime()

```
EXTERNRT 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.

5.5.2.7 rtxCmpTime2()

```
EXTERNRT 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.

Parameters

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.

5.5.2.8 rtxDateIsValid()

```
EXTERNRT 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.

5.5.2.9 rtxDateTimeIsValid()

```
EXTERNRT 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.

5.5.2.10 rtxDateTimeToString()

```
EXTERNRT 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

5.5.2.11 rtxDateToString()

```
EXTERNRT 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

5.5.2.12 rtxDurationToMSecs()

```
EXTERNRT 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

5.5.2.13 rtxGDayToString()

```
EXTERNRT 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

5.5.2.14 rtxGetCurrDateTime()

```
EXTERNRT 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

5.5.2.15 rtxGetCurrDateTimeString()

```
EXTERNRT int rtxGetCurrDateTimeString (
             char * buffer,
             OSSIZE bufsize,
             OSBOOL local )
```

This function reads the current system date and time and returns it as a formatted string.

The format is that returned by the asctime system function.

Parameters

buffer	Character buffer to receive string. Buffer should be at least 32 characters in size.
bufsize	Size of buffer. If string will not fit in buffer, an error is returned.
local	True for local time, false for GMT.

Returns

Completion status of operation:

- 0 in case success
- negative in case failure

5.5.2.16 rtxGetDateTime()

```
EXTERNRT 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.

5.5.2.17 rtxGetGMTime()

```
EXTERNRT int rtxGetGMTime (
            struct tm * pvalue,
            time_t timeMs )
```

This function gets GM time.

It provides a platform independent abstraction of the the C RTL gmtime function.

Parameters

pvalue	A pointer to tm structure to receive time value.
timeMs	Time value to be converted to GMT.

Returns

Completion status of operation:

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

5.5.2.18 rtxGetLocalTime()

```
EXTERNRT int rtxGetLocalTime (
             struct tm * pvalue,
             time_t timeMs )
```

This function gets local time.

It provides a platform independent abstraction of the the C RTL localtime function.

Parameters

pvalue	A pointer to tm structure to receive local time value.
timeMs	Time value to be converted to local time.

Returns

Completion status of operation:

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

5.5.2.19 rtxGMonthDayToString()

```
EXTERNRT 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

5.5.2.20 rtxGMonthToString()

```
EXTERNRT 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

5.5.2.21 rtxGYearMonthToString()

```
EXTERNRT 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

5.5.2.22 rtxGYearToString()

```
EXTERNRT 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

5.5.2.23 rtxMSecsToDuration()

```
EXTERNRT 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

5.5.2.24 rtxParseDateString()

```
EXTERNRT 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

5.5.2.25 rtxParseDateTimeString()

```
EXTERNRT 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

5.5.2.26 rtxParseGDayString()

```
EXTERNRT 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

5.5.2.27 rtxParseGMonthDayString()

```
EXTERNRT 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

5.5.2.28 rtxParseGMonthString()

```
EXTERNRT 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

5.5.2.29 rtxParseGYearMonthString()

```
EXTERNRT 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

5.5.2.30 rtxParseGYearString()

```
EXTERNRT 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

5.5.2.31 rtxParseTimeString()

```
EXTERNRT 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

5.5.2.32 rtxSetDateTime()

```
EXTERNRT 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.

5.5.2.33 rtxSetLocalDateTime()

```
EXTERNRT 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.

5.5.2.34 rtxSetUtcDateTime()

```
EXTERNRT 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.

5.5.2.35 rtxTimeIsValid()

```
EXTERNRT 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.

5.5.2.36 rtxTimeToString()

```
EXTERNRT 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

5.6 Decimal number utility functions

Decimal utility function provide run-time functions for handling decimal number types defined within a schema.

5.6.1 Detailed Description

Decimal utility function provide run-time functions for handling decimal number types defined within a schema.

5.7 Diagnostic trace functions

These functions add diagnostic tracing to the generated code to assist in finding where a problem might occur.

Classes

- struct OSRTPrintStream

  Structure to hold information about a global PrintStream.

Typedefs

- typedef void(∗ rtxPrintCallback) (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)

  Callback function definition for print stream.
- typedef struct OSRTPrintStream OSRTPrintStream

  Structure to hold information about a global PrintStream.

Functions

- EXTERNRT OSBOOL rtxDiagEnabled (OSCTXT ∗pctxt)

  This function is used to determine if diagnostic tracing is currently enabled for the specified context.
- EXTERNRT 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.
- EXTERNRT OSBOOL rtxSetGlobalDiag (OSBOOL value)

  This function is used to turn diagnostic tracing on or off at run-time on a global basis.
- EXTERNRT void rtxDiagPrint (OSCTXT ∗pctxt, const char ∗fmtspec,...)

  This function is used to print a diagnostics message to `stdout`.
- EXTERNRT void rtxDiagStream (OSCTXT ∗pctxt, const char ∗fmtspec,...)

  This function conditionally outputs diagnostic trace messages to an output stream defined within the context.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT void rtxDiagPrintChars (OSCTXT ∗pctxt, const char ∗data, size_t nchars)

  This function is used to print a given number of characters to standard output.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT void rtxDiagSetTraceLevel (OSCTXT ∗pctxt, OSRTDiagTraceLevel level)

  This function is used to set the maximum trace level for diagnostic trace messages.
- EXTERNRT OSBOOL rtxDiagTraceLevelEnabled (OSCTXT ∗pctxt, OSRTDiagTraceLevel level)

  This function tests if a given trace level is enabled.
- EXTERNRT int rtxSetPrintStream (OSCTXT ∗pctxt, rtxPrintCallback myCallback, void ∗pStrmInfo)

  This function is for setting the callback function for a PrintStream.

- EXTERNRT int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void ∗pStrmInfo)

  This function is for setting the callback function for a PrintStream.
- EXTERNRT 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.
- EXTERNRT int rtxDiagToStream (OSCTXT ∗pctxt, const char ∗fmtspec, va_list arglist)

  Diagnostics print-to-stream function.
- EXTERNRT int rtxPrintStreamRelease (OSCTXT ∗pctxt)

  This function releases the memory held by PrintStream in the context.
- EXTERNRT void rtxPrintStreamToStdoutCB (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)

  Standard callback function for use with print-to-stream for writing to stdout.
- EXTERNRT void rtxPrintStreamToFileCB (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)

  Standard callback function for use with print-to-stream for writing to a file.
- EXTERNRT void rtxPrintStreamToStringCB (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)

  Standard callback function for use with print-to-stream for writing to a string.

Variables

- OSRTPrintStream g_PrintStream

  Global PrintStream.

5.7.1 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.

5.7.2 Function Documentation

5.7.2.1 rtxDiagEnabled()

```
EXTERNRT 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.

5.7.2.2 rtxDiagHexDump()

```
EXTERNRT 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.

5.7.2.3 rtxDiagPrint()

```
EXTERNRT 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.

5.7.2.4 rtxDiagPrintChars()

```
EXTERNRT 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).

5.7.2.5 rtxDiagSetTraceLevel()

```
EXTERNRT 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.

5.7.2.6 rtxDiagStream()

```
EXTERNRT 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

5.7.2.7 rtxDiagStreamHexDump()

```
EXTERNRT 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.

5.7.2.8 rtxDiagStreamPrintBits()

```
EXTERNRT 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

5.7.2.9 rtxDiagStreamPrintChars()

```
EXTERNRT 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).

5.7.2.10 rtxDiagToStream()

```
EXTERNRT 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

5.7.2.11 rtxDiagTraceLevelEnabled()

```
EXTERNRT 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.

5.7.2.12 rtxPrintStreamRelease()

```
EXTERNRT 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

5.7.2.13 rtxPrintStreamToFileCB()

```
EXTERNRT 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.

5.7.2.14 rtxPrintStreamToStdoutCB()

```
EXTERNRT 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.

5.7.2.15 rtxPrintStreamToStringCB()

```
EXTERNRT void rtxPrintStreamToStringCB (
             void * pPrntStrmInfo,
             const char * fmtspec,
             va_list arglist )
```

Standard callback function for use with print-to-stream for writing to a string.

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 a pointer to an OSRTStrBuf structure.
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.

5.7.2.16 rtxPrintToStream()

```
EXTERNRT 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

5.7.2.17 rtxSetDiag()

```
EXTERNRT 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.

5.7.2.18 rtxSetGlobalDiag()

```
EXTERNRT 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.

5.7.2.19 rtxSetGlobalPrintStream()

```
EXTERNRT 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

5.7.2.20 rtxSetPrintStream()

```
EXTERNRT 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

5.8 Doubly-Linked List Utility Functions

The doubly-linked list utility functions provide common routines for managing linked lists.

Classes

- struct OSRTDListNode

  This structure is used to hold a single data item within the list.
- struct OSRTDList

  This is the main list structure.

Functions

- EXTERNRT void rtxDListInit (OSRTDList ∗pList)

  This function initializes a doubly linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListAppend (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pData)

  This function appends an item to the linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListAppendCharArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, size_↩ t length, char ∗pData)

  This function appends an item to the linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListAppendNode (OSRTDList ∗pList, OSRTDListNode ∗pListNode)

  This function appends an `OSRTDListNode` to the linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListInsert (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSSIZE idx, void ∗p↩ Data)

  This function inserts an item into the linked list structure.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT OSRTDListNode ∗ rtxDListFindByIndex (const OSRTDList ∗pList, OSSIZE idx)

  This function will return the node pointer of the indexed entry in the list.
- EXTERNRT 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.
- EXTERNRT OSRTDListNode ∗ rtxDListFindFirstData (const OSRTDList ∗pList)

  This function will return the node pointer of the first non-null data item within the list or NULL if there is no node that has non-null data.
- EXTERNRT 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.
- EXTERNRT void rtxDListFreeNode (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node)

  This function will remove the given node from the list and free memory.
- EXTERNRT void rtxDListRemove (OSRTDList ∗pList, OSRTDListNode ∗node)

  This function will remove the given node from the list.
- EXTERNRT void rtxDListFreeNodes (struct OSCTXT ∗pctxt, OSRTDList ∗pList)

  This function will free all of the dynamic memory used to hold the list node pointers.

- EXTERNRT 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.
- EXTERNRT int rtxDListToArray (struct OSCTXT *pctxt, OSRTDList *pList, void **ppArray, OSSIZE *pElemCount, OSSIZE elemSize)

  *This function converts a doubly linked list of *T to a dynamically (or pre-allocated) array of T.*
- EXTERNRT int rtxDListToPointerArray (struct OSCTXT *pctxt, OSRTDList *pList, void ***ppArray, OSSIZE *pElemCount)

  *This function converts a doubly linked list of *T to a dynamically (or pre-allocated) array of *T.*
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.

5.8.1 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.

5.8.2 Function Documentation

5.8.2.1 rtxDListAppend()

```
EXTERNRT 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.

5.8.2.2 rtxDListAppendArray()

```
EXTERNRT 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.

5.8.2.3 rtxDListAppendArrayCopy()

```
EXTERNRT 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.

5.8.2.4 rtxDListAppendCharArray()

```
EXTERNRT 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.

5.8.2.5 rtxDListAppendNode()

```
EXTERNRT 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.

5.8.2.6 rtxDListFindByData()

```
EXTERNRT 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.

5.8.2.7 rtxDListFindByIndex()

```
EXTERNRT 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.

5.8.2.8 rtxDListFindFirstData()

```
EXTERNRT OSRTDListNode* rtxDListFindFirstData (
             const OSRTDList * pList )
```

This function will return the node pointer of the first non-null data item within the list or NULL if there is no node that has non-null data.

Parameters

pList	A pointer to a linked list structure.

Returns

A pointer to an allocated linked list node structure.

5.8.2.9 rtxDListFindIndexByData()

```
EXTERNRT 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.

5.8.2.10 rtxDListFreeAll()

```
EXTERNRT 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.

5.8.2.11 rtxDListFreeNode()

```
EXTERNRT 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.

5.8.2.12 rtxDListFreeNodes()

```
EXTERNRT 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.

5.8.2.13 rtxDListInit()

```
EXTERNRT 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.

5.8.2.14 rtxDListInsert()

```
EXTERNRT 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.

5.8.2.15 rtxDListInsertAfter()

```
EXTERNRT 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.

5.8.2.16 rtxDListInsertBefore()

```
EXTERNRT 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.

5.8.2.17 rtxDListRemove()

```
EXTERNRT 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.

5.8.2.18 rtxDListToArray()

```
EXTERNRT int rtxDListToArray (
            struct OSCTXT * pctxt,
            OSRTDList * pList,
            void ** ppArray,
            OSSIZE * pElemCount,
            OSSIZE elemSize )
```

This function converts a doubly linked list of ∗T to a dynamically (or pre-allocated) array of T.

The values pointed to are copied to the array. Note: if you have an array of strings to copy to, you should use rtxDListToPointerArray.

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.

5.8.2.19 rtxDListToPointerArray()

```
EXTERNRT int rtxDListToPointerArray (
            struct OSCTXT * pctxt,
            OSRTDList * pList,
            void *** ppArray,
            OSSIZE * pElemCount )
```

This function converts a doubly linked list of ∗T to a dynamically (or pre-allocated) array of ∗T.

The pointers are copied from the list to the 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 array of pointers.
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.

Returns

The number of elements in the returned array.

5.8.2.20 rtxDListToUTF8Str()

```
EXTERNRT 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.

5.9 Enumeration utility functions

Enumeration utility function provide run-time functions for handling enumerations defined within a schema.

Functions

- EXTERNRT OSINT32 rtxLookupEnum (const OSUTF8CHAR ∗strValue, size_t strValueSize, const OSEnumItem enumTable[ ], OSUINT16 enumTableSize)

  This function will return the numeric value for the given enumerated identifier string.
- EXTERNRT OSINT32 rtxLookupEnumU32 (const OSUTF8CHAR ∗strValue, size_t strValueSize, const OSEnumItemU32 enumTable[ ], OSUINT16 enumTableSize)

  This function will return the numeric value for the given enumerated identifier string.
- EXTERNRT OSINT32 rtxLookupBigEnum (const OSUTF8CHAR ∗strValue, size_t strValueSize, const OSBigEnumItem enumTable[ ], OSUINT16 enumTableSize)

  This function will return the numeric value for the given enumerated identifier string.
- EXTERNRT OSINT32 rtxLookupEnumByValue (OSINT32 value, const OSEnumItem enumTable[ ], size_t enumTableSize)

  Lookup enum by integer value.
- EXTERNRT OSINT32 rtxLookupEnumU32ByValue (OSUINT32 value, const OSEnumItemU32 enumTable[ ], size_t enumTableSize)

  Lookup enum by integer value (Unsiged 32-bit integer).
- EXTERNRT OSINT32 rtxLookupBigEnumByValue (const char ∗value, const OSBigEnumItem enumTable[ ], size_t enumTableSize)

  Lookup enum by stringified version of value.
- EXTERNRT int rtxTestNumericEnum (OSINT32 ivalue, const OSNumericEnumItem enumTable[ ], OSUINT16 enumTableSize)

  This function determines if the given numeric enumerated value is within the defined numeration set.

5.9.1 Detailed Description

Enumeration utility function provide run-time functions for handling enumerations defined within a schema.

5.9.2 Function Documentation

5.9.2.1 rtxLookupBigEnum()

```
EXTERNRT OSINT32 rtxLookupBigEnum (
             const OSUTF8CHAR ∗ strValue,
             size_t strValueSize,
             const OSBigEnumItem enumTable[],
             OSUINT16 enumTableSize )
```

This function will return the numeric value for the given enumerated identifier string.

Parameters

strValue	Enumerated identifier value
strValueSize	Length of enumerated identifier
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Index to enumerated item if found; otherwise, negative status code (RTERR_INVENUM).

5.9.2.2 rtxLookupBigEnumByValue()

```
EXTERNRT OSINT32 rtxLookupBigEnumByValue (
             const char * value,
             const OSBigEnumItem enumTable[],
             size_t enumTableSize )
```

Lookup enum by stringified version of value.

Required for ASN.1 because enumerated values do not need to be sequential.

Parameters

value	String version of the enumerated item.
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Index to enumerated item if found; otherwiae, negative status code (RTERR_INVENUM).

5.9.2.3 rtxLookupEnum()

```
EXTERNRT OSINT32 rtxLookupEnum (
             const OSUTF8CHAR * strValue,
             size_t strValueSize,
             const OSEnumItem enumTable[],
             OSUINT16 enumTableSize )
```

This function will return the numeric value for the given enumerated identifier string.

Parameters

strValue	Enumerated identifier value
strValueSize	Length of enumerated identifier
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Index to enumerated item if found; otherwise, negative status code (RTERR_INVENUM).

5.9.2.4 rtxLookupEnumByValue()

```
EXTERNRT OSINT32 rtxLookupEnumByValue (
             OSINT32 value,
             const OSEnumItem enumTable[],
             size_t enumTableSize )
```

Lookup enum by integer value.

Required for ASN.1 because enumerated values do not need to be sequential.

Parameters

value	Integer value of the enumerated item.
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Index to enumerated item if found; otherwiae, negative status code (RTERR_INVENUM).

5.9.2.5 rtxLookupEnumU32()

```
EXTERNRT OSINT32 rtxLookupEnumU32 (
             const OSUTF8CHAR * strValue,
             size_t strValueSize,
             const OSEnumItemU32 enumTable[],
             OSUINT16 enumTableSize )
```

This function will return the numeric value for the given enumerated identifier string.

Parameters

strValue	Enumerated identifier value
strValueSize	Length of enumerated identifier
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Index to enumerated item if found; otherwise, negative status code (RTERR_INVENUM).

5.9.2.6 rtxLookupEnumU32ByValue()

```
EXTERNRT OSINT32 rtxLookupEnumU32ByValue (
             OSUINT32 value,
             const OSEnumItemU32 enumTable[],
             size_t enumTableSize )
```

Lookup enum by integer value (Unsiged 32-bit integer).

Required for ASN.1 because enumerated values do not need to be sequential.

Parameters

value	Unsigned integer value of the enumerated item.
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Index to enumerated item if found; otherwiae, negative status code (RTERR_INVENUM).

5.9.2.7 rtxTestNumericEnum()

```
EXTERNRT int rtxTestNumericEnum (
             OSINT32 ivalue,
             const OSNumericEnumItem enumTable[],
             OSUINT16 enumTableSize )
```

This function determines if the given numeric enumerated value is within the defined numeration set.

Parameters

ivalue	Numeric enumerated value
enumTable	Table containing the defined enumeration
enumTableSize	Number of rows in the table

Returns

Zero (0) if item in table, RTERR_INVENUM if not

5.10 Run-time error status codes.

This is a list of status codes that can be returned by the common run-time functions and generated code.

Macros

- #define RT_OK 0

  Normal completion status.
- #define RT_OK_FRAG 2

  Message fragment return status.
- #define RTERR_BUFOVFLW -1

  Encode buffer overflow.
- #define RTERR_ENDOFBUF -2

  Unexpected end-of-buffer.
- #define RTERR_IDNOTFOU -3

  Expected identifier not found.
- #define RTERR_INVENUM -4

  Invalid enumerated identifier.
- #define RTERR_SETDUPL -5

  Duplicate element in set.
- #define RTERR_SETMISRQ -6

  Missing required element in set.
- #define RTERR_NOTINSET -7

  Element not in set.
- #define RTERR_SEQOVFLW -8

  Sequence overflow.
- #define RTERR_INVOPT -9

  Invalid option in choice.
- #define RTERR_NOMEM -10

  No dynamic memory available.
- #define RTERR_INVHEXS -11

  Invalid hexadecimal string.
- #define RTERR_INVREAL -12

  Invalid real number value.
- #define RTERR_STROVFLW -13

  String overflow.
- #define RTERR_BADVALUE -14

  Bad value.
- #define RTERR_TOODEEP -15

  Nesting level too deep.
- #define RTERR_CONSVIO -16

  Constraint violation.
- #define RTERR_ENDOFFILE -17

  Unexpected end-of-file error.
- #define RTERR_INVUTF8 -18

  Invalid UTF-8 character encoding.

- #define RTERR_OUTOFBND -19

  Array index out-of-bounds.
- #define RTERR_INVPARAM -20

  Invalid parameter passed to a function or method.
- #define RTERR_INVFORMAT -21

  Invalid value format.
- #define RTERR_NOTINIT -22

  Context not initialized.
- #define RTERR_TOOBIG -23

  Value will not fit in target variable.
- #define RTERR_INVCHAR -24

  Invalid character.
- #define RTERR_XMLSTATE -25

  XML state error.
- #define RTERR_XMLPARSE -26

  XML parser error.
- #define RTERR_SEQORDER -27

  Sequence order error.
- #define RTERR_FILNOTFOU -28

  File not found.
- #define RTERR_READERR -29

  Read error.
- #define RTERR_WRITEERR -30

  Write error.
- #define RTERR_INVBASE64 -31

  Invalid Base64 encoding.
- #define RTERR_INVSOCKET -32

  Invalid socket.
- #define RTERR_INVATTR -33

  Invalid attribute.
- #define RTERR_REGEXP -34

  Invalid regular expression.
- #define RTERR_PATMATCH -35

  Pattern match error.
- #define RTERR_ATTRMISRQ -36

  Missing required attribute.
- #define RTERR_HOSTNOTFOU -37

  Host name could not be resolved.
- #define RTERR_HTTPERR -38

  HTTP protocol error.
- #define RTERR_SOAPERR -39

  SOAP error.
- #define RTERR_EXPIRED -40

  Evaluation license expired.
- #define RTERR_UNEXPELEM -41

  Unexpected element encountered.
- #define RTERR_INVOCCUR -42

Invalid number of occurrences.

- #define RTERR_INVMSGBUF -43

  Invalid message buffer has been passed to decode or validate method.
- #define RTERR_DECELEMFAIL -44

  Element decode failed.
- #define RTERR_DECATTRFAIL -45

  Attribute decode failed.
- #define RTERR_STRMINUSE -46

  Stream in-use.
- #define RTERR_NULLPTR -47

  Null pointer.
- #define RTERR_FAILED -48

  General failure.
- #define RTERR_ATTRFIXEDVAL -49

  Attribute fixed value mismatch.
- #define RTERR_MULTIPLE -50

  Multiple errors occurred during an encode or decode operation.
- #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.
- #define RTERR_ADDRINUSE -52

  Address already in use.
- #define RTERR_CONNRESET -53

  Remote connection was reset.
- #define RTERR_UNREACHABLE -54

  Network failure.
- #define RTERR_NOCONN -55

  Not connected.
- #define RTERR_CONNREFUSED -56

  Connection refused.
- #define RTERR_INVSOCKOPT -57

  Invalid option.
- #define RTERR_SOAPFAULT -58

  This error is returned when decoded SOAP envelope is fault message.
- #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.
- #define RTERR_NOTSUPP -60 /∗ feature is not supported ∗/

  Feature is not supported.
- #define RTERR_UNBAL -61

  Unbalanced structure.
- #define RTERR_EXPNAME -62

  Expected name.
- #define RTERR_UNICODE -63

  Invalid Unicode sequence.
- #define RTERR_INVBOOL -64

  Invalid boolean keyword.
- #define RTERR_INVNULL -65

Invalid null keyword.

- #define RTERR_INVLEN -66

  Invalid length.
- #define RTERR_UNKNOWNIE -67

  Unknown information element.
- #define RTERR_NOTALIGNED -68

  Not aligned error.
- #define RTERR_EXTRDATA -69

  Extraneous data.
- #define RTERR_INVMAC -70

  Invalid Message Authentication Code.
- #define RTERR_NOSECPARAMS -71

  No security parameters provided.
- #define RTERR_COPYFAIL -72

  Copy failed.
- #define RTERR_PARSEFAIL -73

  Parse failed.
- #define RTERR_VALCMPERR -74

  Value comparison error.
- #define RTERR_BUFCMPERR -75

  Buffer comparison error.
- #define RTERR_INVBITS -76

  Invalid bit string error.
- #define RTERR_RLM -77

  RLM error encounterd.
- #define RTERR_NOCODEC -78

  No codec.
- #define RTERR_WOULDBLOCK -79

  A non-blocking socket could not return any data without blocking.
- #define RTERR_NODATA -80

  CDR or message file contains no data records.
- #define RTERR_MBEDTLS -81

  Error returned from MBEDTLS function.
- #define RTERR_INTOVFLOW -82

  Integer value overflow.
- #define RTERR_ABORT_REQUESTED -83

  User event handler calls for abort.

5.10.1 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.

5.10.2 Macro Definition Documentation

5.10.2.1 RT_OK_FRAG

```
#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.

Definition at line 53 of file rtxErrCodes.h.

5.10.2.2 RTERR_ADDRINUSE

```
#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.

Definition at line 471 of file rtxErrCodes.h.

5.10.2.3 RTERR_ATTRFIXEDVAL

```
#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.

Definition at line 444 of file rtxErrCodes.h.

5.10.2.4 RTERR_ATTRMISRQ

```
#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.

Definition at line 347 of file rtxErrCodes.h.

5.10.2.5 RTERR_BADVALUE

```
#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).

Definition at line 170 of file rtxErrCodes.h.

5.10.2.6 RTERR_BUFCMPERR

```
#define RTERR_BUFCMPERR -75
```

Buffer comparison error.

This error is raised when a comparison operation is done on two buffers and they are not equal.

Definition at line 630 of file rtxErrCodes.h.

5.10.2.7 RTERR_BUFOVFLW

```
#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.

Definition at line 60 of file rtxErrCodes.h.

5.10.2.8 RTERR_CONNREFUSED

```
#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.

Definition at line 495 of file rtxErrCodes.h.

5.10.2.9 RTERR_CONNRESET

```
#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.

Definition at line 477 of file rtxErrCodes.h.

5.10.2.10 RTERR_CONSVIO

```
#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.

Definition at line 185 of file rtxErrCodes.h.

5.10.2.11 RTERR_COPYFAIL

```
#define RTERR_COPYFAIL -72
```

Copy failed.

This occurs when generated copy functions are unable to complete a copy operation due to a runtime library failure.

Definition at line 612 of file rtxErrCodes.h.

5.10.2.12 RTERR_DECATTRFAIL

```
#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.

Definition at line 412 of file rtxErrCodes.h.

5.10.2.13 RTERR_DECELEMFAIL

```
#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.

Definition at line 405 of file rtxErrCodes.h.

5.10.2.14 RTERR_ENDOFBUF

```
#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.

Definition at line 67 of file rtxErrCodes.h.

5.10.2.15 RTERR_ENDOFFILE

```
#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 END↩ OFBUF error code described above except that in this case, decoding is being done from a file stream instead of from a memory buffer.

Definition at line 193 of file rtxErrCodes.h.

5.10.2.16 RTERR_EXPIRED

```
#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.

Definition at line 375 of file rtxErrCodes.h.

5.10.2.17 RTERR_EXPNAME

```
#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.

Definition at line 543 of file rtxErrCodes.h.

5.10.2.18 RTERR_EXTRDATA

```
#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.

Definition at line 592 of file rtxErrCodes.h.

5.10.2.19 RTERR_FAILED

```
#define RTERR_FAILED -48
```

General failure.

Low level call returned error.

Definition at line 433 of file rtxErrCodes.h.

5.10.2.20 RTERR_FILNOTFOU

```
#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.

Definition at line 283 of file rtxErrCodes.h.

5.10.2.21 RTERR_HOSTNOTFOU

```
#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.

Definition at line 354 of file rtxErrCodes.h.

5.10.2.22 RTERR_HTTPERR

```
#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.

Definition at line 362 of file rtxErrCodes.h.

5.10.2.23 RTERR_IDNOTFOU

```
#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.

Definition at line 77 of file rtxErrCodes.h.

5.10.2.24 RTERR_INVATTR

```
#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.

Definition at line 322 of file rtxErrCodes.h.

5.10.2.25 RTERR_INVBASE64

```
#define RTERR_INVBASE64 -31
```

Invalid Base64 encoding.

This status code is returned when an error is detected in decoding base64 data.

Definition at line 303 of file rtxErrCodes.h.

5.10.2.26 RTERR_INVBITS

```
#define RTERR_INVBITS -76
```

Invalid bit string error.

This error is raised when a bit string is decoded that contains bits that have not been set to zero.

Definition at line 636 of file rtxErrCodes.h.

5.10.2.27 RTERR_INVBOOL

```
#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.

Definition at line 557 of file rtxErrCodes.h.

5.10.2.28 RTERR_INVCHAR

```
#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.

Definition at line 254 of file rtxErrCodes.h.

5.10.2.29 RTERR_INVENUM

```
#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.

Definition at line 84 of file rtxErrCodes.h.

5.10.2.30 RTERR_INVFORMAT

```
#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.

Definition at line 224 of file rtxErrCodes.h.

5.10.2.31 RTERR_INVHEXS

```
#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).

Definition at line 143 of file rtxErrCodes.h.

5.10.2.32 RTERR_INVLEN

```
#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.

Definition at line 572 of file rtxErrCodes.h.

5.10.2.33 RTERR_INVMAC

```
#define RTERR_INVMAC -70
```

Invalid Message Authentication Code.

This error is returned when a given message's MAC is not the expected value.

Definition at line 598 of file rtxErrCodes.h.

5.10.2.34 RTERR_INVMSGBUF

```
#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.

Definition at line 398 of file rtxErrCodes.h.

5.10.2.35 RTERR_INVNULL

```
#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.

Definition at line 564 of file rtxErrCodes.h.

5.10.2.36 RTERR_INVOCCUR

```
#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.

Definition at line 390 of file rtxErrCodes.h.

5.10.2.37 RTERR_INVOPT

```
#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.

Definition at line 128 of file rtxErrCodes.h.

5.10.2.38 RTERR_INVPARAM

```
#define RTERR_INVPARAM -20
```

Invalid parameter passed to a function or 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.

Definition at line 215 of file rtxErrCodes.h.

5.10.2.39 RTERR_INVREAL

```
#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).

Definition at line 151 of file rtxErrCodes.h.

5.10.2.40 RTERR_INVSOCKET

```
#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.

Definition at line 311 of file rtxErrCodes.h.

5.10.2.41 RTERR_INVSOCKOPT

```
#define RTERR_INVSOCKOPT -57
```

Invalid option.

This status code is returned when an invalid option is passed to socket.

Definition at line 501 of file rtxErrCodes.h.

5.10.2.42 RTERR_INVUTF8

```
#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.

Definition at line 200 of file rtxErrCodes.h.

5.10.2.43 RTERR_MULTIPLE

```
#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.

Definition at line 454 of file rtxErrCodes.h.

5.10.2.44 RTERR_NOCODEC

```
#define RTERR_NOCODEC -78
```

No codec.

This error is returned when the user has configured ASN1C to not generate a decoder for a type, and then that type appears in an encoding.

Definition at line 647 of file rtxErrCodes.h.

5.10.2.45 RTERR_NOCONN

```
#define RTERR_NOCONN -55
```

Not connected.

This status code is returned when an operation is issued on an unconnected socket.

Definition at line 489 of file rtxErrCodes.h.

5.10.2.46 RTERR_NOMEM

```
#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.

Definition at line 135 of file rtxErrCodes.h.

5.10.2.47 RTERR_NOSECPARAMS

```
#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.

Definition at line 606 of file rtxErrCodes.h.

5.10.2.48 RTERR_NOTALIGNED

```
#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.

Definition at line 586 of file rtxErrCodes.h.

5.10.2.49 RTERR_NOTINIT

```
#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).

Definition at line 234 of file rtxErrCodes.h.

5.10.2.50 RTERR_NOTINSET

```
#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.

Definition at line 110 of file rtxErrCodes.h.

5.10.2.51 RTERR_NOTSUPP

```
#define RTERR_NOTSUPP -60 /* feature is not supported */
```

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.

Definition at line 528 of file rtxErrCodes.h.

5.10.2.52 RTERR_NOTYPEINFO

```
#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.

Definition at line 465 of file rtxErrCodes.h.

5.10.2.53 RTERR_NULLPTR

```
#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.

Definition at line 428 of file rtxErrCodes.h.

5.10.2.54 RTERR_OUTOFBND

```
#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.

Definition at line 207 of file rtxErrCodes.h.

5.10.2.55 RTERR_PARSEFAIL

```
#define RTERR_PARSEFAIL -73
```

Parse failed.

This error is raised when an application receives a text or binary message it is unable to parse.

Definition at line 618 of file rtxErrCodes.h.

5.10.2.56 RTERR_PATMATCH

```
#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.

Definition at line 340 of file rtxErrCodes.h.

5.10.2.57 RTERR_READERR

```
#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.

Definition at line 290 of file rtxErrCodes.h.

5.10.2.58 RTERR_REGEXP

```
#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.

Definition at line 331 of file rtxErrCodes.h.

5.10.2.59 RTERR_SEQORDER

```
#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.

Definition at line 276 of file rtxErrCodes.h.

5.10.2.60 RTERR_SEQOVFLW

```
#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.

Definition at line 118 of file rtxErrCodes.h.

5.10.2.61 RTERR_SETDUPL

```
#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.

Definition at line 92 of file rtxErrCodes.h.

5.10.2.62 RTERR_SETMISRQ

```
#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.

Definition at line 100 of file rtxErrCodes.h.

5.10.2.63 RTERR_SOAPERR

```
#define RTERR_SOAPERR -39
```

SOAP error.

This status code when an error is detected when trying to execute a SOAP operation.

Definition at line 368 of file rtxErrCodes.h.

5.10.2.64 RTERR_STRMINUSE

```
#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.

Definition at line 421 of file rtxErrCodes.h.

5.10.2.65 RTERR_STROVFLW

```
#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.

Definition at line 161 of file rtxErrCodes.h.

5.10.2.66 RTERR_TOOBIG

```
#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.

Definition at line 246 of file rtxErrCodes.h.

5.10.2.67 RTERR_TOODEEP

```
#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.

Definition at line 177 of file rtxErrCodes.h.

5.10.2.68 RTERR_UNBAL

```
#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.

Definition at line 537 of file rtxErrCodes.h.

5.10.2.69 RTERR_UNEXPELEM

```
#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.

Definition at line 382 of file rtxErrCodes.h.

5.10.2.70 RTERR_UNICODE

```
#define RTERR_UNICODE -63
```

Invalid Unicode sequence.

The sequence of characters received did not comprise a valid unicode character.

Definition at line 549 of file rtxErrCodes.h.

5.10.2.71 RTERR_UNKNOWNIE

```
#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.

Definition at line 579 of file rtxErrCodes.h.

5.10.2.72 RTERR_UNREACHABLE

```
#define RTERR_UNREACHABLE -54
```

Network failure.

This status code is returned when the network or host is down or otherwise unreachable.

Definition at line 483 of file rtxErrCodes.h.

5.10.2.73 RTERR_VALCMPERR

```
#define RTERR_VALCMPERR -74
```

Value comparison error.

This error is raised when a comparison operation is done on two values and they are not equal.

Definition at line 624 of file rtxErrCodes.h.

5.10.2.74 RTERR_WRITEERR

```
#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.

Definition at line 297 of file rtxErrCodes.h.

5.10.2.75 RTERR_XMLPARSE

```
#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.

Definition at line 268 of file rtxErrCodes.h.

5.10.2.76 RTERR_XMLSTATE

```
#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.

Definition at line 260 of file rtxErrCodes.h.

5.11 Error Formatting and Print Functions

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.

Macros

- #define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat,__FILE__,__LINE__)

  This macro is used to log a run-time error in the context.
- #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }

  This macro is used to check an assertion.
- #define OSRTCHECKPARAM(condition) if (condition) { /∗ do nothing ∗/ }

  This macro check a condition but takes no action.
- #define LOG_RTERR_AND_FREE_MEM(ctxt_p, stat, mem_p) rtxMemFreePtr ((ctxt_p),(mem_p)), LOG_RTERR(ctxt_p, stat)

  This logs an error to a global context and frees a memory pointer allocated for encoding or decoding.

Functions

- EXTERNRT OSBOOL rtxErrAddCtxtBufParm (OSCTXT ∗pctxt)

  This function adds the contents of the context buffer to the error information structure in the context.
- EXTERNRT OSBOOL rtxErrAddDoubleParm (OSCTXT ∗pctxt, double errParm)

  This function adds a double parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddErrorTableEntry (const char ∗const ∗ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)

  This function adds a set of error codes to the global error table.
- EXTERNRT OSBOOL rtxErrAddElemNameParm (OSCTXT ∗pctxt)

  This function adds an element name parameter to the context error information structure.
- EXTERNRT OSBOOL rtxErrAddIntParm (OSCTXT ∗pctxt, int errParm)

  This function adds an integer parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddInt64Parm (OSCTXT ∗pctxt, OSINT64 errParm)

  This function adds a 64-bit integer parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddSizeParm (OSCTXT ∗pctxt, OSSIZE errParm)

  This function adds a size-typed parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddStrParm (OSCTXT ∗pctxt, const char ∗pErrParm)

  This function adds a character string parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddStrnParm (OSCTXT ∗pctxt, const char ∗pErrParm, OSSIZE nchars)

  This function adds a given number of characters from a character string parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddUIntParm (OSCTXT ∗pctxt, unsigned int errParm)

  This function adds an unsigned integer parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddUInt64Parm (OSCTXT ∗pctxt, OSUINT64 errParm)

  This function adds an unsigned 64-bit integer parameter to an error information structure.
- EXTERNRT void rtxErrAssertionFailed (const char ∗conditionText, int lineNo, const char ∗fileName)

  This function is used to record an assertion failure.
- EXTERNRT const char ∗ rtxErrFmtMsg (OSRTErrInfo ∗pErrInfo, char ∗bufp, OSSIZE bufsiz)

This function formats a given error structure from the context into a finished status message including substituted parameters.

- EXTERNRT void rtxErrFreeParms (OSCTXT ∗pctxt)

  This function is used to free dynamic memory that was used in the recording of error parameters.
- EXTERNRT char ∗ rtxErrGetText (OSCTXT ∗pctxt, char ∗pBuf, OSSIZE ∗pBufSize)

  This function returns error text in a memory buffer.
- EXTERNRT char ∗ rtxErrGetTextBuf (OSCTXT ∗pctxt, char ∗pbuf, OSSIZE bufsiz)

  This function returns error text in the given fixed-size memory buffer.
- EXTERNRT char ∗ rtxErrGetMsgText (OSCTXT ∗pctxt)

  This function returns error message text in a memory buffer.
- EXTERNRT char ∗ rtxErrGetMsgTextBuf (OSCTXT ∗pctxt, char ∗pbuf, OSSIZE bufsiz)

  This function returns error message text in a static memory buffer.
- EXTERNRT OSRTErrInfo ∗ rtxErrNewNode (OSCTXT ∗pctxt)

  This function creates a new empty error record for the passed context.
- EXTERNRT void rtxErrInit (OSVOIDARG)

  This function is a one-time initialization function that must be called before any other error processing functions can be called.
- EXTERNRT int rtxErrReset (OSCTXT ∗pctxt)

  This function is used to reset the error state recorded in the context to successful.
- EXTERNRT int rtxErrResetErrInfo (OSCTXT ∗pctxt)

  This function is used to reset the error state recorded in the context to successful.
- EXTERNRT void rtxErrLogUsingCB (OSCTXT ∗pctxt, OSErrCbFunc cb, void ∗cbArg_p)

  This function allows error information to be logged using a user-defined callback routine.
- EXTERNRT void rtxErrPrint (OSCTXT ∗pctxt)

  This function is used to print the error information stored in the context to the standard output device.
- EXTERNRT void rtxErrPrintElement (OSRTErrInfo ∗pErrInfo)

  This function is used to print the error information stored in the error information element to the standard output device.
- EXTERNRT int rtxErrSetData (OSCTXT ∗pctxt, int status, const char ∗module, int lineno)

  This function is used to record an error in the context structure.
- EXTERNRT int rtxErrSetNewData (OSCTXT ∗pctxt, int status, const char ∗module, int lineno)

  This function is used to record an error in the context structure.
- EXTERNRT void rtxErrSetNonFatal (OSCTXT ∗pctxt)

  This marks the last error recorded in the context as non-fatal.
- EXTERNRT int rtxErrCheckNonFatal (OSCTXT ∗pctxt)

  This function checks the context structure for non-fatal errors.
- EXTERNRT int rtxErrGetFirstError (const OSCTXT ∗pctxt)

  This function returns the error code, stored in the first error record.
- EXTERNRT int rtxErrGetLastError (const OSCTXT ∗pctxt)

  This function returns the error code, stored in the last error record.
- EXTERNRT OSSIZE rtxErrGetErrorCnt (const OSCTXT ∗pctxt)

  This function returns the total number of error records, including non-fatal errors.
- EXTERNRT int rtxErrGetStatus (const OSCTXT ∗pctxt)

  This function returns the status value from the context.
- EXTERNRT int rtxErrResetLastErrors (OSCTXT ∗pctxt, OSSIZE errorsToReset)

  This function resets last 'errorsToReset‘ errors in the context.
- EXTERNRT int rtxErrCopy (OSCTXT ∗pDestCtxt, const OSCTXT ∗pSrcCtxt)

  This function copies error information from one context into another.

- EXTERNRT int rtxErrAppend (OSCTXT ∗pDestCtxt, const OSCTXT ∗pSrcCtxt)

  This function appends error information from one context into another.
- EXTERNRT int rtxErrInvUIntOpt (OSCTXT ∗pctxt, OSUINT32 ident)

  This function create an 'invalid option' error (RTERR_INVOPT) in the context using an unsigned integer parameter.

5.11.1 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.

5.11.2 Macro Definition Documentation

5.11.2.1 LOG_RTERR

```
#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`

Definition at line 79 of file rtxError.h.

5.11.2.2 LOG_RTERR_AND_FREE_MEM

```
#define LOG_RTERR_AND_FREE_MEM(
            ctxt_p,
            stat,
            mem_p ) rtxMemFreePtr ((ctxt_p),(mem_p)), LOG_RTERR(ctxt_p, stat)
```

This logs an error to a global context and frees a memory pointer allocated for encoding or decoding.

Parameters

ctxt_p	A pointer to the main context data structure.
stat	The error status.
mem↩ _p	The memory pointer allocated for encoding/decoding.

Returns

The result of logging the error to the global context.

Definition at line 149 of file rtxError.h.

5.11.2.3 OSRTASSERT

```
#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)")

Definition at line 104 of file rtxError.h.

5.11.2.4 OSRTCHECKPARAM

```
#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)")

Definition at line 113 of file rtxError.h.

5.11.3 Function Documentation

5.11.3.1 rtxErrAddCtxtBufParm()

```
EXTERNRT 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).

5.11.3.2 rtxErrAddDoubleParm()

```
EXTERNRT 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).

5.11.3.3 rtxErrAddElemNameParm()

```
EXTERNRT 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).

5.11.3.4 rtxErrAddErrorTableEntry()

```
EXTERNRT 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).

5.11.3.5 rtxErrAddInt64Parm()

```
EXTERNRT 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).

5.11.3.6 rtxErrAddIntParm()

```
EXTERNRT 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).

5.11.3.7 rtxErrAddSizeParm()

```
EXTERNRT OSBOOL rtxErrAddSizeParm (
             OSCTXT * pctxt,
             OSSIZE errParm )
```

This function adds a size-typed 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).

5.11.3.8 rtxErrAddStrnParm()

```
EXTERNRT OSBOOL rtxErrAddStrnParm (
             OSCTXT * pctxt,
             const char * pErrParm,
             OSSIZE 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).

5.11.3.9 rtxErrAddStrParm()

```
EXTERNRT 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).

5.11.3.10 rtxErrAddUInt64Parm()

```
EXTERNRT 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).

5.11.3.11 rtxErrAddUIntParm()

```
EXTERNRT 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).

5.11.3.12 rtxErrAppend()

```
EXTERNRT 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

5.11.3.13 rtxErrAssertionFailed()

```
EXTERNRT 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.

5.11.3.14 rtxErrCheckNonFatal()

```
EXTERNRT int rtxErrCheckNonFatal (
             OSCTXT * pctxt )
```

This function checks the context structure for non-fatal errors.

If any such errors are found, the function will return the error status if they are all the same error. If there are at least two different error conditions, the function will return RTERR_MULTIPLE. If there are no error conditions, the function will return zero.

Parameters

pctxt	A pointer to a context structure.

Returns

An error status or zero.

5.11.3.15 rtxErrCopy()

```
EXTERNRT 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

5.11.3.16 rtxErrFmtMsg()

```
EXTERNRT const char* rtxErrFmtMsg (
            OSRTErrInfo * pErrInfo,
            char * bufp,
            OSSIZE 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).

5.11.3.17 rtxErrFreeParms()

```
EXTERNRT 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.

5.11.3.18 rtxErrGetErrorCnt()

```
EXTERNRT OSSIZE rtxErrGetErrorCnt (
             const OSCTXT * pctxt )
```

This function returns the total number of error records, including non-fatal errors.

Parameters

pctxt	A pointer to a context structure.

Returns

The total number of error records in the context.

5.11.3.19 rtxErrGetFirstError()

```
EXTERNRT 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.

5.11.3.20 rtxErrGetLastError()

```
EXTERNRT 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.

5.11.3.21 rtxErrGetMsgText()

```
EXTERNRT char* rtxErrGetMsgText (
            OSCTXT * pctxt )
```

This function returns error message text in a memory buffer.

It only returns the formatted error message, not the error status nor stack trace information. Memory is dynamically allocated using the rtxMemAlloc function. It should be freed using rtxMemFreePtr or it will be freed when the context is freed.

Parameters

pctxt	A pointer to a context structure.

Returns

A pointer to a buffer with error text. Dynamic memory will be allocated for this buffer using rtxMemAlloc. It should be freed using rtxMemFreePtr.

5.11.3.22 rtxErrGetMsgTextBuf()

```
EXTERNRT char* rtxErrGetMsgTextBuf (
            OSCTXT * pctxt,
            char * pbuf,
            OSSIZE bufsiz )
```

This function returns error message text in a static memory buffer.

It only returns the formatted error message, not the error status nor stack trace information. If the formatted 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).

5.11.3.23 rtxErrGetStatus()

```
EXTERNRT int rtxErrGetStatus (
             const OSCTXT * pctxt )
```

This function returns the status value from the context.

It examines the error list to see how many fatal errors were logged. If none, OK (zero) is returned; if one, then the status value in that single error record is returned; if more than one, the special code RTERR_MULTIPLE is returned to indicate that multiple errors occurred. Non-fatal errors are entirely ignored.

Parameters

pctxt	A pointer to a context structure.

Returns

Status code corresponding to errors in the context.

5.11.3.24 rtxErrGetText()

```
EXTERNRT char* rtxErrGetText (
             OSCTXT * pctxt,
             char * pBuf,
             OSSIZE * 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. If pBuf is not null, this is expcted to be set and hold the size of the 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.

5.11.3.25 rtxErrGetTextBuf()

```
EXTERNRT char* rtxErrGetTextBuf (
             OSCTXT * pctxt,
             char * pbuf,
             OSSIZE 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).

5.11.3.26 rtxErrInit()

```
EXTERNRT void rtxErrInit (
             OSVOIDARG  )
```

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.

5.11.3.27 rtxErrInvUIntOpt()

```
EXTERNRT 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.

5.11.3.28 rtxErrLogUsingCB()

```
EXTERNRT 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.

5.11.3.29 rtxErrNewNode()

```
EXTERNRT OSRTErrInfo* rtxErrNewNode (
             OSCTXT * pctxt )
```

This function creates a new empty error record for the passed context.

`rtxErrSetData` function call with bAllocNew = FALSE should be used to set the data for this node. The new record will have fatal == TRUE.

Parameters

pctxt	A pointer to a context structure.

Returns

A pointer to a newly allocated error record; NULL, if error occurred.

5.11.3.30 rtxErrPrint()

```
EXTERNRT 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.

5.11.3.31 rtxErrPrintElement()

```
EXTERNRT 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.

5.11.3.32 rtxErrReset()

```
EXTERNRT 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.

5.11.3.33 rtxErrResetErrInfo()

```
EXTERNRT int rtxErrResetErrInfo (
             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. This version differs from rtxErrReset in that it only resets error information within the context and not the element name and container stacks.

Parameters

pctxt	A pointer to a context structure.

5.11.3.34 rtxErrResetLastErrors()

```
EXTERNRT int rtxErrResetLastErrors (
             OSCTXT * pctxt,
             OSSIZE 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

5.11.3.35 rtxErrSetData()

```
EXTERNRT 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. If no error has been logged, a new log entry is created. The new entry will have fatal = TRUE.

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.

5.11.3.36 rtxErrSetNewData()

```
EXTERNRT 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 with fatal = TRUE.

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.

5.11.3.37 rtxErrSetNonFatal()

```
EXTERNRT void rtxErrSetNonFatal (
            OSCTXT * pctxt )
```

This marks the last error recorded in the context as non-fatal.

Non-fatal errors are ignored by rtxErrGetStatus but are otherwise treated the same as any other error. This is useful when recording, for example, canonical violations, which are errors but which don't force an operation to abort.

5.12 Hex Dump Functions

Function for printing binary data in hexadecimal text format and for doing binary to hexadecimal text conversion.

Functions

- EXTERNRT int rtxByteToHexChar (OSOCTET byte, char ∗buf, OSSIZE bufsize)

  This function converts a byte value into its hex string equivalent.
- EXTERNRT int rtxByteToHexCharWithPrefix (OSOCTET byte, char ∗buf, OSSIZE bufsize, const char ∗prefix)

  This function converts a byte value into its hex string equivalent.
- EXTERNRT int 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.
- EXTERNRT void rtxHexDumpToFile (FILE ∗fp, const OSOCTET ∗data, OSSIZE numocts)

  This function outputs a hexadecimal dump of the current buffer contents to a file.
- EXTERNRT void rtxHexDumpToFileEx (FILE ∗fp, const OSOCTET ∗data, OSSIZE numocts, OSSIZE bytesPer←Unit)

  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.
- EXTERNRT void rtxHexDumpToFileExNoAscii (FILE ∗fp, const OSOCTET ∗data, OSSIZE numocts, OSSIZE 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.
- EXTERNRT int rtxHexDumpToNamedFileNoAscii (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.
- EXTERNRT void rtxHexDump (const OSOCTET ∗data, OSSIZE numocts)

  This function outputs a hexadecimal dump of the current buffer contents to stdout.
- EXTERNRT void rtxHexDumpEx (const OSOCTET ∗data, OSSIZE numocts, OSSIZE 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.
- EXTERNRT int rtxHexDumpToString (const OSOCTET ∗data, OSSIZE numocts, char ∗buffer, OSSIZE buffer←Index, OSSIZE bufferSize)

  This function formats a hexadecimal dump of the current buffer contents to a string.
- EXTERNRT int rtxHexDumpToStringEx (const OSOCTET ∗data, OSSIZE numocts, char ∗buffer, OSSIZE buffer←Index, OSSIZE bufferSize, OSSIZE 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.
- EXTERNRT int rtxHexDumpFileContents (const char ∗inFilePath)

  This function outputs a hexadecimal dump of the contents of the named file to stdout.
- EXTERNRT int rtxHexDumpFileContentsToFile (const char ∗inFilePath, const char ∗outFilePath)

  This function outputs a hexadecimal dump of the contents of the named file to a text file.
- EXTERNRT char ∗ rtxHexDiffToDynString (OSCTXT ∗pctxt, const OSOCTET ∗pdata1, const OSOCTET ∗pdata2, OSSIZE numocts)

  This function generates a differences report between two binary data buffers.

5.12.1 Detailed Description

Function for printing binary data in hexadecimal text format and for doing binary to hexadecimal text conversion.

5.12.2 Function Documentation

5.12.2.1 rtxByteToHexChar()

```
EXTERNRT 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.

5.12.2.2 rtxByteToHexCharWithPrefix()

```
EXTERNRT int rtxByteToHexCharWithPrefix (
             OSOCTET byte,
             char * buf,
             OSSIZE bufsize,
             const char * prefix )
```

This function converts a byte value into its hex string equivalent.

The hex string for this function is prefixed with the given parameter.

Parameters

byte	Byte to format.
buf	Output buffer.
bufsize	Output buffer size.
prefix	The string prefix.

5.12.2.3 rtxHexDiffToDynString()

```
EXTERNRT char* rtxHexDiffToDynString (
             OSCTXT * pctxt,
             const OSOCTET * pdata1,
             const OSOCTET * pdata2,
             OSSIZE numocts )
```

This function generates a differences report between two binary data buffers.

The buffers are assumed to each contain the same number of bytes. The result of the comparison operation is returned in a dynamic allocated using the rtxMemAlloc function. The buffer may be freed using the rtxMemFreePtr function.

Parameters

pctxt	Pointer to context structure.
pdata1	Pointer to first binary buffer to compare.
pdata2	Pointer to second binary buffer to compare.
numocts	Number of bytes to compare.

Returns

Result of comparison in dynamically allocated string buffer. Null means buffers have no differences. The format of the output is '[x]yy != zz, ...' where x is index and yy and zz are byte values.

5.12.2.4 rtxHexDump()

```
EXTERNRT 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.

5.12.2.5 rtxHexDumpEx()

```
EXTERNRT void rtxHexDumpEx (
            const OSOCTET * data,
            OSSIZE numocts,
            OSSIZE 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).

5.12.2.6 rtxHexDumpFileContents()

```
EXTERNRT int 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.

5.12.2.7 rtxHexDumpFileContentsToFile()

```
EXTERNRT int 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.

5.12.2.8 rtxHexDumpToFile()

```
EXTERNRT 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

5.12.2.9 rtxHexDumpToFileEx()

```
EXTERNRT void rtxHexDumpToFileEx (
             FILE * fp,
             const OSOCTET * data,
             OSSIZE numocts,
             OSSIZE 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).

5.12.2.10 rtxHexDumpToFileExNoAscii()

```
EXTERNRT void rtxHexDumpToFileExNoAscii (
             FILE * fp,
             const OSOCTET * data,
             OSSIZE numocts,
             OSSIZE 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.

This function never contains an ASCII dump.

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).

5.12.2.11 rtxHexDumpToNamedFile()

```
EXTERNRT int 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

Returns

Status of operation: 0 for success or negative error code.

5.12.2.12 rtxHexDumpToNamedFileNoAscii()

```
EXTERNRT int rtxHexDumpToNamedFileNoAscii (
             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. The dump in this case has no ASCII part which makes it easier to import into hex editor tools.

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.

Returns

Status of operation: 0 for success or negative error code.

5.12.2.13 rtxHexDumpToString()

```
EXTERNRT 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.

5.12.2.14 rtxHexDumpToStringEx()

```
EXTERNRT int rtxHexDumpToStringEx (
             const OSOCTET * data,
             OSSIZE numocts,
             char * buffer,
             OSSIZE bufferIndex,
             OSSIZE bufferSize,
             OSSIZE 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.

5.13 Integer Stack Utility Functions

This is a simple stack structure with supporting push, pop, and peek functions.

Classes

- struct _OSRTIntStack

  This is the main stack structure.

Macros

- #define OSRTISTK_DEFAULT_CAPACITY 100

  This is the default capacity that is used if zero is passed as the capacity argument to rtxIntStackInit.
- #define rtxIntStackIsEmpty(stack) (OSBOOL)((stack).index == 0)

  This macro tests if the stack is empty.

Functions

- EXTERNRT int rtxIntStackInit (OSCTXT ∗pctxt, OSRTIntStack ∗pstack, size_t capacity)

  This function initializes a stack structure.
- EXTERNRT int rtxIntStackPush (OSRTIntStack ∗pstack, OSINT32 value)

  This function pushes an item onto the stack.
- EXTERNRT int rtxIntStackPeek (OSRTIntStack ∗pstack, OSINT32 ∗pvalue)

  This functions returns the data item on the top of the stack.
- EXTERNRT int rtxIntStackPop (OSRTIntStack ∗pstack, OSINT32 ∗pvalue)

  This functions pops the data item on the top of the stack.

5.13.1 Detailed Description

This is a simple stack structure with supporting push, pop, and peek functions.

5.13.2 Macro Definition Documentation

5.13.2.1 rtxIntStackIsEmpty

```
#define rtxIntStackIsEmpty(
              stack ) (OSBOOL)((stack).index == 0)
```

This macro tests if the stack is empty.

Parameters

stack	Stack structure variable to be tested.

Definition at line 122 of file rtxIntStack.h.

5.13.3 Function Documentation

5.13.3.1 rtxIntStackInit()

```
EXTERNRT int rtxIntStackInit (
             OSCTXT * pctxt,
             OSRTIntStack * pstack,
             size_t capacity )
```

This function initializes a stack structure.

It allocates the initial amount of memory required to store data and sets all working variables to their initil state.

Parameters

pctxt	A pointer to the context with which the stack is associated.
pstack	A pointer to a stack structure to be initialized.
capacity	Initial capacity of the stack. This is the number of integer values that can be stored before the stack is expanded. Each expansion doubles the initial capacity value.

Returns

Completion status of operation:

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

5.13.3.2 rtxIntStackPeek()

```
EXTERNRT int rtxIntStackPeek (
             OSRTIntStack * pstack,
             OSINT32 * pvalue )
```

This functions returns the data item on the top of the stack.

Parameters

pstack	A pointer to the stack structure.
pvalue	A pointer to a variable to store the integer value of the item at the top of the stack.

Returns

Status of peek operation:

- 0 (0) = success,
- RTERR_ENDOFBUF if stack is empty

5.13.3.3 rtxIntStackPop()

```
EXTERNRT int rtxIntStackPop (
             OSRTIntStack * pstack,
             OSINT32 * pvalue )
```

This functions pops the data item on the top of the stack.

Parameters

pstack	A pointer to the stack structure.
pvalue	A pointer to a variable to store the integer value of the item at the top of the stack.

Returns

Status of pop operation:

- 0 (0) = success,
- RTERR_ENDOFBUF if stack is empty

5.13.3.4 rtxIntStackPush()

```
EXTERNRT int rtxIntStackPush (
             OSRTIntStack * pstack,
             OSINT32 value )
```

This function pushes an item onto the stack.

Parameters

pstack	A pointer to the stack structure.
value	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.

5.14 Memory Buffer Management Functions

Memory buffer management functions handle the allocation, expansion, and deallocation of dynamic memory buffers used by some encode/decode functions.

Functions

- EXTERNRT int rtxMemBufAppend (OSRTMEMBUF ∗pMemBuf, const OSOCTET ∗pdata, OSSIZE nbytes)

  This function appends the data to the end of a memory buffer.
- EXTERNRT int rtxMemBufCut (OSRTMEMBUF ∗pMemBuf, OSSIZE fromOffset, OSSIZE nbytes)

  This function cuts off the part of memory buffer.
- EXTERNRT void rtxMemBufFree (OSRTMEMBUF ∗pMemBuf)

  This function frees the memory buffer.
- EXTERNRT OSOCTET ∗ rtxMemBufGetData (const OSRTMEMBUF ∗pMemBuf, int ∗length)

  This function returns the pointer to the used part of a memory buffer.
- EXTERNRT OSOCTET ∗ rtxMemBufGetDataExt (const OSRTMEMBUF ∗pMemBuf, OSSIZE ∗length)

  This function returns the pointer to the used part of a memory buffer.
- EXTERNRT OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF ∗pMemBuf)

  This function returns the length of the used part of a memory buffer.
- EXTERNRT void rtxMemBufInit (OSCTXT ∗pCtxt, OSRTMEMBUF ∗pMemBuf, OSSIZE segsize)

  This function initializes a memory buffer structure.
- EXTERNRT void rtxMemBufInitBuffer (OSCTXT ∗pCtxt, OSRTMEMBUF ∗pMemBuf, OSOCTET ∗buf, OSSIZE bufsize, OSSIZE segsize)

  This function assigns a static buffer to the memory buffer structure.
- EXTERNRT int rtxMemBufPreAllocate (OSRTMEMBUF ∗pMemBuf, OSSIZE nbytes)

  This function allocates a buffer with a predetermined amount of space.
- EXTERNRT void rtxMemBufReset (OSRTMEMBUF ∗pMemBuf)

  This function resets the memory buffer structure.
- EXTERNRT int rtxMemBufSet (OSRTMEMBUF ∗pMemBuf, OSOCTET value, OSSIZE nbytes)

  This function sets part of a memory buffer to a specified octet value.
- EXTERNRT OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF ∗pMemBuf, OSBOOL isExpandable)

  This function sets "isExpandable" flag for the memory buffer object.
- EXTERNRT 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.
- EXTERNRT OSSIZE rtxMemBufTrimW (OSRTMEMBUF ∗pMemBuf)

  This function trims white space of the memory buffer.

5.14.1 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.

5.14.2 Function Documentation

5.14.2.1 rtxMemBufAppend()

```
EXTERNRT 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.

5.14.2.2 rtxMemBufCut()

```
EXTERNRT 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.

5.14.2.3 rtxMemBufFree()

```
EXTERNRT 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.

5.14.2.4 rtxMemBufGetData()

```
EXTERNRT 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.

5.14.2.5 rtxMemBufGetDataExt()

```
EXTERNRT 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.

5.14.2.6 rtxMemBufGetDataLen()

```
EXTERNRT 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.

5.14.2.7 rtxMemBufInit()

```
EXTERNRT 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.

5.14.2.8 rtxMemBufInitBuffer()

```
EXTERNRT 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) and a non-zero segsize was specified, 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. If 0, when expansion is needed, an RTERR_NOMEM error will result.

5.14.2.9 rtxMemBufPreAllocate()

```
EXTERNRT 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.

5.14.2.10 rtxMemBufReset()

```
EXTERNRT 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.

5.14.2.11 rtxMemBufSet()

```
EXTERNRT 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 value to append to the end of the memory buffer.
nbytes	The number of times to append "value".

Returns

Completion status of operation:

- 0 = success,

- negative return value is error.

5.14.2.12 rtxMemBufSetExpandable()

```
EXTERNRT 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.

5.14.2.13 rtxMemBufSetUseSysMem()

```
EXTERNRT 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.

5.14.2.14 rtxMemBufTrimW()

```
EXTERNRT 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.

5.15 Memory Allocation Macros and Functions

Memory allocation functions and macros handle memory management for the C run-time.

Macros

- #define OSRTALLOCTYPE(pctxt, type) (type∗) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))

  This macro allocates a single element of the given type.
- #define OSRTALLOCTYPEZ(pctxt, type) (type∗) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))

  This macro allocates and zeros a single element of the given type.
- #define OSRTREALLOCARRAY(pctxt, pseqof, type)

  Reallocate an array.
- #define rtxMemAlloc(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap,nbytes)

  Allocate memory.
- #define rtxMemSysAlloc(pctxt, nbytes) rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)

  This macro makes a direct call to the configured system memory allocation function.
- #define rtxMemAllocZ(pctxt, nbytes) rtxMemHeapAllocZ(&(pctxt)->pMemHeap,nbytes)

  Allocate and zero memory.
- #define rtxMemSysAllocZ(pctxt, nbytes) rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)

  Allocate and zero memory.
- #define rtxMemRealloc(pctxt, mem_p, nbytes) rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, nbytes)

  Reallocate memory.
- #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.
- #define rtxMemFreePtr(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemSysFreePtr(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  This macro makes a direct call to the configured system memory free function.
- #define rtxMemAllocType(pctxt, ctype) (ctype∗)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type.
- #define rtxMemSysAllocType(pctxt, ctype) (ctype∗)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type.
- #define rtxMemAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type and zero memory.
- #define rtxMemSysAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type and zero memory.
- #define rtxMemFreeType(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemSysFreeType(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemAllocArray(pctxt, n, type) (type∗)rtxMemAllocArray2 (pctxt, n, sizeof(type), 0)

  Allocate a dynamic array.
- #define rtxMemSysAllocArray(pctxt, n, type) (type∗)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_SYSALLOC)

Allocate a dynamic array.

- #define rtxMemAllocArrayZ(pctxt, n, type) (type∗)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_ZEROAR↩ RAY)

  Allocate a dynamic array and zero memory.
- #define rtxMemFreeArray(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemSysFreeArray(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemReallocArray(pctxt, mem_p, n, type) (type∗)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, sizeof(type)∗n)

  Reallocate memory.
- #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.
- #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void∗)(ptr))

  This function increments the auto-pointer reference count.
- #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void∗)(ptr))

  This function decrements the auto-pointer reference count.
- #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void∗)(ptr))

  This function returns the reference count of the given pointer.
- #define rtxMemCheckPtr(pctxt, mem_p) rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Check memory pointer.
- #define rtxMemCheck(pctxt) rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)

  Check memory heap.
- #define rtxMemPrint(pctxt) rtxMemHeapPrint(&(pctxt)->pMemHeap)

  Print memory heap structure to stderr.
- #define rtxMemPrintWithFree(pctxt) rtxMemHeapPrintWithFree(&(pctxt)->pMemHeap)

  Print memory heap structure to stderr, the same as rtxMemPrint, but add details about the memory free list.
- #define rtxMemSetProperty(pctxt, propId, pProp) rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, p↩ Prop)

  Set memory heap property.

Functions

- EXTERNRT void rtxMemHeapPrint (void ∗∗ppvMemHeap)

  Print details about the memory heap.
- EXTERNRT void rtxMemHeapPrintWithFree (void ∗∗ppvMemHeap)

  Print the same details about the memory heap as rtxMemHeapPrint but add deatils about the free memory list.
- EXTERNRT int rtxMemHeapCreate (void ∗∗ppvMemHeap)

  This function creates a standard memory heap.
- EXTERNRT int rtxMemHeapCreateExt (void ∗∗ppvMemHeap, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

  This function creates a standard memory heap and sets the low-level memory functions to the specified values.
- EXTERNRT int rtxMemStaticHeapCreate (void ∗∗ppvMemHeap, void ∗pmem, OSSIZE memsize)

  This function creates a static memory heap.
- EXTERNRT int rtxMemHeapConvertStatic (void ∗∗ppvMemHeap, void ∗pmem, OSSIZE memsize)

  This function converts a standard memory heap to a static memory heap.

- EXTERNRT void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

  This function sets the pointers to standard allocation functions.
- EXTERNRT OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT ∗pctxt)

  This function returns the actual granularity of memory blocks in the context.
- EXTERNRT void rtxMemSetDefBlkSize (OSUINT32 blkSize)

  This function sets the minimum size and the granularity of memory blocks for newly created memory heaps.
- EXTERNRT OSUINT32 rtxMemGetDefBlkSize (OSVOIDARG)

  This function returns the actual granularity of memory blocks.
- EXTERNRT OSBOOL rtxMemHeapIsEmpty (OSCTXT ∗pctxt)

  This function determines if the memory heap defined in the give context is empty (i.e.
- EXTERNRT OSBOOL rtxMemIsZero (const void ∗pmem, OSSIZE memsiz)

  This helper function determines if an arbitrarily sized block of memory is set to zero.
- EXTERNRT void rtxMemFree (OSCTXT ∗pctxt)

  Free memory associated with a context.
- EXTERNRT void rtxMemReset (OSCTXT ∗pctxt)

  Reset memory associated with a context.

5.15.1 Detailed Description

Memory allocation functions and macros handle memory management for the C run-time.

Special algorithms are used for allocation and deallocation of memory to improve the run-time performance.

5.15.2 Macro Definition Documentation

5.15.2.1 OSRTALLOCTYPE

```
#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

Definition at line 77 of file rtxMemory.h.

5.15.2.2 OSRTALLOCTYPEZ

```
#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

Definition at line 86 of file rtxMemory.h.

5.15.2.3 OSRTREALLOCARRAY

```
#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

Definition at line 100 of file rtxMemory.h.

5.15.2.4 rtxMemAlloc

```
#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.

Definition at line 350 of file rtxMemory.h.

5.15.2.5 rtxMemAllocArray

```
#define rtxMemAllocArray(
            pctxt,
            n,
            type ) (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), 0)
```

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

Definition at line 577 of file rtxMemory.h.

5.15.2.6 rtxMemAllocArrayZ

```
#define rtxMemAllocArrayZ(
            pctxt,
            n,
            type ) (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_ZEROARRAY)
```

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

Definition at line 608 of file rtxMemory.h.

5.15.2.7 rtxMemAllocType

```
#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.

Definition at line 493 of file rtxMemory.h.

5.15.2.8 rtxMemAllocTypeZ

```
#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.

Definition at line 521 of file rtxMemory.h.

5.15.2.9 rtxMemAllocZ

```
#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.

Definition at line 375 of file rtxMemory.h.

5.15.2.10 rtxMemAutoPtrGetRefCount

```
#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.

Definition at line 704 of file rtxMemory.h.

5.15.2.11 rtxMemAutoPtrRef

```
#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.

Definition at line 680 of file rtxMemory.h.

5.15.2.12 rtxMemAutoPtrUnref

```
#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.

Definition at line 693 of file rtxMemory.h.

5.15.2.13 rtxMemCheck

```
#define rtxMemCheck(
              pctxt ) rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)
```

Check memory heap.

Parameters

pctxt	- Pointer to a context block

Definition at line 723 of file rtxMemory.h.

5.15.2.14 rtxMemCheckPtr

```
#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.

Definition at line 715 of file rtxMemory.h.

5.15.2.15 rtxMemFreeArray

```
#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.

Definition at line 622 of file rtxMemory.h.

5.15.2.16 rtxMemFreePtr

```
#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.

Definition at line 442 of file rtxMemory.h.

5.15.2.17 rtxMemFreeType

```
#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.

Definition at line 551 of file rtxMemory.h.

5.15.2.18 rtxMemNewAutoPtr

```
#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.

Definition at line 669 of file rtxMemory.h.

5.15.2.19 rtxMemPrint

```
#define rtxMemPrint(
             pctxt ) rtxMemHeapPrint(&(pctxt)->pMemHeap)
```

Print memory heap structure to stderr.

Parameters

pctxt	- Pointer to a context block

Definition at line 731 of file rtxMemory.h.

5.15.2.20 rtxMemPrintWithFree

```
#define rtxMemPrintWithFree(
             pctxt ) rtxMemHeapPrintWithFree(&(pctxt)->pMemHeap)
```

Print memory heap structure to stderr, the same as rtxMemPrint, but add details about the memory free list.

Parameters

pctxt	- Pointer to a context block

Definition at line 740 of file rtxMemory.h.

5.15.2.21 rtxMemRealloc

```
#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.

Definition at line 409 of file rtxMemory.h.

5.15.2.22 rtxMemReallocArray

```
#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.

Definition at line 653 of file rtxMemory.h.

5.15.2.23 rtxMemSetProperty

```
#define rtxMemSetProperty(
              pctxt,
              propId,
              pProp ) rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, pProp)
```

Set memory heap property.

Parameters

pctxt	- Pointer to a context block
prop↩ Id	- Property Id.
pProp	- Pointer to property value.

Definition at line 750 of file rtxMemory.h.

5.15.2.24 rtxMemSysAlloc

```
#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.

Definition at line 363 of file rtxMemory.h.

5.15.2.25 rtxMemSysAllocArray

```
#define rtxMemSysAllocArray(
              pctxt,
              n,
              type ) (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_SYSALLOC)
```

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

Definition at line 596 of file rtxMemory.h.

5.15.2.26 rtxMemSysAllocType

```
#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.

Definition at line 509 of file rtxMemory.h.

5.15.2.27 rtxMemSysAllocTypeZ

```
#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.

Definition at line 537 of file rtxMemory.h.

5.15.2.28 rtxMemSysAllocZ

```
#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.

Definition at line 391 of file rtxMemory.h.

5.15.2.29 rtxMemSysFreeArray

```
#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.

Definition at line 636 of file rtxMemory.h.

5.15.2.30 rtxMemSysFreePtr

```
#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.

Definition at line 455 of file rtxMemory.h.

5.15.2.31 rtxMemSysFreeType

```
#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.

Definition at line 565 of file rtxMemory.h.

5.15.2.32 rtxMemSysRealloc

```
#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.

Definition at line 428 of file rtxMemory.h.

5.15.3 Function Documentation

5.15.3.1 rtxMemFree()

```
EXTERNRT 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

5.15.3.2 rtxMemGetDefBlkSize()

```
EXTERNRT OSUINT32 rtxMemGetDefBlkSize (
             OSVOIDARG  )
```

This function returns the actual granularity of memory blocks.

Returns

The currently used minimum size and the granularity of memory blocks.

5.15.3.3 rtxMemHeapConvertStatic()

```
EXTERNRT int rtxMemHeapConvertStatic (
             void ** ppvMemHeap,
             void * pmem,
             OSSIZE memsize )
```

This function converts a standard memory heap to a static memory heap.

All allocations are done from the static block of memory that is provided. It is much faster than the standard management but has some limitations such as the inability to free individual pointer values. All memory in the block must be freed at once.

If memory was previously allocated using the given heap, it will be freed before conversion.

Parameters

ppvMemHeap	Pointer-to-pointer to heap variable. ppvMemHeap and ∗ppvMemHeap must not be null; otherwise, an error is returned.
pmem	Pointer to static memory block to use for allocations.
memsize	Sizeof the memory block in bytes.

Returns

Status of the creation operation: 0 = success or $<$ 0 for an error.

5.15.3.4 rtxMemHeapCreate()

```
EXTERNRT int rtxMemHeapCreate (
             void ** ppvMemHeap )
```

This function creates a standard memory heap.

It is invoked internally from the rtxInitContext function to create the heap in the context.

Parameters

ppvMemHeap	Pointer-to-pointer to variable to receive created object.

Returns

Status of the creation operation: 0 = success or RTERR-NOMEM if no memory available.

5.15.3.5 rtxMemHeapCreateExt()

```
EXTERNRT int rtxMemHeapCreateExt (
             void ** ppvMemHeap,
             OSMallocFunc malloc_func,
             OSReallocFunc realloc_func,
             OSFreeFunc free_func )
```

This function creates a standard memory heap and sets the low-level memory functions to the specified values.

It is invoked internally from the rtxInitContextExt function to create the heap in the context.

Parameters

ppvMemHeap	Pointer-to-pointer to variable to receive created object.
malloc_func	Pointer to memory allocation function.
realloc_func	Pointer to memory reallocation function.
free_func	Pointer to memory free function.

Returns

Status of the creation operation: 0 = success or RTERR-NOMEM if no memory available.

5.15.3.6 rtxMemHeapGetDefBlkSize()

```
EXTERNRT OSUINT32 rtxMemHeapGetDefBlkSize (
             OSCTXT * pctxt )
```

This function returns the actual granularity of memory blocks in the context.

Parameters

pctxt	Pointer to a context block.

5.15.3.7 rtxMemHeapIsEmpty()

```
EXTERNRT 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.

5.15.3.8 rtxMemIsZero()

```
EXTERNRT OSBOOL rtxMemIsZero (
             const void * pmem,
             OSSIZE 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

5.15.3.9 rtxMemReset()

```
EXTERNRT 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

5.15.3.10 rtxMemSetAllocFuncs()

```
EXTERNRT 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).

5.15.3.11 rtxMemSetDefBlkSize()

```
EXTERNRT 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.

5.15.3.12 rtxMemStaticHeapCreate()

```
EXTERNRT int rtxMemStaticHeapCreate (
             void ** ppvMemHeap,
             void * pmem,
             OSSIZE memsize )
```

This function creates a static memory heap.

All allocations are done from the static block of memory that is provided. It is much faster than the standard management but has some limitations such as the inability to free individual pointer values. All memory in the block must be freed at once.

Note: rtxMemHeapConvertStatic is generally preferrable to rtxMemStaticHeapCreate, since in most cases you will already have a heap object that you can reuse.

Parameters

ppvMemHeap	Pointer-to-pointer to variable to receive created object.
pmem	Pointer to static memory block to use for allocations.
memsize	Sizeof the memory block in bytes.

Returns

Status of the creation operation: 0 = success or RTERR-NOMEM if no memory available.

5.16 Pattern matching functions

These functions handle pattern matching which is required to to process XML schema pattern constraints.

Functions

- EXTERNRT OSBOOL rtxMatchPattern (OSCTXT ∗pctxt, const OSUTF8CHAR ∗text, const OSUTF8CHAR ∗pattern)

  This function compares the given string to the given pattern.
- EXTERNRT void rtxFreeRegexpCache (OSCTXT ∗pctxt)

  This function frees the memory associated with the regular expression cache.

5.16.1 Detailed Description

These functions handle pattern matching which is required to to process XML schema pattern constraints.

5.16.2 Function Documentation

5.16.2.1 rtxFreeRegexpCache()

```
EXTERNRT 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.)

5.16.2.2 rtxMatchPattern()

```
EXTERNRT 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.

5.17 Print Functions

These functions print the output in a "name=value" format.

Functions

- EXTERNRT void rtxPrintBoolean (const char ∗name, OSBOOL value)

  Prints a boolean value to stdout.
- EXTERNRT void rtxPrintDate (const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a date value to stdout.
- EXTERNRT void rtxPrintTime (const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a time value to stdout.
- EXTERNRT void rtxPrintDateTime (const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a dateTime value to stdout.
- EXTERNRT void rtxPrintInteger (const char ∗name, OSINT32 value)

  Prints an integer value to stdout.
- EXTERNRT void rtxPrintInt64 (const char ∗name, OSINT64 value)

  Prints a 64-bit integer value to stdout.
- EXTERNRT void rtxPrintIpv4Addr (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in IPv4 address format to standard output.
- EXTERNRT void rtxPrintIpv6Addr (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in IPv6 address format to standard output.
- EXTERNRT void rtxPrintTBCDStr (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in TBCD format to standard output.
- EXTERNRT void rtxPrintText (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in ASCII text format to standard output.
- EXTERNRT void rtxPrintUnsigned (const char ∗name, OSUINT32 value)

  Prints an unsigned integer value to stdout.
- EXTERNRT void rtxPrintUInt64 (const char ∗name, OSUINT64 value)

  Prints an unsigned 64-bit integer value to stdout.
- EXTERNRT 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.
- EXTERNRT void rtxPrintHexStrPlain (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintHexStrNoAscii (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintHexBinary (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  Prints an octet string value in hex binary format to stdout.
- EXTERNRT void rtxPrintCharStr (const char ∗name, const char ∗cstring)

  Prints an ASCII character string value to stdout.
- EXTERNRT void rtxPrintUTF8CharStr (const char ∗name, const OSUTF8CHAR ∗cstring)

  Prints a UTF-8 encoded character string value to stdout.
- EXTERNRT void rtxPrintUnicodeCharStr (const char ∗name, const OSUNICHAR ∗str, int nchars)

  This function prints a Unicode string to standard output.
- EXTERNRT void rtxPrintReal (const char ∗name, OSREAL value)

  Prints a REAL (float, double, decimal) value to stdout.

- EXTERNRT void rtxPrintNull (const char ∗name)

  Prints a NULL value to stdout.
- EXTERNRT void rtxPrintNVP (const char ∗name, const OSUTF8NVP ∗value)

  Prints a name-value pair to stdout.
- EXTERNRT void rtxPrintArrayNVP (const char ∗name, OSSIZE subscript, const OSUTF8NVP ∗value)

  Prints a name-value pair to stdout.
- EXTERNRT int rtxPrintFile (const char ∗filename)

  This function prints the contents of a text file to stdout.
- EXTERNRT void rtxPrintIndent (OSVOIDARG)

  This function prints indentation spaces to stdout.
- EXTERNRT void rtxPrintIncrIndent (OSVOIDARG)

  This function increments the current indentation level.
- EXTERNRT void rtxPrintDecrIndent (OSVOIDARG)

  This function decrements the current indentation level.
- EXTERNRT void rtxPrintCloseBrace (OSVOIDARG)

  This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.
- EXTERNRT 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.
- EXTERNRT const char ∗ rtxGetArrayElemName (char ∗buffer, OSSIZE bufsize, const char ∗name, OSSIZE subscript)

  This function returns an array element name in the form of 'name[subscript]' in the given character array buffer.

5.17.1 Detailed Description

These functions print the output in a "name=value" format.

The value format is obtained by calling one of the ToString functions with the given value.

5.17.2 Function Documentation

5.17.2.1 rtxGetArrayElemName()

```
EXTERNRT const char* rtxGetArrayElemName (
             char * buffer,
             OSSIZE bufsize,
             const char * name,
             OSSIZE subscript )
```

This function returns an array element name in the form of 'name[subscript]' in the given character array buffer.

Parameters

buffer	Character buffer into which formed name is written
bufsize	Size of character buffer
name	Element name
subscript	Array subscript

Returns

Pointer to name buffer

5.17.2.2 rtxPrintArrayNVP()

```
EXTERNRT void rtxPrintArrayNVP (
             const char * name,
             OSSIZE subscript,
             const OSUTF8NVP * value )
```

Prints a name-value pair to stdout.

In this case, the name is formed using the name and subscript arguments in the form of "name[subscript]".

Parameters

name	The name of the variable to print.
subscript	Array subscript value.
value	A pointer to name-value pair structure to print.

5.17.2.3 rtxPrintBoolean()

```
EXTERNRT 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.

5.17.2.4 rtxPrintCharStr()

```
EXTERNRT 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.

5.17.2.5 rtxPrintDate()

```
EXTERNRT 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.

5.17.2.6 rtxPrintDateTime()

```
EXTERNRT 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.

5.17.2.7 rtxPrintFile()

```
EXTERNRT 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.

5.17.2.8 rtxPrintHexBinary()

```
EXTERNRT 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.

5.17.2.9 rtxPrintHexStr()

```
EXTERNRT 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.

5.17.2.10 rtxPrintHexStrNoAscii()

```
EXTERNRT void rtxPrintHexStrNoAscii (
             const char * name,
             OSSIZE numocts,
             const OSOCTET * data )
```

This function prints the value of a binary string in hex format to standard output.

In contrast to rtxPrintHexStr, it never contains an ASCII dump.

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.

5.17.2.11 rtxPrintHexStrPlain()

```
EXTERNRT void rtxPrintHexStrPlain (
             const char * name,
             OSSIZE numocts,
             const OSOCTET * data )
```

This function prints the value of a binary string in hex format to standard output.

In contrast to rtxPrintHexStr, it is always printed on a single line with a '0x' prefix.

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.

5.17.2.12 rtxPrintInt64()

```
EXTERNRT 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.

5.17.2.13 rtxPrintInteger()

```
EXTERNRT 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.

5.17.2.14 rtxPrintIpv4Addr()

```
EXTERNRT void rtxPrintIpv4Addr (
            const char * name,
            OSSIZE numocts,
            const OSOCTET * data )
```

This function prints the value of a binary string in IPv4 address format to standard output.

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.

5.17.2.15 rtxPrintIpv6Addr()

```
EXTERNRT void rtxPrintIpv6Addr (
            const char * name,
            OSSIZE numocts,
            const OSOCTET * data )
```

This function prints the value of a binary string in IPv6 address format to standard output.

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.

5.17.2.16 rtxPrintNull()

```
EXTERNRT void rtxPrintNull (
            const char * name )
```

Prints a NULL value to stdout.

Parameters

name	The name of the variable to print.

5.17.2.17 rtxPrintNVP()

```
EXTERNRT 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.

5.17.2.18 rtxPrintReal()

```
EXTERNRT 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.

5.17.2.19 rtxPrintTBCDStr()

```
EXTERNRT void rtxPrintTBCDStr (
             const char * name,
             OSSIZE numocts,
             const OSOCTET * data )
```

This function prints the value of a binary string in TBCD format to standard output.

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.

5.17.2.20 rtxPrintText()

```
EXTERNRT void rtxPrintText (
             const char * name,
             OSSIZE numocts,
             const OSOCTET * data )
```

This function prints the value of a binary string in ASCII text format to standard output.

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.

5.17.2.21 rtxPrintTime()

```
EXTERNRT 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.

5.17.2.22 rtxPrintUInt64()

```
EXTERNRT 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.

5.17.2.23 rtxPrintUnicodeCharStr()

```
EXTERNRT 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).

5.17.2.24 rtxPrintUnsigned()

```
EXTERNRT 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.

5.17.2.25 rtxPrintUTF8CharStr()

```
EXTERNRT 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.

5.18 Print-To-Stream Functions

These functions print typed data in a "name=value" format.

Functions

- EXTERNRT void rtxPrintToStreamBoolean (OSCTXT ∗pctxt, const char ∗name, OSBOOL value)

  Prints a boolean value to a print stream.
- EXTERNRT void rtxPrintToStreamDate (OSCTXT ∗pctxt, const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a date value to a print stream.
- EXTERNRT void rtxPrintToStreamTime (OSCTXT ∗pctxt, const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a time value to a print stream.
- EXTERNRT void rtxPrintToStreamDateTime (OSCTXT ∗pctxt, const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a dateTime value to a print stream.
- EXTERNRT void rtxPrintToStreamInteger (OSCTXT ∗pctxt, const char ∗name, OSINT32 value)

  Prints an integer value to a print stream.
- EXTERNRT void rtxPrintToStreamInt64 (OSCTXT ∗pctxt, const char ∗name, OSINT64 value)

  Prints a 64-bit integer value to a print stream.
- EXTERNRT void rtxPrintToStreamUnsigned (OSCTXT ∗pctxt, const char ∗name, OSUINT32 value)

  Prints an unsigned integer value to a print stream.
- EXTERNRT void rtxPrintToStreamUInt64 (OSCTXT ∗pctxt, const char ∗name, OSUINT64 value)

  Prints an unsigned 64-bit integer value to a print stream.
- EXTERNRT void rtxPrintToStreamHexStr (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintToStreamHexStrPlain (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintToStreamHexStrNoAscii (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintToStreamHexBinary (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  Prints an octet string value in hex binary format to a print stream.
- EXTERNRT void rtxPrintToStreamCharStr (OSCTXT ∗pctxt, const char ∗name, const char ∗cstring)

  Prints an ASCII character string value to a print stream.
- EXTERNRT void rtxPrintToStreamUTF8CharStr (OSCTXT ∗pctxt, const char ∗name, const OSUTF8CHAR ∗cstring)

  Prints a UTF-8 encoded character string value to a print stream.
- EXTERNRT void rtxPrintToStreamUnicodeCharStr (OSCTXT ∗pctxt, const char ∗name, const OSUNICHAR ∗str, int nchars)

  This function prints a Unicode string to standard output.
- EXTERNRT void rtxPrintToStreamReal (OSCTXT ∗pctxt, const char ∗name, OSREAL value)

  Prints a REAL (float, double, decimal) value to a print stream.
- EXTERNRT void rtxPrintToStreamNull (OSCTXT ∗pctxt, const char ∗name)

  Prints a NULL value to a print stream.

- EXTERNRT void rtxPrintToStreamNVP (OSCTXT ∗pctxt, const char ∗name, const OSUTF8NVP ∗value)

  Prints a name-value pair to a print stream.
- EXTERNRT int rtxPrintToStreamFile (OSCTXT ∗pctxt, const char ∗filename)

  This function prints the contents of a text file to a print stream.
- EXTERNRT void rtxPrintToStreamIndent (OSCTXT ∗pctxt)

  This function prints indentation spaces to a print stream.
- EXTERNRT void rtxPrintToStreamResetIndent (OSCTXT ∗pctxt)

  This function resets the current indentation level to zero.
- EXTERNRT void rtxPrintToStreamIncrIndent (OSCTXT ∗pctxt)

  This function increments the current indentation level.
- EXTERNRT void rtxPrintToStreamDecrIndent (OSCTXT ∗pctxt)

  This function decrements the current indentation level.
- EXTERNRT void rtxPrintToStreamCloseBrace (OSCTXT ∗pctxt)

  This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.
- EXTERNRT 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.
- EXTERNRT void rtxHexDumpToStream (OSCTXT ∗pctxt, const OSOCTET ∗data, OSSIZE numocts)

  This function outputs a hexadecimal dump of the current buffer contents to a print stream.
- EXTERNRT void rtxHexDumpToStreamEx (OSCTXT ∗pctxt, const OSOCTET ∗data, OSSIZE numocts, OSSIZE 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.
- EXTERNRT void rtxHexDumpToStreamExNoAscii (OSCTXT ∗pctxt, const OSOCTET ∗data, OSSIZE numocts, OSSIZE bytesPerUnit)

  This function outputs a formatted hexadecimal dump of the current buffer to a print stream.

5.18.1 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.

5.18.2 Function Documentation

5.18.2.1 rtxHexDumpToStream()

```
EXTERNRT void rtxHexDumpToStream (
             OSCTXT ∗ pctxt,
             const OSOCTET ∗ data,
             OSSIZE 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

5.18.2.2 rtxHexDumpToStreamEx()

```
EXTERNRT void rtxHexDumpToStreamEx (
             OSCTXT * pctxt,
             const OSOCTET * data,
             OSSIZE numocts,
             OSSIZE 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).

5.18.2.3 rtxHexDumpToStreamExNoAscii()

```
EXTERNRT void rtxHexDumpToStreamExNoAscii (
             OSCTXT * pctxt,
             const OSOCTET * data,
             OSSIZE numocts,
             OSSIZE bytesPerUnit )
```

This function outputs a formatted hexadecimal dump of the current buffer to a print stream.

It outputs the dump as an array of bytes, words, or double words. It does not output any ASCII equivalent.

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).

5.18.2.4 rtxPrintToStreamBoolean()

```
EXTERNRT 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.

5.18.2.5 rtxPrintToStreamCharStr()

```
EXTERNRT 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.

5.18.2.6 rtxPrintToStreamDate()

```
EXTERNRT 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.

5.18.2.7 rtxPrintToStreamDateTime()

```
EXTERNRT 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.

5.18.2.8 rtxPrintToStreamDecrIndent()

```
EXTERNRT 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.

5.18.2.9 rtxPrintToStreamFile()

```
EXTERNRT 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.

5.18.2.10 rtxPrintToStreamHexBinary()

```
EXTERNRT void rtxPrintToStreamHexBinary (
            OSCTXT * pctxt,
            const char * name,
            OSSIZE 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.

5.18.2.11 rtxPrintToStreamHexStr()

```
EXTERNRT void rtxPrintToStreamHexStr (
            OSCTXT * pctxt,
            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

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.

5.18.2.12 rtxPrintToStreamHexStrNoAscii()

```
EXTERNRT void rtxPrintToStreamHexStrNoAscii (
            OSCTXT * pctxt,
            const char * name,
            OSSIZE numocts,
            const OSOCTET * data )
```

This function prints the value of a binary string in hex format to standard output.

In contrast to rtxPrintToStreamHexStr, it contains no ASCII output, but instead is a formatted block of hex text printed on multiple lines if needed.

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.

5.18.2.13 rtxPrintToStreamHexStrPlain()

```
EXTERNRT void rtxPrintToStreamHexStrPlain (
            OSCTXT * pctxt,
            const char * name,
            OSSIZE numocts,
            const OSOCTET * data )
```

This function prints the value of a binary string in hex format to standard output.

In contrast to rtxPrintToStreamHexStr, it is always printed on a single line with a '0x' prefix.

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.

5.18.2.14 rtxPrintToStreamIncrIndent()

```
EXTERNRT 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.

5.18.2.15 rtxPrintToStreamInt64()

```
EXTERNRT 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.

5.18.2.16 rtxPrintToStreamInteger()

```
EXTERNRT 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.

5.18.2.17 rtxPrintToStreamNull()

```
EXTERNRT 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.

5.18.2.18 rtxPrintToStreamNVP()

```
EXTERNRT 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.

5.18.2.19 rtxPrintToStreamReal()

```
EXTERNRT 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.

5.18.2.20 rtxPrintToStreamResetIndent()

```
EXTERNRT void rtxPrintToStreamResetIndent (
            OSCTXT * pctxt )
```

This function resets the current indentation level to zero.

Parameters

pctxt	A pointer to a context data structure that holds the print stream.

5.18.2.21 rtxPrintToStreamTime()

```
EXTERNRT 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.

5.18.2.22 rtxPrintToStreamUInt64()

```
EXTERNRT 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.

5.18.2.23 rtxPrintToStreamUnicodeCharStr()

```
EXTERNRT 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).

5.18.2.24 rtxPrintToStreamUnsigned()

```
EXTERNRT 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.

5.18.2.25 rtxPrintToStreamUTF8CharStr()

```
EXTERNRT 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.

5.19 Floating-point number utility functions

Floating-point utility function provide run-time functions for handling floating-point number types defined within a schema.

Functions

- EXTERNRT OSREAL rtxGetMinusInfinity (OSVOIDARG)

  Returns the IEEE negative infinity value.
- EXTERNRT OSREAL rtxGetMinusZero (OSVOIDARG)

  Returns the IEEE minus zero value.
- EXTERNRT OSREAL rtxGetNaN (OSVOIDARG)

  Returns the IEEE Not-A-Number (NaN) value.
- EXTERNRT OSREAL rtxGetPlusInfinity (OSVOIDARG)

  Returns the IEEE posative infinity value.
- EXTERNRT OSBOOL rtxIsMinusInfinity (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for negative infinity.
- EXTERNRT OSBOOL rtxIsMinusZero (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for minus zero.
- EXTERNRT OSBOOL rtxIsNaN (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for Not-A-Number (NaN).
- EXTERNRT OSBOOL rtxIsPlusInfinity (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for positive infinity.
- EXTERNRT 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.
- EXTERNRT 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.

5.19.1 Detailed Description

Floating-point utility function provide run-time functions for handling floating-point number types defined within a schema.

5.19.2 Function Documentation

5.19.2.1 rtxGetMinusInfinity()

```
EXTERNRT OSREAL rtxGetMinusInfinity (
            OSVOIDARG  )
```

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.

5.19.2.2 rtxGetMinusZero()

```
EXTERNRT OSREAL rtxGetMinusZero (
             OSVOIDARG  )
```

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.

5.19.2.3 rtxGetNaN()

```
EXTERNRT OSREAL rtxGetNaN (
             OSVOIDARG  )
```

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.

5.19.2.4 rtxGetPlusInfinity()

```
EXTERNRT OSREAL rtxGetPlusInfinity (
             OSVOIDARG  )
```

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.

5.19.2.5 rtxIsApproximate()

```
EXTERNRT 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.

5.19.2.6 rtxIsApproximateAbs()

```
EXTERNRT 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

5.19.2.7 rtxIsMinusInfinity()

```
EXTERNRT 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.

5.19.2.8 rtxIsMinusZero()

```
EXTERNRT 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.

5.19.2.9 rtxIsNaN()

```
EXTERNRT 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.

5.19.2.10 rtxIsPlusInfinity()

```
EXTERNRT 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.

5.20 Scalar Doubly-Linked List Utility Functions

The doubly-linked list utility functions provide common routines for managing linked lists.

Classes

- struct OSRTScalarDListNode

  This structure is used to hold a single data item within the list.
- struct OSRTScalarDList

  This is the main list structure.

Functions

- EXTERNRT void rtxScalarDListInit (OSRTScalarDList ∗pList)

  This function initializes a doubly linked list structure.
- EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListAppendDouble (struct OSCTXT ∗pctxt, OSRTScalarDList ∗pList, OSDOUBLE value)

  This set of functions appends an item of the given scalar type to the linked list structure.
- EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListAppendNode (OSRTScalarDList ∗pList, OSRTScalarDList←Node ∗pListNode)

  This function is used to append a node to the linked list.
- EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListInsertNode (OSRTScalarDList ∗pList, OSUINT32 idx, OS←RTScalarDListNode ∗pListNode)

  This function is used to insert a node into the linked list.
- EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListFindByIndex (const OSRTScalarDList ∗pList, OSUINT32 idx)

  This function will return the node pointer of the indexed entry in the list.
- EXTERNRT void rtxScalarDListFreeNode (struct OSCTXT ∗pctxt, OSRTScalarDList ∗pList, OSRTScalarDList←Node ∗node)

  This function will remove the given node from the list and free memory.
- EXTERNRT void rtxScalarDListRemove (OSRTScalarDList ∗pList, OSRTScalarDListNode ∗node)

  This function will remove the given node from the list.
- EXTERNRT void rtxScalarDListFreeNodes (struct OSCTXT ∗pctxt, OSRTScalarDList ∗pList)

  This function will free all of the dynamic memory used to hold the list node pointers.

5.20.1 Detailed Description

The doubly-linked list utility functions provide common routines for managing linked lists.

This module is identical to the rtxDList module except that the data varaibles that can be added to the lists are scalars (integer, double, float, etc.) whereas the standard rtxDList type hold pointers to more complex data items.

5.20.2 Function Documentation

5.20.2.1 rtxScalarDListAppendDouble()

```
EXTERNRT OSRTScalarDListNode* rtxScalarDListAppendDouble (
            struct OSCTXT * pctxt,
            OSRTScalarDList * pList,
            OSDOUBLE value )
```

This set of functions appends an item of the given scalar type to the linked list structure.

Separate functions exist for all of the different supported scalar types.

Parameters

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

5.20.2.2 rtxScalarDListAppendNode()

```
EXTERNRT OSRTScalarDListNode* rtxScalarDListAppendNode (
            OSRTScalarDList * pList,
            OSRTScalarDListNode * pListNode )
```

This function is used to append a node to the linked list.

This can be used instead of a scalar value append function. It requires the user to allocate and populate the list node structure.

Parameters

pList	A pointer to a linked list structure onto which the list node will be appended.
pListNode	List node structure to be appended to the list. If this memory is to be released with the standard list memory free function, then it must be allocated using the rtxMemAlloc function.

Returns

A pointer to an allocated node structure used to link the given data value into the list. This is the node structure that was passed in.

5.20.2.3 rtxScalarDListFindByIndex()

```
EXTERNRT OSRTScalarDListNode* rtxScalarDListFindByIndex (
             const OSRTScalarDList * pList,
             OSUINT32 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 ident field must be examined to determine what type of value is stored in the union structure.

5.20.2.4 rtxScalarDListFreeNode()

```
EXTERNRT void rtxScalarDListFreeNode (
             struct OSCTXT * pctxt,
             OSRTScalarDList * pList,
             OSRTScalarDListNode * node )
```

This function will remove the given node from the list and free memory.

It is assumed that memory for the list node structure was allocated using the `rtxMemAlloc` function.

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.

5.20.2.5 rtxScalarDListFreeNodes()

```
EXTERNRT void rtxScalarDListFreeNodes (
             struct OSCTXT * pctxt,
             OSRTScalarDList * pList )
```

This function will free all of the dynamic memory used to hold the list node pointers.

Parameters

pctxt	A pointer to a context structure.
pList	A pointer to a linked list structure.

5.20.2.6 rtxScalarDListInit()

```
EXTERNRT void rtxScalarDListInit (
            OSRTScalarDList * 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 scalar list structure is described by the `OSRTScalarDList` type. Nodes of the list are of type `OSRTScalarDListNode`.

Parameters

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

5.20.2.7 rtxScalarDListInsertNode()

```
EXTERNRT OSRTScalarDListNode* rtxScalarDListInsertNode (
            OSRTScalarDList * pList,
            OSUINT32 idx,
            OSRTScalarDListNode * pListNode )
```

This function is used to insert a node into the linked list.

Parameters

pList	A pointer to a linked list structure onto which the list node will be appended.
idx	Zero-based index into list where the specified node is to be inserted.
pListNode	List node structure to be appended to the list. If this memory is to be released with the standard list memory free function, then it must be allocated using the rtxMemAlloc function.

Returns

A pointer to an allocated node structure used to link the given data value into the list. This is the node structure that was passed in.

5.20.2.8 rtxScalarDListRemove()

```
EXTERNRT void rtxScalarDListRemove (
            OSRTScalarDList * pList,
            OSRTScalarDListNode * node )
```

This function will remove the given node from the list.

Parameters

pList	A pointer to a linked list structure.
node	Pointer to the list node to be removed.

5.21 TCP/IP or UDP socket utility functions

Typedefs

- typedef unsigned long OSIPADDR

  The IP address represented as unsigned long value.

Functions

- EXTERNRT int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET ∗pNewSocket, OSIPADDR ∗destAddr, int ∗destPort)

  This function permits an incoming connection attempt on a socket.
- EXTERNRT int rtxSocketAddrToStr (OSIPADDR ipAddr, char ∗pbuf, size_t bufsize)

  This function converts an IP address to its string representation.
- EXTERNRT int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)

  This function associates a local address with a socket.
- EXTERNRT int rtxSocketClose (OSRTSOCKET socket)

  This function closes an existing socket.
- EXTERNRT int rtxSocketConnect (OSRTSOCKET socket, const char ∗host, int port)

  This function establishes a connection to a specified socket.
- EXTERNRT int rtxSocketConnectTimed (OSRTSOCKET socket, const char ∗host, int port, int nsecs)

  This function establishes a connection to a specified socket.
- EXTERNRT int rtxSocketCreate (OSRTSOCKET ∗psocket)

  This function creates a TCP socket.
- EXTERNRT int rtxSocketGetHost (const char ∗host, struct in_addr ∗inaddr)

  This function resolves the given host name to an IP address.
- EXTERNRT int rtxSocketsInit (OSVOIDARG)

  This function initiates use of sockets by an application.
- EXTERNRT int rtxSocketListen (OSRTSOCKET socket, int maxConnection)

  This function places a socket a state where it is listening for an incoming connection.
- EXTERNRT 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.
- EXTERNRT int rtxSocketRecv (OSRTSOCKET socket, OSOCTET ∗pbuf, size_t bufsize)

  This function receives data from a connected socket.
- EXTERNRT int rtxSocketRecvTimed (OSRTSOCKET socket, OSOCTET ∗pbuf, size_t bufsize, OSUINT32 secs)

  This function receives data from a connected socket on a timed basis.
- EXTERNRT 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.
- EXTERNRT int rtxSocketSend (OSRTSOCKET socket, const OSOCTET ∗pdata, size_t size)

  This function sends data on a connected socket.
- EXTERNRT int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)

  This function turns blocking mode for a socket on or off.
- EXTERNRT int rtxSocketStrToAddr (const char ∗pIPAddrStr, OSIPADDR ∗pIPAddr)

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

5.21.1 Detailed Description

5.21.2 Typedef Documentation

5.21.2.1 OSIPADDR

```
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.

Definition at line 80 of file rtxSocket.h.

5.21.3 Function Documentation

5.21.3.1 rtxSocketAccept()

```
EXTERNRT 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.

5.21.3.2 rtxSocketAddrToStr()

```
EXTERNRT 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.

5.21.3.3 rtxSocketBind()

```
EXTERNRT 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.

5.21.3.4 rtxSocketClose()

```
EXTERNRT 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.

5.21.3.5 rtxSocketConnect()

```
EXTERNRT 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.

5.21.3.6 rtxSocketConnectTimed()

```
EXTERNRT 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.

5.21.3.7 rtxSocketCreate()

```
EXTERNRT 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.

5.21.3.8 rtxSocketGetHost()

```
EXTERNRT 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.

5.21.3.9 rtxSocketListen()

```
EXTERNRT 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.

5.21.3.10 rtxSocketParseURL()

```
EXTERNRT 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.

5.21.3.11 rtxSocketRecv()

```
EXTERNRT 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. RTERR_↩ WOULDBLOCK is returned if the socket is non-blocking and 0 bytes were available without blocking.

5.21.3.12 rtxSocketRecvTimed()

```
EXTERNRT 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.

5.21.3.13 rtxSocketSelect()

```
EXTERNRT 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.

5.21.3.14 rtxSocketSend()

```
EXTERNRT 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.

5.21.3.15 rtxSocketSetBlocking()

```
EXTERNRT 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.

5.21.3.16 rtxSocketsInit()

```
EXTERNRT int rtxSocketsInit (
             OSVOIDARG  )
```

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.

5.21.3.17 rtxSocketStrToAddr()

```
EXTERNRT 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.

5.22 Input/Output Data Stream Utility Functions

Stream functions are used for unbuffered stream operations.

Classes

- struct OSRTSTREAM

  The stream control block.

Macros

- #define rtxStreamBlockingRead rtxStreamRead

  rtxStreamBlockingRead is deprecated.

Typedefs

- typedef long(∗ OSRTStreamReadProc) (struct OSRTSTREAM ∗pStream, OSOCTET ∗pbuffer, size_t nocts)

  Stream read function pointer type.
- typedef OSRTStreamReadProc OSRTStreamBlockingReadProc

  OSRTStreamBlockingReadProc is deprecated.
- typedef long(∗ OSRTStreamWriteProc) (struct OSRTSTREAM ∗pStream, const OSOCTET ∗data, size_t numocts)

  Stream write function pointer type.
- typedef int(∗ OSRTStreamFlushProc) (struct OSRTSTREAM ∗pStream)

  Stream flush function pointer type.
- typedef int(∗ OSRTStreamCloseProc) (struct OSRTSTREAM ∗pStream)

  Stream close function pointer type.
- typedef int(∗ OSRTStreamSkipProc) (struct OSRTSTREAM ∗pStream, size_t skipBytes)

  Stream skip function pointer type.
- typedef int(∗ OSRTStreamMarkProc) (struct OSRTSTREAM ∗pStream, size_t readAheadLimit)

  Stream mark function pointer type.
- typedef int(∗ OSRTStreamResetProc) (struct OSRTSTREAM ∗pStream)

  Stream reset function pointer type.
- typedef int(∗ OSRTStreamGetPosProc) (struct OSRTSTREAM ∗pStream, size_t ∗ppos)

  Stream get position function pointer type.
- typedef int(∗ OSRTStreamSetPosProc) (struct OSRTSTREAM ∗pStream, size_t pos)

  Stream set position function pointer type.
- typedef struct OSRTSTREAM OSRTSTREAM

  The stream control block.

Functions

- EXTERNRT int rtxStreamClose (OSCTXT ∗pctxt)

  This function closes the input or output stream and releases any system resources associated with the stream.
- EXTERNRT int rtxStreamFlush (OSCTXT ∗pctxt)

  This function flushes the output stream and forces any buffered output octets to be written out.
- EXTERNRT int rtxStreamLoadInputBuffer (OSCTXT ∗pctxt, OSSIZE nbytes)

  This is for meant for internal use by the runtime.
- EXTERNRT int rtxStreamInit (OSCTXT ∗pctxt)

  This function initializes a stream part of the context block.
- EXTERNRT int rtxStreamInitCtxtBuf (OSCTXT ∗pctxt)

  This function initializes a stream to use the context memory buffer for stream buffering.
- EXTERNRT int rtxStreamRemoveCtxtBuf (OSCTXT ∗pctxt)

  This function removes the use of a context memory buffer from a stream.
- EXTERNRT long rtxStreamRead (OSCTXT ∗pctxt, OSOCTET ∗pbuffer, size_t nocts)

  Read up to nocts bytes of data from the input stream into an array of octets.
- EXTERNRT long rtxStreamRead2 (OSCTXT ∗pctxt, OSOCTET ∗pbuffer, size_t nocts, size_t bufSize)

  Attempt to read nocts bytes from the input stream into an array of octets.
- EXTERNRT int rtxStreamSkip (OSCTXT ∗pctxt, size_t skipBytes)

  This function skips over and discards the specified amount of data octets from this input stream.
- EXTERNRT 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.
- EXTERNRT int rtxStreamGetIOBytes (OSCTXT ∗pctxt, size_t ∗pPos)

  This function returns the number of processed octets.
- EXTERNRT int rtxStreamMark (OSCTXT ∗pctxt, size_t readAheadLimit)

  Marks the current position in this input stream.
- EXTERNRT int rtxStreamReset (OSCTXT ∗pctxt)

  Repositions this stream to the position recorded by the last call to the rtxStreamMark function.
- EXTERNRT OSBOOL rtxStreamMarkSupported (OSCTXT ∗pctxt)

  Tests if this input stream supports the mark and reset methods.
- EXTERNRT OSBOOL rtxStreamIsOpened (OSCTXT ∗pctxt)

  Tests if this stream opened (for reading or writing).
- EXTERNRT OSBOOL rtxStreamIsReadable (OSCTXT ∗pctxt)

  Tests if this stream opened for reading.
- EXTERNRT OSBOOL rtxStreamIsWritable (OSCTXT ∗pctxt)

  Tests if this stream opened for writing.
- EXTERNRT int rtxStreamRelease (OSCTXT ∗pctxt)

  This function releases the stream's resources.
- EXTERNRT void rtxStreamSetCapture (OSCTXT ∗pctxt, OSRTMEMBUF ∗pmembuf)

  This function sets a capture buffer for the stream.
- EXTERNRT OSRTMEMBUF ∗ rtxStreamGetCapture (OSCTXT ∗pctxt)

  This function returns the capture buffer currently assigned to the stream.
- EXTERNRT int rtxStreamGetPos (OSCTXT ∗pctxt, size_t ∗ppos)

  Get the current position in input stream.
- EXTERNRT int rtxStreamSetPos (OSCTXT ∗pctxt, size_t pos)

  Set the current position in input stream.

5.22.1 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`

5.22.2 Macro Definition Documentation

5.22.2.1 rtxStreamBlockingRead

```
#define rtxStreamBlockingRead rtxStreamRead
```

rtxStreamBlockingRead is deprecated.

Use rtxStreamRead.

Definition at line 344 of file rtxStream.h.

5.22.3 Typedef Documentation

5.22.3.1 OSRTSTREAM

```
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.

5.22.3.2 OSRTStreamBlockingReadProc

```
typedef OSRTStreamReadProc OSRTStreamBlockingReadProc
```

OSRTStreamBlockingReadProc is deprecated.

Use OSRTStreamReadProc.

Definition at line 83 of file rtxStream.h.

5.22.3.3 OSRTStreamCloseProc

```
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.

Definition at line 105 of file rtxStream.h.

5.22.3.4 OSRTStreamFlushProc

```
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.

Definition at line 98 of file rtxStream.h.

5.22.3.5 OSRTStreamGetPosProc

```
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.

Definition at line 136 of file rtxStream.h.

5.22.3.6 OSRTStreamMarkProc

```
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.

Definition at line 121 of file rtxStream.h.

5.22.3.7 OSRTStreamReadProc

```
typedef long(* OSRTStreamReadProc) (struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t nocts)
```

Stream read function pointer type.

A user may implement a customized read function for specific input streams. The read function is defined in the OSR←↩TSTREAM control structure.

The read function may read fewer bytes than requested, but it will read at least one byte (assuming nocts $>$ 0) unless EOF is reached or there is an error. Thus, if nocts $>$ 0, this will return 0 only if EOF is reached. Reaching EOF shall not be considered an error.

Parameters

pStream	Identifies the stream control structure.
pbuffer	is the buffer into which bytes shall be read.
nocts	is the number of bytes to be read; the buffer must be at least this large.

Returns

The number of bytes read (possibly 0), or a negative error code.

Definition at line 78 of file rtxStream.h.

5.22.3.8 OSRTStreamResetProc

```
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.

Definition at line 128 of file rtxStream.h.

5.22.3.9 OSRTStreamSetPosProc

```
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.

Definition at line 144 of file rtxStream.h.

5.22.3.10 OSRTStreamSkipProc

```
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.

Definition at line 113 of file rtxStream.h.

5.22.3.11 OSRTStreamWriteProc

```
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.

Definition at line 90 of file rtxStream.h.

5.22.4 Function Documentation

5.22.4.1 rtxStreamClose()

```
EXTERNRT 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`.

5.22.4.2 rtxStreamFlush()

```
EXTERNRT 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.

5.22.4.3 rtxStreamGetCapture()

```
EXTERNRT 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.

5.22.4.4 rtxStreamGetIOBytes()

```
EXTERNRT 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).

5.22.4.5 rtxStreamGetPos()

```
EXTERNRT 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.

5.22.4.6 rtxStreamInit()

```
EXTERNRT 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.

5.22.4.7 rtxStreamInitCtxtBuf()

```
EXTERNRT int rtxStreamInitCtxtBuf (
             OSCTXT * pctxt )
```

This function initializes a stream to use the context memory buffer for stream buffering.

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.

5.22.4.8 rtxStreamIsOpened()

```
EXTERNRT 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.

5.22.4.9 rtxStreamIsReadable()

```
EXTERNRT 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.

5.22.4.10 rtxStreamIsWritable()

```
EXTERNRT 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.

5.22.4.11 rtxStreamLoadInputBuffer()

```
EXTERNRT int rtxStreamLoadInputBuffer (
             OSCTXT * pctxt,
             OSSIZE nbytes )
```

This is for meant for internal use by the runtime.

Read at least as many bytes, from the context's input stream into the context buffer, as necessary to make nbytes of data available in the context buffer.

Upon return, pctxt.buffer.byteIndex + nbytes <= pctxt.buffer.size OR EOF has been reached OR an error has been logged and is being returned.

If the context buffer has not been created, this will create it. If the context buffer needs to be made larger, this will enlarge it or else log, and return, an error.

Any bytes read from the stream will be sent to the capture buffer, if there is one.

Parameters

pctxt	A context with an attached stream using the context's buffer as a buffer for the stream.

Returns

0 or or negative error. EOF is not considered an error.

5.22.4.12 rtxStreamMark()

```
EXTERNRT 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.

5.22.4.13 rtxStreamMarkSupported()

```
EXTERNRT 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.

5.22.4.14 rtxStreamRead()

```
EXTERNRT long rtxStreamRead (
             OSCTXT * pctxt,
             OSOCTET * pbuffer,
             size_t nocts )
```

Read up to nocts bytes of data from the input stream into an array of octets.

This function blocks until nocts of data are read, end of file is detected, or an error occurs. EOF is not treated as an error.

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.
nocts	Number of bytes to read.

Returns

The total number of octets read into pbuffer, or negative value with error code if an error occurs.

5.22.4.15 rtxStreamRead2()

```
EXTERNRT long rtxStreamRead2 (
             OSCTXT * pctxt,
             OSOCTET * pbuffer,
             size_t nocts,
             size_t bufSize )
```

Attempt to read nocts bytes from the input stream into an array of octets.

It may read more bytes (as many as bufSize). It may read fewer bytes if end of file is detected or an error occurs. EOF is not treated as an error.

This function blocks until nocts of data are read, end of file is detected, or an error occurs. EOF is not treated as an error.

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.
nocts	Number of octets to read; nocts >= 1
bufSize	Size of the buffer.

Returns

The total number of octets read into pbuffer, or negative value with error code if an error occurs.

5.22.4.16 rtxStreamRelease()

```
EXTERNRT 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.

5.22.4.17 rtxStreamRemoveCtxtBuf()

```
EXTERNRT int rtxStreamRemoveCtxtBuf (
            OSCTXT * pctxt )
```

This function removes the use of a context memory buffer from a stream.

Parameters

pctxt	Pointer to context structure variable which is assumed to contain an initialized stream with context buffering enabled.

Returns

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

5.22.4.18 rtxStreamReset()

```
EXTERNRT 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.

5.22.4.19 rtxStreamSetCapture()

```
EXTERNRT 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.

5.22.4.20 rtxStreamSetPos()

```
EXTERNRT 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.

5.22.4.21 rtxStreamSkip()

```
EXTERNRT int rtxStreamSkip (
             OSCTXT * pctxt,
             size_t skipBytes )
```

This function skips over and discards the specified amount of data octets from this input stream.

An attempt to skip past the end of the input is an error. (This skips bytes of input, which may be held in a buffer, so it does not necessarily skip bytes in the underlying stream. Also, pctxt->buffer.bitOffset is irrelevant to this function.)

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.

5.22.4.22 rtxStreamWrite()

```
EXTERNRT 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.

5.23 File stream functions.

File stream functions are used for stream operations with files.

Functions

- EXTERNRT int rtxStreamFileAttach (OSCTXT ∗pctxt, FILE ∗pFile, OSUINT16 flags)

  Attaches the existing file structure pointer to the stream.
- EXTERNRT int rtxStreamFileOpen (OSCTXT ∗pctxt, const char ∗pFilename, OSUINT16 flags)

  Opens a file stream.
- EXTERNRT int rtxStreamFileCreateReader (OSCTXT ∗pctxt, const char ∗pFilename)

  This function creates an input file stream using the specified file name.
- EXTERNRT int rtxStreamFileCreateWriter (OSCTXT ∗pctxt, const char ∗pFilename)

  This function creates an output file stream using the file name.

5.23.1 Detailed Description

File stream functions are used for stream operations with files.

5.23.2 Function Documentation

5.23.2.1 rtxStreamFileAttach()

```
EXTERNRT 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.

5.23.2.2 rtxStreamFileCreateReader()

```
EXTERNRT 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.

5.23.2.3 rtxStreamFileCreateWriter()

```
EXTERNRT 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.

5.23.2.4 rtxStreamFileOpen()

```
EXTERNRT 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.

5.24 Memory stream functions.

Memory stream functions are used for memory stream operations.

Functions

- EXTERNRT int rtxStreamMemoryCreate (OSCTXT ∗pctxt, OSUINT16 flags)

  Opens a memory stream.
- EXTERNRT int rtxStreamMemoryAttach (OSCTXT ∗pctxt, OSOCTET ∗pMemBuf, size_t bufSize, OSUINT16 flags)

  Opens a memory stream using the specified memory buffer.
- EXTERNRT OSOCTET ∗ rtxStreamMemoryGetBuffer (OSCTXT ∗pctxt, size_t ∗pSize)

  This function returns the memory buffer and its size for the given memory stream.
- EXTERNRT int rtxStreamMemoryCreateReader (OSCTXT ∗pctxt, OSOCTET ∗pMemBuf, size_t bufSize)

  This function creates an input memory stream using the specified buffer.
- EXTERNRT int rtxStreamMemoryCreateWriter (OSCTXT ∗pctxt, OSOCTET ∗pMemBuf, size_t bufSize)

  This function creates an output memory stream using the specified buffer.
- EXTERNRT int rtxStreamMemoryResetWriter (OSCTXT ∗pctxt)

  This function resets the output memory stream internal buffer to allow it to be overwritten with new data.

5.24.1 Detailed Description

Memory stream functions are used for memory stream operations.

5.24.2 Function Documentation

5.24.2.1 rtxStreamMemoryAttach()

```
EXTERNRT 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.

5.24.2.2 rtxStreamMemoryCreate()

```
EXTERNRT 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.

5.24.2.3 rtxStreamMemoryCreateReader()

```
EXTERNRT 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.

5.24.2.4 rtxStreamMemoryCreateWriter()

```
EXTERNRT 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.

5.24.2.5 rtxStreamMemoryGetBuffer()

```
EXTERNRT OSOCTET* rtxStreamMemoryGetBuffer (
             OSCTXT * pctxt,
             size_t * pSize )
```

This function returns the memory buffer and its size for the given memory stream.

The caller of this function is responsible for freeing the memory.

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.

5.24.2.6 rtxStreamMemoryResetWriter()

```
EXTERNRT 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.

5.25 Socket stream functions.

Socket stream functions are used for socket stream operations.

Functions

- EXTERNRT int rtxStreamSocketAttach (OSCTXT ∗pctxt, OSRTSOCKET socket, OSUINT16 flags)

  Attaches the existing socket handle to the stream.
- EXTERNRT int rtxStreamSocketClose (OSCTXT ∗pctxt)

  This function closes a socket stream.
- EXTERNRT int rtxStreamSocketCreateWriter (OSCTXT ∗pctxt, const char ∗host, int port)

  This function opens a socket stream for writing.
- EXTERNRT int rtxStreamSocketSetOwnership (OSCTXT ∗pctxt, OSBOOL ownSocket)

  This function transfers ownership of the socket to or from the stream instance.
- EXTERNRT int rtxStreamSocketSetReadTimeout (OSCTXT ∗pctxt, OSUINT32 nsecs)

  This function sets the read timeout value to the given number of seconds.

5.25.1 Detailed Description

Socket stream functions are used for socket stream operations.

5.25.2 Function Documentation

5.25.2.1 rtxStreamSocketAttach()

```
EXTERNRT 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.

5.25.2.2 rtxStreamSocketClose()

```
EXTERNRT 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.

5.25.2.3 rtxStreamSocketCreateWriter()

```
EXTERNRT 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.

5.25.2.4 rtxStreamSocketSetOwnership()

```
EXTERNRT 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.

5.25.2.5 rtxStreamSocketSetReadTimeout()

```
EXTERNRT 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.

5.26 Telephony Binary Coded Decimal (TBCD) Helper Functions

Telephony Binary Coded Decimal (TBCD) helper functions provide assistance in converting TBCD strings to and from binary form as document in standard ITU-T Q.825.

Functions

- EXTERNRT int rtxQ825TBCDToString (OSSIZE numocts, const OSOCTET ∗data, char ∗buffer, OSSIZE bufsiz)

  This function converts a Q.825 TBCD value to a standard null-terminated string.
- EXTERNRT int rtxDecQ825TBCDString (OSCTXT ∗pctxt, OSSIZE numocts, char ∗buffer, OSSIZE bufsiz)

  This function decodes a Q.825 TBCD value to a standard null-terminated string.
- EXTERNRT int rtxEncQ825TBCDString (OSCTXT ∗pctxt, const char ∗str)

  This function encodes a null-terminated string Q.825 TBCD string.
- EXTERNRT int rtxTBCDBinToChar (OSUINT8 bcdDigit, char ∗pdigit)

  This function converts a TBCD binary character into its ASCII equivalent.
- EXTERNRT int rtxTBCDCharToBin (char digit, OSUINT8 ∗pbyte)

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

5.26.1 Detailed Description

Telephony Binary Coded Decimal (TBCD) helper functions provide assistance in converting TBCD strings to and from binary form as document in standard ITU-T Q.825.

5.26.2 Function Documentation

5.26.2.1 rtxDecQ825TBCDString()

```
EXTERNRT int rtxDecQ825TBCDString (
             OSCTXT ∗ pctxt,
             OSSIZE numocts,
             char ∗ buffer,
             OSSIZE 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

5.26.2.2 rtxEncQ825TBCDString()

```
EXTERNRT int rtxEncQ825TBCDString (
            OSCTXT * pctxt,
            const char * 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

5.26.2.3 rtxQ825TBCDToString()

```
EXTERNRT int rtxQ825TBCDToString (
            OSSIZE numocts,
            const OSOCTET * data,
            char * buffer,
            OSSIZE 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

5.26.2.4 rtxTBCDBinToChar()

```
EXTERNRT int rtxTBCDBinToChar (
             OSUINT8 bcdDigit,
             char * pdigit )
```

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

Parameters

tbcdDigit	TBCD digit
pdigit	Pointer to character to receive converted character

Returns

0 if conversion successful, or negative error code

Since

6.6

5.26.2.5 rtxTBCDCharToBin()

```
EXTERNRT int rtxTBCDCharToBin (
             char digit,
             OSUINT8 * 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

5.27 Text Reading and Writing Functions

These functions support working with a textual representation of values.

Functions

- EXTERNRT int rtxTxtMatchChar (OSCTXT ∗pctxt, OSUTF8CHAR ch, OSBOOL skipWs)

  This function matches the given character or logs and returns an error.
- EXTERNRT int rtxTxtMatchChars (OSCTXT ∗pctxt, const OSUTF8CHAR ∗chars, OSBOOL skipWs)

  This function matches the given characters or logs and returns an error.
- EXTERNRT int rtxTxtPeekChar (OSCTXT ∗pctxt, OSUTF8CHAR ∗pch, OSBOOL skipWs)

  This function determines the next character in the input.
- EXTERNRT char rtxTxtPeekChar2 (OSCTXT ∗pctxt, OSBOOL skipWs)

  This function determines the next character in the input.
- EXTERNRT int rtxTxtSkipWhitespace (OSCTXT ∗pctxt)

  This function skips any whitespace in the input.
- EXTERNRT int rtxTxtReadBigInt (OSCTXT ∗pctxt, char ∗∗ppvalue)

  This function reads an integer, using standard decimal notation, into a character string.
- EXTERNRT int rtxTxtReadInt8 (OSCTXT ∗pctxt, OSINT8 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 8-bit signed integer.
- EXTERNRT int rtxTxtReadInt16 (OSCTXT ∗pctxt, OSINT16 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 16-bit signed integer.
- EXTERNRT int rtxTxtReadInt32 (OSCTXT ∗pctxt, OSINT32 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 32-bit signed integer.
- EXTERNRT int rtxTxtReadInt64 (OSCTXT ∗pctxt, OSINT64 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 64-bit signed integer.
- EXTERNRT int rtxTxtReadUInt8 (OSCTXT ∗pctxt, OSUINT8 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 8-bit unsigned integer.
- EXTERNRT int rtxTxtReadUInt16 (OSCTXT ∗pctxt, OSUINT16 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 16-bit unsigned integer.
- EXTERNRT int rtxTxtReadUInt32 (OSCTXT ∗pctxt, OSUINT32 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 32-bit unsigned integer.
- EXTERNRT int rtxTxtReadUInt64 (OSCTXT ∗pctxt, OSUINT64 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 64-bit unsigned integer.
- EXTERNRT int rtxTxtWriteInt (OSCTXT ∗pctxt, OSINT32 value)

  This writes a 32-bit signed integer to the buffer using standard decimal (base 10) notation.
- EXTERNRT int rtxTxtWriteInt64 (OSCTXT ∗pctxt, OSINT64 value)

  This writes a 64-bit signed integer to the buffer using standard decimal (base 10) notation.
- EXTERNRT int rtxTxtWriteUInt (OSCTXT ∗pctxt, OSUINT32 value)

  This writes a 32-bit unsigned integer to the buffer using standard decimal (base 10) notation.
- EXTERNRT int rtxTxtWriteUInt64 (OSCTXT ∗pctxt, OSUINT64 value)

  This writes a 64-bit unsigned integer to the buffer using standard decimal (base 10) notation.

5.27.1 Detailed Description

These functions support working with a textual representation of values.

Values of various C/C++ types can be written to, or read from, the context buffer, with the buffer holding a text representation.

5.27.2 Function Documentation

5.27.2.1 rtxTxtMatchChar()

```
EXTERNRT int rtxTxtMatchChar (
             OSCTXT * pctxt,
             OSUTF8CHAR ch,
             OSBOOL skipWs )
```

This function matches the given character or logs and returns an error.

You may optionally choose to skip any leading whitespace.

Parameters

pctxt	Pointer to context block structure.
ch	The character to be matched.
skipWs	If TRUE, skip any leading whitespace.

Returns

Completion status of operation:

- 0 = success,
- RTERR_INVCHAR = different character found
- negative return value is error.

5.27.2.2 rtxTxtMatchChars()

```
EXTERNRT int rtxTxtMatchChars (
             OSCTXT * pctxt,
             const OSUTF8CHAR * chars,
             OSBOOL skipWs )
```

This function matches the given characters or logs and returns an error.

You may optionally choose to skip any leading whitespace.

Parameters

pctxt	Pointer to context block structure.
chars	The characters to be matched.
skipWs	If TRUE, skip any leading whitespace.

Returns

Completion status of operation:

- 0 = success,
- RTERR_INVCHAR = different character found
- negative return value is error.

5.27.2.3 rtxTxtPeekChar()

```
EXTERNRT int rtxTxtPeekChar (
             OSCTXT * pctxt,
             OSUTF8CHAR * pch,
             OSBOOL skipWs )
```

This function determines the next character in the input.

You may optionally skip any whitespace and peek at the first non-whitespace character. Skipping whitespace consumes it.

It is an error if EOF prevents a character from being returned.

Parameters

pctxt	Pointer to OSCTXT structure
pch	A pointer to a variable to receive the next character.

Returns

Completion status of operation:

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

5.27.2.4 rtxTxtPeekChar2()

```
EXTERNRT char rtxTxtPeekChar2 (
             OSCTXT * pctxt,
             OSBOOL skipWs )
```

This function determines the next character in the input.

You may optionally skip any whitespace and peek at the first non-whitespace character. Skipping whitespace consumes it.

It is an error if EOF prevents a character from being returned.

Parameters

pctxt	Pointer to OSCTXT structure

Returns

The peeked character, or null if there is a failure. The error will be logged in the context.

5.27.2.5 rtxTxtReadBigInt()

```
EXTERNRT int rtxTxtReadBigInt (
             OSCTXT * pctxt,
             char ** ppvalue )
```

This function reads an integer, using standard decimal notation, into a character string.

Leading whitespace is consumed.

This recognizes an optional minus sign, followed by 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
ppvalue	Pointer to string to receive newly allocated string with decoded result.

Returns

Completion status of operation:

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

5.27.2.6 rtxTxtReadInt16()

```
EXTERNRT int rtxTxtReadInt16 (
             OSCTXT * pctxt,
             OSINT16 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 16-bit signed integer.

Leading whitespace is consumed.

This recognizes an optional minus sign, followed by 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to 16-bit integer value to receive decoded result.

Returns

Completion status of operation:

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

5.27.2.7 rtxTxtReadInt32()

```
EXTERNRT int rtxTxtReadInt32 (
             OSCTXT * pctxt,
             OSINT32 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 32-bit signed integer.

Leading whitespace is consumed.

This recognizes an optional minus sign, followed by 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to 32-bit integer value to receive decoded result.

Returns

Completion status of operation:

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

5.27.2.8 rtxTxtReadInt64()

```
EXTERNRT int rtxTxtReadInt64 (
            OSCTXT * pctxt,
            OSINT64 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 64-bit signed integer.

Leading whitespace is consumed.

This recognizes an optional minus sign, followed by 1 or more digits, with no superfluous leading zeros.

Parameters

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

Returns

Completion status of operation:

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

5.27.2.9 rtxTxtReadInt8()

```
EXTERNRT int rtxTxtReadInt8 (
            OSCTXT * pctxt,
            OSINT8 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 8-bit signed integer.

Leading whitespace is consumed.

This recognizes an optional minus sign, followed by 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to 8-bit integer value to receive decoded result.

Returns

Completion status of operation:

- 0 = success,

- negative return value is error.

5.27.2.10 rtxTxtReadUInt16()

```
EXTERNRT int rtxTxtReadUInt16 (
             OSCTXT * pctxt,
             OSUINT16 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 16-bit unsigned integer.

Leading whitespace is consumed.

This recognizes 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to 16-bit unsigned integer value to receive decoded result.

Returns

Completion status of operation:

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

5.27.2.11 rtxTxtReadUInt32()

```
EXTERNRT int rtxTxtReadUInt32 (
             OSCTXT * pctxt,
             OSUINT32 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 32-bit unsigned integer.

Leading whitespace is consumed.

This recognizes 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to 32-bit unsigned integer value to receive decoded result.

Returns

Completion status of operation:

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

5.27.2.12 rtxTxtReadUInt64()

```
EXTERNRT int rtxTxtReadUInt64 (
             OSCTXT * pctxt,
             OSUINT64 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 64-bit unsigned integer.

Leading whitespace is consumed.

This recognizes 1 or more digits, with no superfluous leading zeros.

Parameters

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

Returns

Completion status of operation:

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

5.27.2.13 rtxTxtReadUInt8()

```
EXTERNRT int rtxTxtReadUInt8 (
             OSCTXT * pctxt,
             OSUINT8 * pvalue )
```

This function reads an integer, using standard decimal notation, into an 8-bit unsigned integer.

Leading whitespace is consumed.

This recognizes 1 or more digits, with no superfluous leading zeros.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to 8-bit unsigned integer value to receive decoded result.

Returns

Completion status of operation:

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

5.27.2.14 rtxTxtSkipWhitespace()

```
EXTERNRT int rtxTxtSkipWhitespace (
             OSCTXT * pctxt )
```

This function skips any whitespace in the input.

Parameters

pctxt	Pointer to OSCTXT structure

Returns

Completion status of operation:

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

5.27.2.15 rtxTxtWriteInt()

```
EXTERNRT int rtxTxtWriteInt (
             OSCTXT * pctxt,
             OSINT32 value )
```

This writes a 32-bit signed integer to the buffer using standard decimal (base 10) notation.

A sign is used only for negative values.

Parameters

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

Returns

Completion status of operation:

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

5.27.2.16 rtxTxtWriteInt64()

```
EXTERNRT int rtxTxtWriteInt64 (
            OSCTXT * pctxt,
            OSINT64 value )
```

This writes a 64-bit signed integer to the buffer using standard decimal (base 10) notation.

A sign is used only for negative values.

Parameters

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

Returns

Completion status of operation:

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

5.27.2.17 rtxTxtWriteUInt()

```
EXTERNRT int rtxTxtWriteUInt (
            OSCTXT * pctxt,
            OSUINT32 value )
```

This writes a 32-bit unsigned integer to the buffer using standard decimal (base 10) notation.

No sign is used.

Parameters

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

Returns

Completion status of operation:

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

5.27.2.18 rtxTxtWriteUInt64()

```
EXTERNRT int rtxTxtWriteUInt64 (
            OSCTXT * pctxt,
            OSUINT64 value )
```

This writes a 64-bit unsigned integer to the buffer using standard decimal (base 10) notation.

No sign is used.

Parameters

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

Returns

Completion status of operation:

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

5.28 UTF-8 String Functions

The UTF-8 string functions handle string operations on UTF-8 encoded strings.

Macros

- #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR∗)lstr)

  Compare UTF-8 string to a string literal.

Functions

- EXTERNRT 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).
- EXTERNRT long rtxUTF8ToUnicode32 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗inbuf, OS32BITCHAR ∗outbuf, size_t outbufsiz)

  This function converts a UTF-8 string to a Unicode string (UTF-32).
- EXTERNRT int rtxValidateUTF8 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗inbuf)

  This function will validate a UTF-8 encoded string to ensure that it is encoded correctly.
- EXTERNRT size_t rtxUTF8Len (const OSUTF8CHAR ∗inbuf)

  This function will return the length (in characters) of a null-terminated UTF-8 encoded string.
- EXTERNRT size_t rtxUTF8LenBytes (const OSUTF8CHAR ∗inbuf)

  This function will return the length (in bytes) of a null-terminated UTF-8 encoded string.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET ∗buf, size_t bufsiz)

  This function will convert a wide character into an encoded UTF-8 character byte string.
- EXTERNRT 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.
- EXTERNRT OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR ∗buf, OSUINT32 ∗len)

  Thia function will convert a UTF-8 encoded character value into a wide character.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8StrChr (OSUTF8CHAR ∗utf8str, OS32BITCHAR utf8char)

  This function finds a character in the given UTF-8 character string.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8Strdup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str)

  This function creates a duplicate copy of the given UTF-8 character string.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8Strndup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, size_t nbytes)

  This function creates a duplicate copy of the given UTF-8 character string.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8StrRefOrDup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str)

  This function check to see if the given UTF8 string pointer exists on the memory heap.
- EXTERNRT OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR ∗utf8str1, const OSUTF8CHAR ∗utf8str2)

  This function compares two UTF-8 string values for equality.
- EXTERNRT OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR ∗utf8str1, const OSUTF8CHAR ∗utf8str2, size_t count)

  This function compares two UTF-8 string values for equality.
- EXTERNRT 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).

- EXTERNRT 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).
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8Strcpy (OSUTF8CHAR ∗dest, size_t bufsiz, const OSUTF8CHAR ∗src)

  This function copies a null-terminated UTF-8 string to a target buffer.
- EXTERNRT 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.
- EXTERNRT OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR ∗str)

  This function computes a hash code for the given string value.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8StrToBool (const OSUTF8CHAR ∗utf8str, OSBOOL ∗pvalue)

  This function converts the given null-terminated UTF-8 string to a boolean (true/false) value.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8StrnToDouble (const OSUTF8CHAR ∗utf8str, size_t nbytes, OSREAL ∗pvalue)

  This function converts the given part of UTF-8 string to a double value.
- EXTERNRT int rtxUTF8StrToInt (const OSUTF8CHAR ∗utf8str, OSINT32 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to an integer value.
- EXTERNRT int rtxUTF8StrnToInt (const OSUTF8CHAR ∗utf8str, size_t nbytes, OSINT32 ∗pvalue)

  This function converts the given part of UTF-8 string to an integer value.
- EXTERNRT int rtxUTF8StrToUInt (const OSUTF8CHAR ∗utf8str, OSUINT32 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to an unsigned integer value.
- EXTERNRT 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.
- EXTERNRT 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).
- EXTERNRT 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).
- EXTERNRT int rtxUTF8StrToInt64 (const OSUTF8CHAR ∗utf8str, OSINT64 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to a 64-bit integer value.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8StrToUInt64 (const OSUTF8CHAR ∗utf8str, OSUINT64 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to an unsigned 64-bit integer value.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8ToDynUniStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, const OSUNICHAR ∗∗ppdata, OSUINT32 ∗pnchars)

  This function converts the given UTF-8 string to a Unicode string (UTF-16).
- EXTERNRT int rtxUTF8ToDynUniStr32 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, const OS32BITCHAR ∗∗ppdata, OSUINT32 ∗pnchars)

  This function converts the given UTF-8 string to a Unicode string (UTF-32).

- EXTERNRT int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR ∗utf8instr, size_t nbytes, const OSUTF8C↩ HAR ∗∗putf8outstr)

  This function removes leading and trailing whitespace from a string.
- EXTERNRT int rtxUTF8StrToDynHexStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, OSDynOctStr ∗pvalue)

  This function converts the given null-terminated UTF-8 string to a octet string value.
- EXTERNRT int rtxUTF8StrnToDynHexStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, size_t nbytes, OS↩ DynOctStr ∗pvalue)

  This function converts the given part of UTF-8 string to a octet string value.
- EXTERNRT 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.
- EXTERNRT const OSUTF8CHAR ∗ rtxUTF8StrNextTok (OSUTF8CHAR ∗utf8str, OSUTF8CHAR ∗∗ppNext)

  This function returns the next whitespace-separated token from the input string.

5.28.1 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.

5.28.2 Macro Definition Documentation

5.28.2.1 RTUTF8STRCMPL

```
#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")

Definition at line 567 of file rtxUTF8.h.

5.28.3 Function Documentation

5.28.3.1 rtxUTF8CharSize()

```
EXTERNRT 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.

5.28.3.2 rtxUTF8CharToWC()

```
EXTERNRT 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.

5.28.3.3 rtxUTF8DecodeChar()

```
EXTERNRT 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.

5.28.3.4 rtxUTF8EncodeChar()

```
EXTERNRT 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.

5.28.3.5 rtxUTF8Len()

```
EXTERNRT 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.

5.28.3.6 rtxUTF8LenBytes()

```
EXTERNRT 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.

5.28.3.7 rtxUTF8RemoveWhiteSpace()

```
EXTERNRT 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.

5.28.3.8 rtxUTF8StrChr()

```
EXTERNRT 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.

5.28.3.9 rtxUTF8Strcmp()

```
EXTERNRT 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.

5.28.3.10 rtxUTF8Strcpy()

```
EXTERNRT 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.

5.28.3.11 rtxUTF8Strdup()

```
EXTERNRT 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.

5.28.3.12 rtxUTF8StrEqual()

```
EXTERNRT 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.

5.28.3.13 rtxUTF8StrHash()

```
EXTERNRT 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.

5.28.3.14 rtxUTF8StrJoin()

```
EXTERNRT 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.

5.28.3.15 rtxUTF8Strncmp()

```
EXTERNRT 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.

5.28.3.16 rtxUTF8Strncpy()

```
EXTERNRT 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.

5.28.3.17 rtxUTF8Strndup()

```
EXTERNRT 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.

5.28.3.18 rtxUTF8StrnEqual()

```
EXTERNRT 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.

5.28.3.19 rtxUTF8StrNextTok()

```
EXTERNRT 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.

5.28.3.20 rtxUTF8StrnToBool()

```
EXTERNRT 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

5.28.3.21 rtxUTF8StrnToDouble()

```
EXTERNRT 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

5.28.3.22 rtxUTF8StrnToDynHexStr()

```
EXTERNRT 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.

5.28.3.23 rtxUTF8StrnToInt()

```
EXTERNRT 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

5.28.3.24 rtxUTF8StrnToInt64()

```
EXTERNRT 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

5.28.3.25 rtxUTF8StrnToSize()

```
EXTERNRT 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

5.28.3.26 rtxUTF8StrnToUInt()

```
EXTERNRT 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

5.28.3.27 rtxUTF8StrnToUInt64()

```
EXTERNRT 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

5.28.3.28 rtxUTF8StrRefOrDup()

```
EXTERNRT 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.

5.28.3.29 rtxUTF8StrToBool()

```
EXTERNRT 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

5.28.3.30 rtxUTF8StrToDouble()

```
EXTERNRT 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

5.28.3.31 rtxUTF8StrToDynHexStr()

```
EXTERNRT 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.

5.28.3.32 rtxUTF8StrToInt()

```
EXTERNRT 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

5.28.3.33 rtxUTF8StrToInt64()

```
EXTERNRT 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

5.28.3.34 rtxUTF8StrToNamedBits()

```
EXTERNRT 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

5.28.3.35 rtxUTF8StrToSize()

```
EXTERNRT 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

5.28.3.36 rtxUTF8StrToUInt()

```
EXTERNRT 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

5.28.3.37 rtxUTF8StrToUInt64()

```
EXTERNRT 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

5.28.3.38 rtxUTF8ToDynUniStr()

```
EXTERNRT int rtxUTF8ToDynUniStr (
            OSCTXT * pctxt,
            const OSUTF8CHAR * utf8str,
            const OSUNICHAR ** ppdata,
            OSUINT32 * pnchars )
```

This function converts the given UTF-8 string to a Unicode string (UTF-16).

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 (the size of ∗ppdata array).

Returns

Status: 0 = OK, negative value = error

5.28.3.39 rtxUTF8ToDynUniStr32()

```
EXTERNRT int rtxUTF8ToDynUniStr32 (
             OSCTXT * pctxt,
             const OSUTF8CHAR * utf8str,
             const OS32BITCHAR ** ppdata,
             OSUINT32 * pnchars )
```

This function converts the given UTF-8 string to a Unicode string (UTF-32).

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 (the size of ∗ppdata array).

Returns

Status: 0 = OK, negative value = error

5.28.3.40 rtxUTF8ToUnicode()

```
EXTERNRT 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.

5.28.3.41 rtxUTF8ToUnicode32()

```
EXTERNRT long rtxUTF8ToUnicode32 (
             OSCTXT * pctxt,
             const OSUTF8CHAR * inbuf,
             OS32BITCHAR * outbuf,
             size_t outbufsiz )
```

This function converts a UTF-8 string to a Unicode string (UTF-32).

The Unicode string is stored as an array of 32-bit characters (unsigned 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.

5.28.3.42 rtxValidateUTF8()

```
EXTERNRT 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.

Chapter 6

Class Documentation

6.1 _OSRTBufLocDescr Struct Reference

Buffer location descriptor.

```
#include <rtxBuffer.h>
```

6.1.1 Detailed Description

Buffer location descriptor.

Definition at line 50 of file rtxBuffer.h.

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

- rtxBuffer.h

6.2 _OSRTIntStack Struct Reference

This is the main stack structure.

```
#include <rtxIntStack.h>
```

6.2.1 Detailed Description

This is the main stack structure.

It uses an expandable array for storing integer values.

Definition at line 52 of file rtxIntStack.h.

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

- rtxIntStack.h

6.3 OSBitMapItem Struct Reference

Named bit in a bit map.

```
#include <osSysTypes.h>
```

6.3.1 Detailed Description

Named bit in a bit map.

This structure is used to equate a name with a bit in a bit map.

Definition at line 325 of file osSysTypes.h.

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

- osSysTypes.h

6.4 OSBufferIndex Struct Reference

This structure can be used as an index into the buffer.

```
#include <rtxContext.h>
```

6.4.1 Detailed Description

This structure can be used as an index into the buffer.

Definition at line 121 of file rtxContext.h.

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

- rtxContext.h

6.5 OSCTXT Struct Reference

Run-time context structure.

```
#include <rtxContext.h>
```

Public Attributes

- OSRTStack containerEndIndexStack

  Stack of OSBufferIndex, representing pointers to the end of currently open containers having length determinants.

6.5.1 Detailed Description

Run-time context structure.

This structure is a container structure that holds all working variables involved in encoding or decoding a message.

Definition at line 198 of file rtxContext.h.

6.5.2 Member Data Documentation

6.5.2.1 containerEndIndexStack

```
OSRTStack OSCTXT::containerEndIndexStack
```

Stack of OSBufferIndex, representing pointers to the end of currently open containers having length determinants.

Each OSBufferIndex represents the value of the buffer's byteIndex and bitOffset after the container has been decoded (i.e. the location of the first bit beyond the container). Used by 3GPP decoders.

Definition at line 227 of file rtxContext.h.

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

- rtxContext.h

6.6 OSDynOctStr Struct Reference

Dynamic binary string structure.

```
#include <osSysTypes.h>
```

6.6.1 Detailed Description

Dynamic binary string structure.

This structure is used in generated code for XSD hexBinary and base64Binary types.

Definition at line 272 of file osSysTypes.h.

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

- osSysTypes.h

6.7 OSNumDateTime Struct Reference

Numeric date/time structure.

```
#include <osSysTypes.h>
```

6.7.1 Detailed Description

Numeric date/time structure.

Definition at line 140 of file osSysTypes.h.

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

- osSysTypes.h

6.8 OSRTBuffer Struct Reference

Run-time message buffer structure.

```
#include <rtxContext.h>
```

6.8.1 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.

Definition at line 94 of file rtxContext.h.

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

- rtxContext.h

6.9 OSRTBufSave Struct Reference

Structure to save the current message buffer state.

```
#include <rtxContext.h>
```

6.9.1 Detailed Description

Structure to save the current message buffer state.

This structure is used to save the current state of the buffer.

Definition at line 111 of file rtxContext.h.

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

- rtxContext.h

6.10 OSRTDList Struct Reference

This is the main list structure.

```
#include <rtxDList.h>
```

Public Attributes

- OSSIZE count

  Count of items in the list.
- OSRTDListNode ∗ head

  Pointer to first entry in list.
- OSRTDListNode ∗ tail

  Pointer to last entry in list.

6.10.1 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.

Definition at line 64 of file rtxDList.h.

6.10.2 Member Data Documentation

6.10.2.1 count

```
OSSIZE OSRTDList::count
```

Count of items in the list.

Definition at line 65 of file rtxDList.h.

6.10.2.2 head

```
OSRTDListNode* OSRTDList::head
```

Pointer to first entry in list.

Definition at line 66 of file rtxDList.h.

6.10.2.3 tail

```
OSRTDListNode* OSRTDList::tail
```

Pointer to last entry in list.

Definition at line 67 of file rtxDList.h.

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

- rtxDList.h

6.11 OSRTDListNode Struct Reference

This structure is used to hold a single data item within the list.

```
#include <rtxDList.h>
```

Public Attributes

- void ∗ data

  Pointer to list data item.
- struct OSRTDListNode ∗ next

  Pointer to next node in list.
- struct OSRTDListNode ∗ prev

  Pointer to previous node in list.

6.11.1 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.

Definition at line 52 of file rtxDList.h.

6.11.2 Member Data Documentation

6.11.2.1 data

```
void* OSRTDListNode::data
```

Pointer to list data item.

Definition at line 53 of file rtxDList.h.

6.11.2.2 next

```
struct OSRTDListNode* OSRTDListNode::next
```

Pointer to next node in list.

Definition at line 54 of file rtxDList.h.

6.11.2.3 prev

```
struct OSRTDListNode* OSRTDListNode::prev
```

Pointer to previous node in list.

Definition at line 55 of file rtxDList.h.

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

- rtxDList.h

6.12 OSRTErrInfo Struct Reference

Run-time error information structure.

```
#include <rtxContext.h>
```

6.12.1 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.

Definition at line 67 of file rtxContext.h.

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

- rtxContext.h

6.13 OSRTErrLocn Struct Reference

Run-time error location structure.

```
#include <rtxContext.h>
```

6.13.1 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.

Definition at line 52 of file rtxContext.h.

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

- rtxContext.h

6.14 OSRTPrintStream Struct Reference

Structure to hold information about a global PrintStream.

```
#include <rtxPrintStream.h>
```

6.14.1 Detailed Description

Structure to hold information about a global PrintStream.

Definition at line 51 of file rtxPrintStream.h.

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

- rtxPrintStream.h

6.15 OSRTScalarDList Struct Reference

This is the main list structure.

```
#include <rtxScalarDList.h>
```

Public Attributes

- OSUINT32 count

  Count of items in the list.
- OSRTScalarDListNode ∗ head

  Pointer to first entry in list.
- OSRTScalarDListNode ∗ tail

  Pointer to last entry in list.

6.15.1 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.

Definition at line 92 of file rtxScalarDList.h.

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

- rtxScalarDList.h

6.16 OSRTScalarDListNode Struct Reference

This structure is used to hold a single data item within the list.

```
#include <rtxScalarDList.h>
```

Public Attributes

- struct OSRTScalarDListNode ∗ next

  Pointer to next node in list.
- struct OSRTScalarDListNode ∗ prev

  Pointer to previous node in list.
- OSDOUBLE dfltval

  Double prec floating point value.
- OSFLOAT fltval

  Single prec floating point value.
- OSINT32 i32val

  32-bit signed integer
- OSUINT32 ui32val

  32-bit unsigned integer
- OSINT64 i64val

  64-bit signed integer
- OSUINT64 ui64val

  64-bit unsigned integer

6.16.1 Detailed Description

This structure is used to hold a single data item within the list.

The data item is a union of all of the possible scalar types it can hold. The node also contains forward and backward pointers to the next and previous entries in the list.

Definition at line 68 of file rtxScalarDList.h.

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

- rtxScalarDList.h

6.17 OSRTSTREAM Struct Reference

The stream control block.

```
#include <rtxStream.h>
```

Public Attributes

- OSRTStreamReadProc read

  pointer to read function
- OSRTStreamWriteProc write

  pointer to write function
- OSRTStreamFlushProc flush

  pointer to flush function
- OSRTStreamCloseProc close

  pointer to close function
- OSRTStreamSkipProc skip

  pointer to skip function
- OSRTStreamMarkProc mark

  pointer to mark function
- OSRTStreamResetProc reset

  pointer to reset function
- OSRTStreamGetPosProc getPos

  pointer to getPos function
- OSRTStreamSetPosProc setPos

  pointer to setPos function
- void ∗ extra

  pointer to stream-specific data
- size_t bufsize

  physical size of pctxt->buffer.data buffer.
- size_t readAheadLimit

  read ahead limit (used by rtxStreamMark/rtxStreamReset
- size_t bytesProcessed

  the number of bytes processed by the application program
- size_t markedBytesProcessed

  the marked number of bytes already processed
- size_t ioBytes

  the actual number of bytes read from or written to the stream
- size_t nextMarkOffset

  offset of next appropriate mark position
- size_t segsize

  size of decoded segment
- OSUINT32 id

  id of stream (see OSRTSTRMID_∗ macros)
- OSRTMEMBUF ∗ pCaptureBuf

  Buffer into which data read from stream can be captured for debugging purposes.
- OSUINT16 flags

  flags (see OSRTSTRMF_∗ macros

6.17.1 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.

Definition at line 184 of file rtxStream.h.

6.17.2 Member Data Documentation

6.17.2.1 bufsize

```
size_t OSRTSTREAM::bufsize
```

physical size of pctxt->buffer.data buffer.

pctxt->buffer.size represents the logical size - the amount of actual data held in the buffer.

Definition at line 196 of file rtxStream.h.

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

- rtxStream.h

6.18 OSUTF8NameAndLen Struct Reference

UTF-8 name and length structure.

```
#include <osSysTypes.h>
```

6.18.1 Detailed Description

UTF-8 name and length structure.

This structure holds a pointer to UTF-8 text (it does not need to be null-terminated) and a variable containing the length of the string. The primary use of this structure is to improve performance in token matching by not having to calculate string length every time.

Definition at line 340 of file osSysTypes.h.

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

- osSysTypes.h

6.19 OSXMLFullQName Struct Reference

This version of QName contains complete namespace info (prefix + URI)

```
#include <rtxXmlQName.h>
```

6.19.1 Detailed Description

This version of QName contains complete namespace info (prefix + URI)

Definition at line 36 of file rtxXmlQName.h.

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

- rtxXmlQName.h

6.20 OSXMLQNameValue Struct Reference

This version of QName models the value space for XML Schema QNames, which consists of just a namespace URI and a local name.

```
#include <rtxXmlQName.h>
```

6.20.1 Detailed Description

This version of QName models the value space for XML Schema QNames, which consists of just a namespace URI and a local name.

The name OSXMLQName is already used for a type that consists of prefix and local name, which is the lexical representation.

Definition at line 48 of file rtxXmlQName.h.

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

- rtxXmlQName.h

6.21 OSXMLSTRING Struct Reference

XML UTF-8 character string structure.

```
#include <osSysTypes.h>
```

6.21.1 Detailed Description

XML UTF-8 character string structure.

This structure is used in generated code for XML string types.

Definition at line 304 of file osSysTypes.h.

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

- osSysTypes.h

6.22 OSXSDDateTime Struct Reference

Numeric date/time structure.

```
#include <osSysTypes.h>
```

6.22.1 Detailed Description

Numeric date/time structure.

This structure is used in generated code for XSD date/time types when code generation is configured to use numeric date/time types (-numDateTime command-line option).

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

- osSysTypes.h

Chapter 7

File Documentation

7.1 rtxArrayList.h File Reference

ArrayList functions.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT void rtxArrayListInit (OSRTArrayList *pArrayList, OSSIZE capacity)

  This function initializes an array list structure.
- EXTERNRT OSRTArrayList * rtxNewArrayList (OSCTXT *pctxt, OSSIZE capacity)

  This function creates a new array list to hold the initial capacity of elements.
- EXTERNRT void rtxFreeArrayList (OSCTXT *pctxt, OSRTArrayList *pArrayList)

  This function frees all dynamic memory held by the array list.
- EXTERNRT int rtxArrayListAdd (OSCTXT *pctxt, OSRTArrayList *pArrayList, void *pdata, OSSIZE *pindex)

  This function adds an element to an array list.
- EXTERNRT void rtxArrayListRemove (OSCTXT *pctxt, OSRTArrayList *pArrayList, void *pdata)

  This function removes an element from an array list.
- EXTERNRT void rtxArrayListRemoveIndexed (OSCTXT *pctxt, OSRTArrayList *pArrayList, OSSIZE idx)

  This function removes the element at the given index from the array list.
- EXTERNRT int rtxArrayListInsert (OSCTXT *pctxt, OSRTArrayList *pArrayList, void *pdata, OSSIZE idx)

  This function inserts an element at the given position in the array list.
- EXTERNRT int rtxArrayListReplace (OSRTArrayList *pArrayList, void *pdata, OSSIZE idx)

  This function replaces (overwrites) the element at the given position in the array list with the new element.
- EXTERNRT void * rtxArrayListGetIndexed (const OSRTArrayList *pArrayList, OSSIZE idx)

  This function gets the indexed data item from the array list.
- EXTERNRT int rtxArrayListIndexOf (OSRTArrayList *pArrayList, void *pdata)

  This function returns the index of the given data item in the list.
- EXTERNRT int rtxArrayListInitIter (OSRTArrayListIter *piter, const OSRTArrayList *pArrayList, OSSIZE startIndex)

  This function initializes an array list iterator with the given start index.
- EXTERNRT OSBOOL rtxArrayListHasNextItem (OSRTArrayListIter *piter)

  This function determines if another element exists at the next sequential position in the array list.
- EXTERNRT void * rtxArrayListNextItem (OSRTArrayListIter *piter)

  This function gets the next item from the array list.

7.1.1 Detailed Description

ArrayList functions.

7.1.2 Function Documentation

7.1.2.1 rtxArrayListAdd()

```
EXTERNRT int rtxArrayListAdd (
             OSCTXT * pctxt,
             OSRTArrayList * pArrayList,
             void * pdata,
             OSSIZE * pindex )
```

This function adds an element to an array list.

Parameters

pctxt	Pointer to a context structure.
pArrayList	Pointer to array list structure to initialize.
pdata	Pointer to data item to add.
pindex	Pointer to index variable to receive index at which entry was added.

Returns

Zero if item was successfully added; a negative status code if error.

7.1.2.2 rtxArrayListGetIndexed()

```
EXTERNRT void* rtxArrayListGetIndexed (
             const OSRTArrayList * pArrayList,
             OSSIZE idx )
```

This function gets the indexed data item from the array list.

Parameters

pArrayList	Pointer to array list structure to initialize.
idx	Index of location where item should be inserted.

Returns

Pointer to indexed data item or NULL if index is greater than max index in list.

7.1.2.3 rtxArrayListHasNextItem()

```
EXTERNRT OSBOOL rtxArrayListHasNextItem (
             OSRTArrayListIter * piter )
```

This function determines if another element exists at the next sequential position in the array list.

Parameters

piter	Pointer to array list iterator structure.

Returns

True if another element exists; false otherwise.

7.1.2.4 rtxArrayListIndexOf()

```
EXTERNRT int rtxArrayListIndexOf (
             OSRTArrayList * pArrayList,
             void * pdata )
```

This function returns the index of the given data item in the list.

The 'equals' callback function is used to do comparisons.

Parameters

pArrayList	Pointer to array list structure to initialize.
pdata	Pointer to data item to find in list.

Returns

Index of item in list or -1 if not found.

7.1.2.5 rtxArrayListInit()

```
EXTERNRT void rtxArrayListInit (
            OSRTArrayList * pArrayList,
            OSSIZE capacity )
```

This function initializes an array list structure.

Parameters

pArrayList	Pointer to array list structure to initialize.
capacity	Initial capacity of the array or zero to use default.

7.1.2.6 rtxArrayListInitIter()

```
EXTERNRT int rtxArrayListInitIter (
            OSRTArrayListIter * piter,
            const OSRTArrayList * pArrayList,
            OSSIZE startIndex )
```

This function initializes an array list iterator with the given start index.

Parameters

piter	Pointer to array list iterator structure.
pArrayList	Pointer to array list structure.
startIndex	Index from which iteration is to start.

Returns

Zero if successfully initialized or RTERR_OUTOFBND if start index is beyond the current size of the array list.

7.1.2.7 rtxArrayListInsert()

```
EXTERNRT int rtxArrayListInsert (
            OSCTXT * pctxt,
            OSRTArrayList * pArrayList,
            void * pdata,
            OSSIZE idx )
```

This function inserts an element at the given position in the array list.

Parameters

pctxt	Pointer to a context structure.
pArrayList	Pointer to array list structure to initialize.
pdata	Pointer to data item to insert.
idx	Index of location where item should be inserted.

Returns

Zero if item was successfully added; a negative status code if error.

7.1.2.8 rtxArrayListNextItem()

```
EXTERNRT void* rtxArrayListNextItem (
            OSRTArrayListIter * piter )
```

This function gets the next item from the array list.

Parameters

piter	Pointer to array list iterator structure.

Returns

Pointer to next item or null if beyond the end of the array.

7.1.2.9 rtxArrayListRemove()

```
EXTERNRT void rtxArrayListRemove (
            OSCTXT * pctxt,
            OSRTArrayList * pArrayList,
            void * pdata )
```

This function removes an element from an array list.

Parameters

pctxt	Pointer to a context structure.
pArrayList	Pointer to array list structure to initialize.
pdata	Pointer to data item to remove.

7.1.2.10 rtxArrayListRemoveIndexed()

```
EXTERNRT void rtxArrayListRemoveIndexed (
             OSCTXT * pctxt,
             OSRTArrayList * pArrayList,
             OSSIZE idx )
```

This function removes the element at the given index from the array list.

Parameters

pctxt	Pointer to a context structure.
pArrayList	Pointer to array list structure to initialize.
idx	Index of item to remove. -1 indicates tail item should be removed.

7.1.2.11 rtxArrayListReplace()

```
EXTERNRT int rtxArrayListReplace (
             OSRTArrayList * pArrayList,
             void * pdata,
             OSSIZE idx )
```

This function replaces (overwrites) the element at the given position in the array list with the new element.

Parameters

pArrayList	Pointer to array list structure to initialize.
pdata	Pointer to data item to insert.
idx	Index of location where item should be inserted.

Returns

Zero if item was successfully added; a negative status code if error.

7.1.2.12 rtxFreeArrayList()

```
EXTERNRT void rtxFreeArrayList (
             OSCTXT * pctxt,
             OSRTArrayList * pArrayList )
```

This function frees all dynamic memory held by the array list.

It does not free the array list structure itself, just the internal data array.

Parameters

pctxt	Pointer to a context structure.
pArrayList	Pointer to array list structure.

7.1.2.13 rtxNewArrayList()

```
EXTERNRT OSRTArrayList* rtxNewArrayList (
            OSCTXT * pctxt,
            OSSIZE capacity )
```

This function creates a new array list to hold the initial capacity of elements.

Parameters

pctxt	Pointer to a context structure.
capacity	Initial capacity of the array or zero to use default.

Returns

Allocated array list structure or NULL if insufficient dynamic memory is available to hold the structure.

7.2 rtxBase64.h File Reference

base64 and base64url are defined in RFC 4648.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT long rtxBase64EncodeData (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, OSOCTET ∗∗ppDstData)

  Encode binary data into base64 string form to a dynamic buffer.
- EXTERNRT long rtxBase64EncodeURLParam (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, O↩SOCTET ∗∗ppDstData)

  Encode binary data into base64 string form to a dynamic buffer, converting '+' characters to the URL escape sequence %2B so that the encoded string may be used in a query string parameter in a URL.
- EXTERNRT long rtxBase64DecodeData (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, OSOCTET ∗∗ppDstData)

  Decode base64 string to binary form into a dynamic buffer.

- EXTERNRT long rtxBase64DecodeDataToFSB (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, O↩ SOCTET ∗buf, size_t bufsiz)

    Decode base64 string to binary form into a fixed-size buffer.
- EXTERNRT long rtxBase64GetBinDataLen (const char ∗pSrcData, size_t srcDataSize)

    Calculate number of byte required to hold a decoded base64/base64url string in binary form.
- EXTERNRT long rtxBase64UrlEncodeData (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, OSO↩ CTET ∗∗ppDstData)

    Encode binary data into base64url string form to a dynamic buffer.
- EXTERNRT long rtxBase64UrlDecodeData (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, OSO↩ CTET ∗∗ppDstData)

    Decode base64url string to binary form into a dynamic buffer.
- EXTERNRT long rtxBase64UrlDecodeDataToFSB (OSCTXT ∗pctxt, const char ∗pSrcData, size_t srcDataSize, OSOCTET ∗buf, size_t bufsiz)

    Decode base64url string to binary form into a fixed-size buffer.
- EXTERNRT int rtxBase64CharToIdx (char c, OSBOOL url)

    Convert base64 character to index.
- EXTERNRT char rtxBase64IdxToChar (int idx, OSBOOL url)

    Convert base64 index to character.

7.2.1 Detailed Description

base64 and base64url are defined in RFC 4648.

7.2.2 Function Documentation

7.2.2.1 rtxBase64CharToIdx()

```
EXTERNRT int rtxBase64CharToIdx (
             char c,
             OSBOOL url )
```

Convert base64 character to index.

Parameters

c	Character to convert.
url	Flag indicating if base64 string is used in a URL

Returns

Converted integer value or -1 if invalid character.

7.2.2.2 rtxBase64DecodeData()

```
EXTERNRT 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.

7.2.2.3 rtxBase64DecodeDataToFSB()

```
EXTERNRT 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.

7.2.2.4 rtxBase64EncodeData()

```
EXTERNRT 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.

7.2.2.5 rtxBase64EncodeURLParam()

```
EXTERNRT long rtxBase64EncodeURLParam (
            OSCTXT * pctxt,
            const char * pSrcData,
            size_t srcDataSize,
            OSOCTET ** ppDstData )
```

Encode binary data into base64 string form to a dynamic buffer, converting '+' characters to the URL escape sequence %2B so that the encoded string may be used in a query string parameter in a URL.

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.

7.2.2.6 rtxBase64GetBinDataLen()

```
EXTERNRT long rtxBase64GetBinDataLen (
             const char * pSrcData,
             size_t srcDataSize )
```

Calculate number of byte required to hold a decoded base64/base64url string in binary form.

Parameters

pSrcData	Pointer to base64/base64url string.
srcDataSize	Length of the base64/base64url string.

Returns

Completion status of operation: If success, positive value is number of bytes, If failure, negative status code.

7.2.2.7 rtxBase64IdxToChar()

```
EXTERNRT char rtxBase64IdxToChar (
             int idx,
             OSBOOL url )
```

Convert base64 index to character.

Parameters

idx	Index to convert.
url	Flag indicating if base64 string is used in a URL

Returns

Converted character value or -1 if invalid index;

7.2.2.8 rtxBase64UrlDecodeData()

```
EXTERNRT long rtxBase64UrlDecodeData (
            OSCTXT * pctxt,
            const char * pSrcData,
            size_t srcDataSize,
            OSOCTET ** ppDstData )
```

Decode base64url 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.

7.2.2.9 rtxBase64UrlDecodeDataToFSB()

```
EXTERNRT long rtxBase64UrlDecodeDataToFSB (
            OSCTXT * pctxt,
            const char * pSrcData,
            size_t srcDataSize,
            OSOCTET * buf,
            size_t bufsiz )
```

Decode base64url 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.

7.2.2.10 rtxBase64UrlEncodeData()

```
EXTERNRT long rtxBase64UrlEncodeData (
            OSCTXT * pctxt,
            const char * pSrcData,
            size_t srcDataSize,
            OSOCTET ** ppDstData )
```

Encode binary data into base64url 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.

7.3 rtxBench.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/stat.h>
#include "rtxsrc/rtxDiag.h"
```

Functions

- double rtxBenchAverageMS (clock_t start, clock_t finish, double icnt)

  This function calculates the average number of milliseconds a test takes given the starting time, finishing time, and number of iterations.

7.3.1 Function Documentation

7.3.1.1 rtxBenchAverageMS()

```
double rtxBenchAverageMS (
             clock_t start,
             clock_t finish,
             double icnt )
```

This function calculates the average number of milliseconds a test takes given the starting time, finishing time, and number of iterations.

Parameters

start	The initial start time, as a clock_t structure.
finish	The final end time, as a clock_t structure.
icnt	The iteration count.

Returns

The average time a single iteration takes, in milliseconds.

7.4 rtxBigFloat.h File Reference

```
#include "rtxsrc/rtxBigInt.h"
```

Functions

- EXTERNRT void rtxBigFloatInit (OSBigFloat ∗pFloat)

  This function initializes a big float structure.
- EXTERNRT void rtxBigFloatFree (OSCTXT ∗pctxt, OSBigFloat ∗pFloat)

  This function frees internal memory within the big float structure.
- EXTERNRT int rtxBigFloatSetStr (OSCTXT ∗pctxt, OSBigFloat ∗pFloat, const char ∗value)

  This function sets an OSBigFloat from a null-terminated decimal string.
- EXTERNRT int rtxBigFloatSetStrn (OSCTXT ∗pctxt, OSBigFloat ∗pFloat, const char ∗value, OSSIZE len)

  This function sets an OSBigFloat from a character string using the given number of characters.
- EXTERNRT OSSIZE rtxBigFloatStringSize (OSBigFloat ∗pFloat)

  Return an approximate size for the string returned by rtxBigFloatToString.
- EXTERNRT OSBOOL rtxBigFloatToReal (const OSBigFloat ∗pFloat, OSREAL ∗pvalue)

  This function converts the given floating point number, which must be base 2, into an OSREAL, if this can be done exactly, without rounding.
- EXTERNRT int rtxBigFloatToString (OSCTXT ∗pctxt, const OSBigFloat ∗pFloat, char ∗str, OSSIZE strSize)

  This function is used to convert a floating point number to a string.

7.4.1 Function Documentation

7.4.1.1 rtxBigFloatFree()

```
EXTERNRT void rtxBigFloatFree (
             OSCTXT * pctxt,
             OSBigFloat * pFloat )
```

This function frees internal memory within the big float structure.

Parameters

pctxt	Pointer to a context structure.
pInt	Pointer to big float structure in which memory is to be freed.

7.4.1.2 rtxBigFloatInit()

```
EXTERNRT void rtxBigFloatInit (
             OSBigFloat * pFloat )
```

This function initializes a big float structure.

It must be called prior to working with the structure.

Parameters

pInt	Pointer to big float data structure.

7.4.1.3 rtxBigFloatSetStr()

```
EXTERNRT int rtxBigFloatSetStr (
             OSCTXT * pctxt,
             OSBigFloat * pFloat,
             const char * value )
```

This function sets an OSBigFloat from a null-terminated decimal string.

Parameters

pctxt	Pointer to a context structure.
pFloat	Pointer to big float structure to receive converted value. pFloat->base10 will be set TRUE.
value	Numeric string to convert. It must be in the following format: [-] integer [.fraction] [E|e [-] integer] There must be at least one integer digit before the optional fraction. Use of an exponent is optional. The exponent has an optional sign. Leading and trailing whitespace are acceptable and ignored; no other whitespace is allowed.

Returns

Status of the operation, 0 = success, negative code if error.

7.4.1.4 rtxBigFloatSetStrn()

```
EXTERNRT int rtxBigFloatSetStrn (
             OSCTXT * pctxt,
             OSBigFloat * pFloat,
             const char * value,
             OSSIZE len )
```

This function sets an OSBigFloat from a character string using the given number of characters.

Parameters

pctxt	Pointer to a context structure.
pFloat	Pointer to big float structure to receive converted value. pFloat->base10 will be set TRUE.
value	Numeric string to convert. It must be in the following format: [-] integer [.fraction] [E|e [-] integer] There must be at least one integer digit before the optional fraction. Use of an exponent is optional. The exponent has an optional sign. Leading and trailing whitespace are acceptable and ignored; no other whitespace is allowed.
len	Number of characters from character string to use.

Returns

Status of the operation, 0 = success, negative code if error.

7.4.1.5 rtxBigFloatStringSize()

```
EXTERNRT OSSIZE rtxBigFloatStringSize (
             OSBigFloat * pFloat )
```

Return an approximate size for the string returned by rtxBigFloatToString.

The value returned will be at least as large as needed to hold the result of using rtxBigFloatToString to convert to a string, including room for the null terminator.

7.4.1.6 rtxBigFloatToReal()

```
EXTERNRT OSBOOL rtxBigFloatToReal (
             const OSBigFloat * pFloat,
             OSREAL * pvalue )
```

This function converts the given floating point number, which must be base 2, into an OSREAL, if this can be done exactly, without rounding.

Returns

FALSE if the conversion cannot be done exactly.

7.4.1.7 rtxBigFloatToString()

```
EXTERNRT int rtxBigFloatToString (
             OSCTXT * pctxt,
             const OSBigFloat * pFloat,
             char * str,
             OSSIZE strSize )
```

This function is used to convert a floating point number to a string.

If pFloat->base10 is true, the format of the string will be: [-]ddd[E[-]ddd] That is, an integer mantissa, followed by an optional integer exponent, either of which may have a negative sign.

If pFloat->base10 is false, the result depends on whether the OSBigFloat can be converted exactly to a double (see rtxBigFloatToReal). If exact conversion can be done, the result is as for fprint using G. Otherwise, the result is inspired by fprint's A format: [-]0xHHHHHP{+/-]d,

Parameters

pctxt	Pointer to a context structure.
pFloat	Pointer to OSBigFloat structure to convert.
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.

7.5 rtxBigInt.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT void rtxBigIntInit (OSBigInt ∗pInt)

  This function initializes a big integer structure.
- EXTERNRT int rtxBigIntEnsureCapacity (OSCTXT ∗pctxt, OSBigInt ∗pInt, OSSIZE capacity)

  This function ensures that the big integer has capacity to hold at least the given number of bytes.
- EXTERNRT int rtxBigIntSetStr (OSCTXT ∗pctxt, OSBigInt ∗pInt, const char ∗value, int radix)

  This function sets a big integer binary value from a null-terminated string.
- EXTERNRT 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.
- EXTERNRT int rtxBigIntSetInt64 (OSCTXT ∗pctxt, OSBigInt ∗pInt, OSINT64 value)

  This function sets a big integer binary value from a signed 64-bit integer value.
- EXTERNRT int rtxBigIntSetUInt64 (OSCTXT ∗pctxt, OSBigInt ∗pInt, OSUINT64 value)

  This function sets a big integer binary value from an unsigned 64-bit integer value.
- EXTERNRT int rtxBigIntSetBytesSigned (OSCTXT ∗pctxt, OSBigInt ∗pInt, OSOCTET ∗value, OSSIZE vallen)

  This function sets a big integer binary value from a byte array.
- EXTERNRT int rtxBigIntSetBytesUnsigned (OSCTXT ∗pctxt, OSBigInt ∗pInt, OSOCTET ∗value, OSSIZE vallen)

  This function sets a big integer binary value from a byte array.
- EXTERNRT OSSIZE rtxBigIntGetDataLen (const OSBigInt ∗pInt)

  This function is used to get the size in bytes of the binary big integer data value.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxBigIntFastCopy (OSCTXT ∗pctxt, const OSBigInt ∗pSrc, OSBigInt ∗pDst)

  This function is used to copy one BigInt to another.
- EXTERNRT OSBOOL rtxBigIntToReal (const OSBigInt ∗pSrc, OSREAL ∗pvalue)

  This function converts the given big integer to a real, if it will fit without requiring rounding (i.e.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxBigIntToDynStr (OSCTXT ∗pctxt, const OSBigInt ∗pInt, int radix)

  This function is used to convert a binary big integer value to a dynamic string.
- EXTERNRT int rtxBigIntPrint (const OSUTF8CHAR ∗name, const OSBigInt ∗bigint, int radix)

  This function is used to print a big integer value to standard output.
- EXTERNRT int rtxBigIntCompare (const OSBigInt ∗arg1, const OSBigInt ∗arg2)

  This function is used to compare two big integer values.
- EXTERNRT int rtxBigIntStrCompare (OSCTXT ∗pctxt, const char ∗arg1, const char ∗arg2)

  This function is used to compare two big integer numeric strings.
- EXTERNRT void rtxBigIntFree (OSCTXT ∗pctxt, OSBigInt ∗pInt)

This function frees internal memory within the big integer structure.

- EXTERNRT int rtxBigIntAdd (OSCTXT ∗pctxt, OSBigInt ∗result, const OSBigInt ∗arg1, const OSBigInt ∗arg2)

  This function is used to add two big integer values.
- EXTERNRT int rtxBigIntSubtract (OSCTXT ∗pctxt, OSBigInt ∗result, const OSBigInt ∗arg1, const OSBigInt ∗arg2)

  This function is used to subtract one big integer value from another.
- EXTERNRT int rtxBigIntMultiply (OSCTXT ∗pctxt, OSBigInt ∗result, const OSBigInt ∗arg1, const OSBigInt ∗arg2)

  This function is used to multiply two big integer values.
- EXTERN OSSIZE rtxBase2DigitsToRadix (OSUINT8 radix, OSSIZE n)

  Return the number of radix digits that may be required to represent an integer having n base 2 digits (assuming the base 2 number is unsigned).
- EXTERN OSSIZE rtxRadixDigitsToBase2 (OSUINT8 radix, OSSIZE n)

  Return the number of base 2 digits that may be required to represent an integer having n radix digits (assuming the base 2 number is unsigned).
- EXTERNRT unsigned short rtxBigIntBitsPerDigit (int halfRadix)

  For internal use only.

7.5.1 Function Documentation

7.5.1.1 rtxBase2DigitsToRadix()

```
EXTERN OSSIZE rtxBase2DigitsToRadix (
             OSUINT8 radix,
             OSSIZE n )
```

Return the number of radix digits that may be required to represent an integer having n base 2 digits (assuming the base 2 number is unsigned).

This may return a larger value than the actual value.

Parameters

radix	8, 10, or 16
n	Number of base 2 digits; n $>$ 0.

7.5.1.2 rtxBigIntAdd()

```
EXTERNRT 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. Must have been initialized using rtxBigIntInit.
arg1	First big integer to add.
arg2	Second big integer to add.

Returns

Result of operation: 0 = success, negative error code if error.

7.5.1.3 rtxBigIntBitsPerDigit()

```
EXTERNRT unsigned short rtxBigIntBitsPerDigit (
             int halfRadix )
```

For internal use only.

The name is misleading. Documentation is in the .c file. Consider instead rtxRadixDigitsToBase2.

7.5.1.4 rtxBigIntCompare()

```
EXTERNRT 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

7.5.1.5 rtxBigIntCopy()

```
EXTERNRT 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.

7.5.1.6 rtxBigIntDigitsNum()

```
EXTERNRT 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.

7.5.1.7 rtxBigIntEnsureCapacity()

```
EXTERNRT int rtxBigIntEnsureCapacity (
             OSCTXT * pctxt,
             OSBigInt * pInt,
             OSSIZE capacity )
```

This function ensures that the big integer has capacity to hold at least the given number of bytes.

This can be used prior to directly copying bytes into mag. POST: On a successful return, pInt->allocated <= capacity; pInt->dynamic might now be TRUE. The value of the big integer is unchanged.

Parameters

pctxt	Pointer to a context structure.
pInt	Pointer to big integer structure.
capacity	Required capacity.

Returns

Status of the operation, 0 = success, negative code if error.

7.5.1.8 rtxBigIntFastCopy()

```
EXTERNRT 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.

7.5.1.9 rtxBigIntFree()

```
EXTERNRT 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.

7.5.1.10 rtxBigIntGetData()

```
EXTERNRT 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.

7.5.1.11 rtxBigIntGetDataLen()

```
EXTERNRT 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.

7.5.1.12 rtxBigIntInit()

```
EXTERNRT 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.

7.5.1.13 rtxBigIntMultiply()

```
EXTERNRT 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.

7.5.1.14 rtxBigIntPrint()

```
EXTERNRT 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.

7.5.1.15 rtxBigIntSetBytesSigned()

```
EXTERNRT int rtxBigIntSetBytesSigned (
             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 2's complement form.

Parameters

pctxt	Pointer to a context structure.
pInt	Pointer to big integer structure to receive converted value.
value	Buffer containing binary integer value in 2's complement form.
vallen	Number of byte in the value buffer.

Returns

Status of the operation, 0 = success, negative code if error.

7.5.1.16 rtxBigIntSetBytesUnsigned()

```
EXTERNRT int rtxBigIntSetBytesUnsigned (
             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 unsigned form.

Parameters

pctxt	Pointer to a context structure.
pInt	Pointer to big integer structure to receive converted value.
value	Buffer containing binary unsigned integer value.
vallen	Number of byte in the value buffer.

Returns

Status of the operation, 0 = success, negative code if error.

7.5.1.17 rtxBigIntSetInt64()

```
EXTERNRT 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.

7.5.1.18 rtxBigIntSetStr()

```
EXTERNRT 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 0, 2, 8, 10, or 16. Zero must be used if string contains a prefix that identifies the radix (for example, 0x).

Returns

Status of the operation, 0 = success, negative code if error.

7.5.1.19 rtxBigIntSetStrn()

```
EXTERNRT 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 0, 2, 8, 10, or 16. Zero must be used if string contains a prefix that identifies the radix (for example, 0x).

Returns

Status of the operation, 0 = success, negative code if error.

7.5.1.20 rtxBigIntSetUInt64()

```
EXTERNRT 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.

7.5.1.21 rtxBigIntStrCompare()

```
EXTERNRT 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

7.5.1.22 rtxBigIntSubtract()

```
EXTERNRT 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.

7.5.1.23 rtxBigIntToDynStr()

```
EXTERNRT char* rtxBigIntToDynStr (
             OSCTXT * pctxt,
             const OSBigInt * pInt,
             int radix )
```

This function is used to convert a binary big integer value to a dynamic string.

Memory for the string is allocated using the rtxMemAlloc function. It may be freed using rtxMemFreePtr or when the context if freed.

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.

Returns

Character pointer to converted string value or null if error.

7.5.1.24 rtxBigIntToReal()

```
EXTERNRT OSBOOL rtxBigIntToReal (
             const OSBigInt * pSrc,
             OSREAL * pvalue )
```

This function converts the given big integer to a real, if it will fit without requiring rounding (i.e.

loss of precision).

Parameters

pvalue	Receives the OSREAL value. If null, this function only tests whether the conversion can be done.

Returns

TRUE/FALSE indicating whether the value can fit in a OSREAL without rounding.

7.5.1.25 rtxBigIntToString()

```
EXTERNRT 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.

7.5.1.26 rtxRadixDigitsToBase2()

```
EXTERN OSSIZE rtxRadixDigitsToBase2 (
             OSUINT8 radix,
             OSSIZE n )
```

Return the number of base 2 digits that may be required to represent an integer having n radix digits (assuming the base 2 number is unsigned).

This may return a larger value than the actual value.

Parameters

radix	8, 10, or 16

7.6 rtxBigNumber.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- int rtxAddBigNum (const OSOCTET ∗a, OSSIZE szA, const OSOCTET ∗b, OSSIZE szB, OSOCTET ∗c, OSSIZE szC)

  Addition big numbers: a + b = c.
- int rtxSubBigNum (const OSOCTET ∗a, OSSIZE szA, const OSOCTET ∗b, OSSIZE szB, OSOCTET ∗c, OSSIZE szC)

  Substraction big numbers: a - b = c.
- int rtxMulBigNum (const OSOCTET ∗a, OSSIZE szA, const OSOCTET ∗b, OSSIZE szB, OSOCTET ∗c, OSSIZE szC)

  Multiplication big numbers: a ∗ b = c.
- int rtxDivRemBigNum (const OSOCTET ∗a, OSSIZE szA, const OSOCTET ∗b, OSSIZE szB, OSOCTET ∗c, OSSIZE szC, OSOCTET ∗rem, OSSIZE szRem)

  Division big numbers with reminder: a / b = c.
- int rtxDivBigNum (const OSOCTET ∗a, OSSIZE szA, const OSOCTET ∗b, OSSIZE szB, OSOCTET ∗c, OSSIZE szC)

  Division big numbers: a / b = c.
- int rtxModBigNum (const OSOCTET ∗a, OSSIZE szA, const OSOCTET ∗b, OSSIZE szB, OSOCTET ∗rem, O↩ SSIZE szRem)

  Division by module big numbers: a % b = rem.
- int rtxBigNumToStr (const OSOCTET ∗a, OSSIZE szA, char ∗str, OSSIZE szStr)

  Convert big number to string.
- int rtxStrToBigNum (const char ∗str, OSOCTET ∗a, OSSIZE szA)

  Convert string to big number.

7.6.1 Function Documentation

7.6.1.1 rtxAddBigNum()

```
int rtxAddBigNum (
              const OSOCTET * a,
              OSSIZE szA,
              const OSOCTET * b,
              OSSIZE szB,
              OSOCTET * c,
              OSSIZE szC )
```

Addition big numbers: a + b = c.

Parameters

a	First addend.
szA	Length of first addend in octets.
b	Second addend.
szB	Length of second addend in octets.
c	Sum.
szC	Length of sum buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.2 rtxBigNumToStr()

```
int rtxBigNumToStr (
             const OSOCTET * a,
             OSSIZE szA,
             char * str,
             OSSIZE szStr )
```

Convert big number to string.

Parameters

a	Number.
szA	Length of number in octets.
str	Result string.
szStr	Length of string buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.3 rtxDivBigNum()

```
int rtxDivBigNum (
             const OSOCTET * a,
             OSSIZE szA,
             const OSOCTET * b,
             OSSIZE szB,
             OSOCTET * c,
             OSSIZE szC )
```

Division big numbers: a / b = c.

Parameters

a	Divident.
szA	Length of divident in octets.
b	Divisor.
szB	Length of divisor in octets.
c	Quotient.
szC	Length of quotient buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.4 rtxDivRemBigNum()

```
int rtxDivRemBigNum (
             const OSOCTET * a,
             OSSIZE szA,
             const OSOCTET * b,
             OSSIZE szB,
             OSOCTET * c,
             OSSIZE szC,
             OSOCTET * rem,
             OSSIZE szRem )
```

Division big numbers with reminder: a / b = c.

Parameters

a	Divident.
szA	Length of divident in octets.
b	Divisor.
szB	Length of divisor in octets.
c	Quotient.
szC	Length of quotient buffer in octets.
rem	Reminder.
szRem	Length of reminder buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.5 rtxModBigNum()

```
int rtxModBigNum (
             const OSOCTET * a,
             OSSIZE szA,
             const OSOCTET * b,
             OSSIZE szB,
             OSOCTET * rem,
             OSSIZE szRem )
```

Division by module big numbers: a % b = rem.

Parameters

a	Divident.
szA	Length of divident in octets.
b	Divisor.
szB	Length of divisor in octets.
rem	Reminder.
szRem	Length of reminder buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.6 rtxMulBigNum()

```
int rtxMulBigNum (
             const OSOCTET * a,
             OSSIZE szA,
             const OSOCTET * b,
             OSSIZE szB,
             OSOCTET * c,
             OSSIZE szC )
```

Multiplication big numbers: a $*$ b = c.

Parameters

a	Multiplicand.
szA	Length of multiplicand in octets.
b	Multiplier.
szB	Length of multiplier in octets.
c	Product.
szC	Length of product buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.7 rtxStrToBigNum()

```
int rtxStrToBigNum (
             const char * str,
             OSOCTET * a,
             OSSIZE szA )
```

Convert string to big number.

Parameters

str	Input null terminated string.
a	Result number.
szA	Length of number buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.6.1.8 rtxSubBigNum()

```
int rtxSubBigNum (
             const OSOCTET * a,
             OSSIZE szA,
             const OSOCTET * b,
             OSSIZE szB,
             OSOCTET * c,
             OSSIZE szC )
```

Substraction big numbers: a - b = c.

Parameters

a	Minuend.
szA	Length of minuend in octets.
b	Substrahend.
szB	Length of substrahend in octets.
c	Difference.
szC	Length of difference buffer in octets.

Returns

Status of the operation. Zero if successful; a negative status code if overflow.

7.7 rtxBitDecode.h File Reference

Bit decode functions.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxDecBit (OSCTXT ∗pctxt, OSBOOL ∗pvalue)

  This function will decode a single bit and return the result in an OSBOOL value.
- EXTERNRT int rtxDecBits (OSCTXT ∗pctxt, OSUINT32 ∗pvalue, OSSIZE nbits)

  This function decodes up to sizeof(unsigned) bits and returns the result in an unsigned integer value.
- EXTERNRT int rtxDecBitsToSize (OSCTXT ∗pctxt, OSSIZE ∗pvalue, OSSIZE nbits)

  This function decodes up to sizeof(size_t) bits and returns the result in a size-typed value.
- EXTERNRT int rtxDecBitsToByte (OSCTXT ∗pctxt, OSUINT8 ∗pvalue, OSUINT8 nbits)

  This function decodes bits and returns the result in a byte (octet) value.
- EXTERNRT int rtxDecBitsToUInt16 (OSCTXT ∗pctxt, OSUINT16 ∗pvalue, OSSIZE nbits)

  This function decodes bits and returns the result in an unsigned 16-bit (short) value.
- EXTERNRT int rtxDecBitsToUInt64 (OSCTXT ∗pctxt, OSUINT64 ∗pvalue, OSSIZE nbits)

  This function decodes bits and returns the result in an unsigned 64-bit value.
- EXTERNRT int rtxDecBitsToByteArray (OSCTXT ∗pctxt, OSOCTET ∗pbuffer, OSSIZE bufsiz, OSSIZE nbits)

  This function decodes bits and returns the result in an octet array.
- EXTERNRT int rtxPeekBit (OSCTXT ∗pctxt, OSBOOL ∗pvalue)

  This function decodes the bit at the current position and the resets the bit cursor back to the original position.
- EXTERNRT int rtxPeekBits (OSCTXT ∗pctxt, OSUINT32 ∗pvalue, OSSIZE nbits)

  This function decodes up to sizeof(unsigned) bits and returns the result in an unsigned integer value.
- EXTERNRT int rtxSkipBits (OSCTXT ∗pctxt, OSSIZE nbits)

  This function skips the given number of bits.
- EXTERNRT int rtxSkipBytes (OSCTXT ∗pctxt, OSSIZE nbytes)

  This function skips the given number of bytes worth of input.

7.7.1 Detailed Description

Bit decode functions.

7.7.2 Function Documentation

7.7.2.1 rtxDecBit()

```
EXTERNRT int rtxDecBit (
            OSCTXT * pctxt,
            OSBOOL * pvalue )
```

This function will decode a single bit and return the result in an OSBOOL value.

Parameters

pctxt	A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue	A pointer to the BOOLEAN value to receive the decoded result. A null pointer value may be passed to skip the bit.

Returns

Completion status of operation:

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

7.7.2.2 rtxDecBits()

```
EXTERNRT int rtxDecBits (
             OSCTXT * pctxt,
             OSUINT32 * pvalue,
             OSSIZE nbits )
```

This function decodes up to sizeof(unsigned) bits and returns the result in an unsigned integer value.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue	Pointer to value to be receive decoded result.
nbits	Number of bits to read from decode buffer. If zero, the output value is set to zero.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.3 rtxDecBitsToByte()

```
EXTERNRT int rtxDecBitsToByte (
             OSCTXT * pctxt,
             OSUINT8 * pvalue,
             OSUINT8 nbits )
```

This function decodes bits and returns the result in a byte (octet) value.

Bits are shifted to the right in the decode byte to remove unused bits. For example, if a single bit is decoded from octet 0x80, the decoded byte value will be 1.

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.
pvalue	Pointer to byte value to receive decoded data.
nbits	Number of bits to read from decode buffer. The maximum that can be read is eight.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.4 rtxDecBitsToByteArray()

```
EXTERNRT int rtxDecBitsToByteArray (
             OSCTXT * pctxt,
             OSOCTET * pbuffer,
             OSSIZE bufsiz,
             OSSIZE nbits )
```

This function decodes bits and returns the result in an octet array.

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.
pbuffer	Address of buffer to receive decoded binary data.
bufsiz	Size of output buffer.
nbits	Number of bits to read from decode buffer.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.5 rtxDecBitsToSize()

```
EXTERNRT int rtxDecBitsToSize (
             OSCTXT * pctxt,
             OSSIZE * pvalue,
             OSSIZE nbits )
```

This function decodes up to sizeof(size_t) bits and returns the result in a size-typed value.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue	Pointer to value to be receive decoded result.
nbits	Number of bits to read from decode buffer. If zero, the output value is set to zero.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.6 rtxDecBitsToUInt16()

```
EXTERNRT int rtxDecBitsToUInt16 (
            OSCTXT * pctxt,
            OSUINT16 * pvalue,
            OSSIZE nbits )
```

This function decodes bits and returns the result in an unsigned 16-bit (short) value.

Bits are shifted to the right in the decode byte to remove unused bits. For example, if a single bit is decoded from octet 0x80, the decoded byte value will be 1.

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.
pvalue	Pointer to byte value to receive decoded data.
nbits	Number of bits to read from decode buffer. The maximum that can be read is sixteen. If zero, the output value is set to zero.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.7 rtxDecBitsToUInt64()

```
EXTERNRT int rtxDecBitsToUInt64 (
            OSCTXT * pctxt,
            OSUINT64 * pvalue,
            OSSIZE nbits )
```

This function decodes bits and returns the result in an unsigned 64-bit value.

Bits are shifted to the right in the decode byte to remove unused bits. For example, if a single bit is decoded from octet 0x80, the decoded byte value will be 1.

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.
pvalue	Pointer to byte value to receive decoded data.
nbits	Number of bits to read from decode buffer. The maximum that can be read is sixteen. If zero, the output value is set to zero.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.8 rtxPeekBit()

```
EXTERNRT int rtxPeekBit (
            OSCTXT * pctxt,
            OSBOOL * pvalue )
```

This function decodes the bit at the current position and the resets the bit cursor back to the original position.

Parameters

pctxt	A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue	A pointer to the BOOLEAN value to receive the decoded result. A null pointer value may be passed to skip the bit.

Returns

Completion status of operation:

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

7.7.2.9 rtxPeekBits()

```
EXTERNRT int rtxPeekBits (
            OSCTXT * pctxt,
            OSUINT32 * pvalue,
            OSSIZE nbits )
```

This function decodes up to sizeof(unsigned) bits and returns the result in an unsigned integer value.

The bit cursor is returned to the orignal starting position.

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.
pvalue	Pointer to value to be receive decoded result.
nbits	Number of bits to read from decode buffer. If zero, the output value is set to zero.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.10 rtxSkipBits()

```
EXTERNRT int rtxSkipBits (
             OSCTXT * pctxt,
             OSSIZE nbits )
```

This function skips the given number of bits.

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.
nbits	Number of bits to skip.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.7.2.11 rtxSkipBytes()

```
EXTERNRT int rtxSkipBytes (
             OSCTXT * pctxt,
             OSSIZE nbytes )
```

This function skips the given number of bytes worth of input.

Regardless of whether the buffer is aligned on a byte boundary or not, skipping a byte of input means skipping 8 bits of input.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
nbytes	Number of bytes to skip.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.8 rtxBitEncode.h File Reference

Bit encode functions.

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define rtxEncByteAlignPattern(pctxt, pattern)

  This macro will byte-align the context buffer by encoding according to the given pattern.

Functions

- EXTERNRT int rtxEncBitsPattern (OSCTXT ∗pctxt, OSUINT8 pattern, OSSIZE nbits)

  This function encodes the given number of bits using a repeating bit pattern.
- EXTERNRT int rtxEncBit (OSCTXT ∗pctxt, OSBOOL value)

  This function will set the bit at the current encode bit cursor position to 1 or 0 and advance the cursor pointer.
- EXTERNRT int rtxEncBits (OSCTXT ∗pctxt, OSUINT32 value, OSSIZE nbits)

  This function encodes a number of the least significant bits (up to 32) of an unsigned integer value.
- EXTERNRT int rtxEncBitsFromSize (OSCTXT ∗pctxt, OSSIZE value, OSSIZE nbits)

  This function encodes a number of the least significant bits from a size type.
- EXTERNRT int rtxEncBitsFromByteArray (OSCTXT ∗pctxt, const OSOCTET ∗pvalue, OSSIZE nbits)

  This function will encode a series of bits from an octet array.
- EXTERNRT int rtxCopyBits (OSCTXT ∗pctxt, const OSOCTET ∗pvalue, OSSIZE nbits, OSUINT32 bitOffset)

  This function will encode a series of bits from an octet array.
- EXTERNRT int rtxMergeBits (OSCTXT ∗pctxt, OSUINT32 value, OSSIZE nbits)

  This function will merge a series of bits (up to 32) from an unsigned integer value into an existing encoded data buffer.

7.8.1 Detailed Description

Bit encode functions.

7.8.2 Macro Definition Documentation

7.8.2.1 rtxEncByteAlignPattern

```
#define rtxEncByteAlignPattern(
              pctxt,
              pattern )
```

Value:

```
if ((pctxt)->buffer.bitOffset != 8) { \
   rtxEncBits(pctxt, pattern, (pctxt)->buffer.bitOffset); }
```

This macro will byte-align the context buffer by encoding according to the given pattern.

If the buffer is not aligned, the lowest bits of the current byte, which have not be written already, will be set to the corresponding lowest bites of the pattern.

Definition at line 44 of file rtxBitEncode.h.

7.8.3 Function Documentation

7.8.3.1 rtxCopyBits()

```
EXTERNRT int rtxCopyBits (
             OSCTXT * pctxt,
             const OSOCTET * pvalue,
             OSSIZE nbits,
             OSUINT32 bitOffset )
```

This function will encode a series of bits from an octet array.

Encoding started from specified bit offset.

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.
pvalue	Pointer to bit string to be encoded.
nbits	Number of bits from the value to encode.
bitOffset	Starting bit offset.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.8.3.2 rtxEncBit()

```
EXTERNRT int rtxEncBit (
            OSCTXT * pctxt,
            OSBOOL value )
```

This function will set the bit at the current encode bit cursor position to 1 or 0 and advance the cursor pointer.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
value	The value to be encoded.

7.8.3.3 rtxEncBits()

```
EXTERNRT int rtxEncBits (
            OSCTXT * pctxt,
            OSUINT32 value,
            OSSIZE nbits )
```

This function encodes a number of the least significant bits (up to 32) of an unsigned integer value.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
value	The value to be encoded.
nbits	Number of bits from the value to encode.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.8.3.4 rtxEncBitsFromByteArray()

```
EXTERNRT int rtxEncBitsFromByteArray (
            OSCTXT * pctxt,
            const OSOCTET * pvalue,
            OSSIZE nbits )
```

This function will encode a series of bits from an octet array.

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.
pvalue	Pointer to bit string to be encoded.
nbits	Number of bits from the value to encode.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.8.3.5 rtxEncBitsFromSize()

```
EXTERNRT int rtxEncBitsFromSize (
            OSCTXT * pctxt,
            OSSIZE value,
            OSSIZE nbits )
```

This function encodes a number of the least significant bits from a size type.

The maximum that may be encoded is the maximum held by the type (32 or 64 bits).

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
value	The value to be encoded.
nbits	Number of bits from the value to encode.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.8.3.6 rtxEncBitsPattern()

```
EXTERNRT int rtxEncBitsPattern (
            OSCTXT * pctxt,
            OSUINT8 pattern,
            OSSIZE nbits )
```

This function encodes the given number of bits using a repeating bit pattern.

This function may be used to encode any number of bits (including more than 8 bits). It will take the bit values to encode from the pattern, in accordance with the buffer's bit offset. For example, if the next bit to encode is the highest bit of the next byte in the buffer, then the bit value encoded will be the highest bit of the pattern.

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.
pattern	The repeating pattern.
bits	The number of bits to encode.

7.8.3.7 rtxMergeBits()

```
EXTERNRT int rtxMergeBits (
            OSCTXT * pctxt,
            OSUINT32 value,
            OSSIZE nbits )
```

This function will merge a series of bits (up to 32) from an unsigned integer value into an existing encoded data buffer.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
value	The value to be encoded.
nbits	Number of bits from the value to merge.

Returns

Status of the operation. Zero if successful; a negative status code if failed.

7.9 rtxBitString.h File Reference

Contains utility functions for setting, clearing, and testing bits at any position in an arbitrarily sized array of bytes.

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define OSRTBYTEARRAYSIZE(numbits) ((numbits+7)/8)

  This macro is used to calculate the byte array size required to hold the given number of bits.

Functions

- EXTERNRT OSUINT32 rtxGetBitCount (OSUINT32 value)

  This function returns the minimum size of the bit field required to hold the given integer value.
- EXTERNRT int rtxSetBit (OSOCTET ∗pBits, OSSIZE numbits, OSSIZE bitIndex)

  This function sets the specified zero-counted bit in the bit string.
- EXTERNRT 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.
- EXTERNRT int rtxClearBit (OSOCTET ∗pBits, OSSIZE numbits, OSSIZE bitIndex)

  This function clears the specified zero-counted bit in the bit string.
- EXTERNRT OSBOOL rtxTestBit (const OSOCTET ∗pBits, OSSIZE numbits, OSSIZE bitIndex)

  This function tests the specified zero-counted bit in the bit string.
- EXTERNRT OSSIZE rtxLastBitSet (const OSOCTET ∗pBits, OSSIZE numbits)

  This function returns the zero-counted index of the last bit set in a bit string.
- EXTERNRT int rtxCheckBitBounds (OSCTXT ∗pctxt, OSOCTET ∗∗ppBits, OSSIZE ∗pNumocts, OSSIZE min↩ RequiredBits, OSSIZE preferredLimitBits)

  Check whether the given bit string is large enough, and expand it if necessary.
- EXTERNRT int rtxZeroUnusedBits (OSOCTET ∗pBits, OSSIZE numbits)

  This function zeros unused bits at the end of the given bit string.
- EXTERNRT int rtxCheckUnusedBitsZero (const OSOCTET ∗pBits, OSSIZE numbits)

  This function checks to see if unused bits at the end of the given bit string are zero.

7.9.1 Detailed Description

Contains utility functions for setting, clearing, and testing bits at any position in an arbitrarily sized array of bytes.

7.10 rtxBuffer.h File Reference

Common runtime functions for reading from or writing to the message buffer defined within the context structure.

```
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxSList.h"
```

Classes

- struct _OSRTBufLocDescr

  Buffer location descriptor.

Typedefs

- typedef struct _OSRTBufLocDescr OSRTBufLocDescr

  Buffer location descriptor.

Functions

- EXTERNRT int rtxCheckOutputBuffer (OSCTXT ∗pctxt, size_t nbytes)

  This function checks to ensure that the output buffer has sufficient space to hold an additional nbytes full bytes (if there is a partially filled byte, it is treated as though full).
- EXTERNRT OSBOOL rtxIsOutputBufferFlushable (OSCTXT ∗pctxt)

  This function returns true if the context buffer can be flushed to a stream by calling rtxFlushOutputBuffer.
- EXTERNRT int rtxFlushOutputBuffer (OSCTXT ∗pctxt)

  This function flushes the buffer to a stream.
- EXTERNRT int rtxExpandOutputBuffer (OSCTXT ∗pctxt, size_t nbytes)

  This function attempts to ensure the output buffer has at least the given number of bytes available.
- EXTERNRT int rtxCheckInputBuffer (OSCTXT ∗pctxt, size_t nbytes)

  Ensures the given number of bytes are available in the context buffer.
- EXTERNRT int rtxLoadInputBuffer (OSCTXT ∗pctxt, OSSIZE nbytes)

  This is for meant for internal use by the runtime.
- EXTERNRT int rtxPeekByte (OSCTXT ∗pctxt, OSOCTET ∗pbyte)

  This function peeks at the next byte of input, if there is one before EOF.
- EXTERNRT int rtxPeekBytes (OSCTXT ∗pctxt, OSOCTET ∗pdata, OSSIZE bufsize, OSSIZE nocts, OSSIZE ∗pactual)

  This function peeks at the next nocts bytes of input, peeking at fewer bytes if EOF is encountered first.
- EXTERNRT int rtxReadBytesSafe (OSCTXT ∗pctxt, OSOCTET ∗buffer, size_t bufsize, size_t nocts)

  This function safely reads bytes from the currently open stream or memory buffer into the given static buffer.
- EXTERNRT int rtxReadBytes (OSCTXT ∗pctxt, OSOCTET ∗pdata, size_t nocts)

  This function reads bytes from the currently open stream or memory buffer.
- EXTERNRT int rtxReadBytesDynamic (OSCTXT ∗pctxt, OSOCTET ∗∗ppdata, size_t nocts, OSBOOL ∗pMem←Alloc)

  This function reads bytes from the currently open stream or memory buffer.
- EXTERNRT int rtxWriteBytes (OSCTXT ∗pctxt, const OSOCTET ∗pdata, size_t nocts)

  This function writes bytes to the currently open stream or memory buffer.
- EXTERNRT int rtxWriteIndent (OSCTXT ∗pctxt)

  This function writes a newline followed by indentation whitespace to the buffer.
- EXTERNRT void rtxIndentDecr (OSCTXT ∗pctxt)

  This decreases the indentation level set in the given context by updating the indent member.
- EXTERNRT void rtxIndentIncr (OSCTXT ∗pctxt)

  This increases the indentation level set in the given context by updating the indent member.
- EXTERNRT void rtxIndentReset (OSCTXT ∗pctxt)

  This resets the indentation level in the given context to zero.
- EXTERNRT size_t rtxGetIndentLevels (OSCTXT ∗pctxt)

  This returns the number of levels of indentation set in the given context.
- EXTERNRT OSBOOL rtxCanonicalSort (OSOCTET ∗refPoint, OSRTSList ∗pList, OSBOOL normal)

  Sort a list of buffer locations, referring to component encodings, by comparing the referenced encodings as octet strings.
- EXTERNRT int rtxEncCanonicalSort (OSCTXT ∗pctxt, OSCTXT ∗pMemCtxt, OSRTSList ∗pList)

Encode the encodings held in pMemCtxt into pctxt, first sorting them as required for canonical BER (and other encoding rules) by X.690 11.6.

- EXTERNRT void rtxGetBufLocDescr (OSCTXT ∗pctxt, OSRTBufLocDescr ∗pDescr)

  Set the buffer location description's offset (pDescr->offset) to the current position in pCtxt's buffer.
- EXTERNRT void rtxAddBufLocDescr (OSCTXT ∗pctxt, OSRTSList ∗pElemList, OSRTBufLocDescr ∗pDescr)

  Create a new Asn1BufLocDescr for an element just encoded and append it to pElemList.

7.10.1 Detailed Description

Common runtime functions for reading from or writing to the message buffer defined within the context structure.

7.11 rtxCharStr.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxStricmp (const char ∗str1, const char ∗str2)

  This is an implementation of the non-standard stricmp function.
- EXTERNRT int rtxStrnicmp (const char ∗str1, const char ∗str2, size_t count)

  This is an implementation of the non-standard stricmp function.
- EXTERNRT char ∗ rtxStrcat (char ∗dest, size_t bufsiz, const char ∗src)

  This function concatanates the given string onto the string buffer.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxStrcpy (char ∗dest, size_t bufsiz, const char ∗src)

  This function copies a null-terminated string to a target buffer.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxStrdup (OSCTXT ∗pctxt, const char ∗src)

  This function creates a duplicate copy of a null-terminated string.
- EXTERNRT char ∗ rtxStrndup (OSCTXT ∗pctxt, const char ∗src, OSSIZE nchars)

  This function creates a duplicate copy of up to the given number of characters in a string.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT char ∗ rtxStrTrimEnd (char ∗s)

  This function trims whitespace from the end of a string.
- EXTERNRT int rtxValidateConstrainedStr (OSCTXT ∗pctxt, const char ∗pvalue, const char ∗pCharSet)

  This function will validate a constrained character string by checking to see if all the characters in the given string are contained within the

constraining character set.

- EXTERNRT int rtxIntToCharStr (OSINT32 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts a signed 32-bit integer into a character string.
- EXTERNRT int rtxUIntToCharStr (OSUINT32 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts an unsigned 32-bit integer into a character string.
- EXTERNRT int rtxInt64ToCharStr (OSINT64 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts a signed 64-bit integer into a character string.
- EXTERNRT int rtxUInt64ToCharStr (OSUINT64 value, char ∗dest, size_t bufsiz, char padchar)

  This function converts an unsigned 64-bit integer into a character string.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxHexCharsToBin (const char ∗hexstr, size_t nchars, OSOCTET ∗binbuf, size_t bufsize)

  This function converts the given hex string to binary.
- EXTERNRT int rtxCharStrToInt (const char ∗cstr, OSINT32 ∗pvalue)

  This function converts the given character string to a signed 32-bit integer value.
- EXTERNRT int rtxCharStrnToInt (const char ∗cstr, OSSIZE ndigits, OSINT32 ∗pvalue)

  This function converts up to the given number of digits from the given character string to a signed 32-bit integer value.
- EXTERNRT int rtxCharStrToInt8 (const char ∗cstr, OSINT8 ∗pvalue)

  This function converts the given character string to a signed 8-bit integer value.
- EXTERNRT int rtxCharStrToInt16 (const char ∗cstr, OSINT16 ∗pvalue)

  This function converts the given character string to a signed 16-bit integer value.
- EXTERNRT int rtxCharStrToInt64 (const char ∗cstr, OSINT64 ∗pvalue)

  This function converts the given character string to a signed 64-bit integer value.
- EXTERNRT int rtxCharStrToUInt (const char ∗cstr, OSUINT32 ∗pvalue)

  This function converts the given character string to an unsigned 32-bit integer value.
- EXTERNRT int rtxCharStrToUInt8 (const char ∗cstr, OSUINT8 ∗pvalue)

  This function converts the given character string to an unsigned 8-bit integer value.
- EXTERNRT int rtxCharStrToUInt16 (const char ∗cstr, OSUINT16 ∗pvalue)

  This function converts the given character string to an unsigned 16-bit integer value.
- EXTERNRT int rtxCharStrToUInt64 (const char ∗cstr, OSUINT64 ∗pvalue)

  This function converts the given character string to an unsigned 64-bit integer value.

7.12 rtxCommon.h File Reference

Common runtime constants, data structure definitions, and run-time functions to support various data encoding standards.

```
#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"
```

7.12.1 Detailed Description

Common runtime constants, data structure definitions, and run-time functions to support various data encoding standards.

7.13 rtxContext.h File Reference

Common run-time context definitions.

```
#include "rtxsrc/rtxDList.h"
#include "rtxsrc/rtxStack.h"
```

Classes

- struct OSRTErrLocn

  Run-time error location structure.
- struct OSRTErrInfo

  Run-time error information structure.
- struct OSRTBuffer

  Run-time message buffer structure.
- struct OSRTBufSave

  Structure to save the current message buffer state.
- struct OSBufferIndex

  This structure can be used as an index into the buffer.
- struct OSCTXT

  Run-time context structure.

Macros

- #define OSNOWHITESPACE 0x00400000 /∗ Turn off indentation whitesapce ∗/

  Turn off unnecessary whitespace in text output.
- #define rtxCtxtGetMsgPtr(pctxt) (pctxt)->buffer.data

  This macro returns the start address of an encoded message.
- #define rtxCtxtGetMsgLen(pctxt) (pctxt)->buffer.byteIndex

  This macro returns the length of an encoded message.
- #define rtxCtxtTestFlag(pctxt, mask) (((pctxt)->flags & mask) != 0)

  This macro tests if the given bit flag is set in the context.
- #define rtxCtxtPeekElemName(pctxt)

  This macro returns the last element name from the context stack.
- #define rtxByteAlign(pctxt)

  This macro will byte-align the context buffer.
- #define rtxCtxtSetProtocolVersion(pctxt, value) (pctxt)->version = value

  This macro sets the protocol version in the context.

Typedefs

- 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.
- 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.
- typedef void(∗ OSFreeCtxtGlobalPtr) (struct OSCTXT ∗pctxt)

  OSRTFreeCtxtGlobalPtr is a pointer to a memory free function.

Functions

- EXTERNRT int rtxInitContext (OSCTXT ∗pctxt)

  This function initializes an OSCTXT block.
- EXTERNRT int rtxInitContextExt (OSCTXT ∗pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, O↩ SFreeFunc free_func)

  This function initializes an OSCTXT block.
- EXTERNRT int rtxInitThreadContext (OSCTXT ∗pctxt, const OSCTXT ∗pSrcCtxt)

  This function initializes a context for use in a thread.
- EXTERNRT int rtxInitContextUsingKey (OSCTXT ∗pctxt, const OSOCTET ∗key, OSSIZE keylen)

  This function initializes a context using a run-time key.
- EXTERNRT int rtxInitContextBuffer (OSCTXT ∗pctxt, OSOCTET ∗bufaddr, OSSIZE bufsiz)

  This function assigns a message buffer to a context block.
- EXTERNRT int rtxCtxtSetBufPtr (OSCTXT ∗pctxt, OSOCTET ∗bufaddr, OSSIZE bufsiz)

  This function is used to set the internal buffer pointer for in-memory encoding or decoding.
- EXTERNRT OSSIZE rtxCtxtGetBitOffset (OSCTXT ∗pctxt)

  This function returns the total bit offset to the current element in the context buffer.
- EXTERNRT int rtxCtxtSetBitOffset (OSCTXT ∗pctxt, OSSIZE offset)

This function sets the bit offset in the context to the given value.

- EXTERNRT OSSIZE rtxCtxtGetIOByteCount (OSCTXT ∗pctxt)

  This function returns the count of bytes either written to a stream or memory buffer.
- EXTERNRT int rtxCheckContext (OSCTXT ∗pctxt)

  This function verifies that the given context structure is initialized and ready for use.
- EXTERNRT void rtxFreeContext (OSCTXT ∗pctxt)

  This function frees all dynamic memory associated with a context.
- EXTERNRT void rtxCopyContext (OSCTXT ∗pdest, OSCTXT ∗psrc)

  This function creates a copy of a context structure.
- EXTERNRT void rtxCtxtSetFlag (OSCTXT ∗pctxt, OSUINT32 mask)

  This function is used to set a processing flag within the context structure.
- EXTERNRT void rtxCtxtClearFlag (OSCTXT ∗pctxt, OSUINT32 mask)

  This function is used to clear a processing flag within the context structure.
- EXTERNRT 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.
- EXTERNRT int rtxCtxtPushElemName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗elemName)

  This function is used to push an element name onto the context element name stack.
- EXTERNRT int rtxCtxtPushElemNameCopy (OSCTXT ∗pctxt, const OSUTF8CHAR ∗elemName)

  This function is used to push a copy of the given element name onto the context element name stack.
- EXTERNRT int rtxCtxtPushTypeName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗typeName)

  This function is used to push a type name onto the context element name stack.
- EXTERNRT OSBOOL rtxCtxtPopArrayElemName (OSCTXT ∗pctxt)

  This function pops the last element name from the context stack.
- EXTERNRT const OSUTF8CHAR ∗ rtxCtxtPopElemName (OSCTXT ∗pctxt)

  This function pops the last element name from the context stack.
- EXTERNRT void rtxCtxtPopElemNameCopy (OSCTXT ∗pctxt)

  This function pops the last element name from the context stack and frees the associated memory.
- EXTERNRT const OSUTF8CHAR ∗ rtxCtxtPopTypeName (OSCTXT ∗pctxt)

  This function pops the type name from the context stack.
- EXTERNRT OSBOOL rtxCtxtContainerHasRemBits (OSCTXT ∗pctxt)

  Return true iff there are bits remaining to be decoded in the current length-constrained container, which is possibly the outer PDU.
- EXTERNRT OSBOOL rtxCtxtContainerEnd (OSCTXT ∗pctxt)

  Return true if we are at the end of container - neither having more bits remaining nor having overrun it; otherwise return false.
- EXTERNRT OSSIZE rtxCtxtGetContainerRemBits (OSCTXT ∗pctxt)

  Return the number of bits remaining to be decoded in the current length-constrained container, which is possibly the outer PDU.
- EXTERNRT int rtxCtxtPushContainerBytes (OSCTXT ∗pctxt, OSSIZE bytes)

  Notify the runtime layer of the start of decoding of a length-constrained container of a given length.
- EXTERNRT int rtxCtxtPushContainerBits (OSCTXT ∗pctxt, OSSIZE bits)

  Notify the runtime layer of the start of decoding of a length-constrained container of a given length.
- EXTERNRT void rtxCtxtPopContainer (OSCTXT ∗pctxt)

  Notify the runtime layer of the end of decoding of a length-constrained container of the given length.
- EXTERNRT void rtxCtxtPopAllContainers (OSCTXT ∗pctxt)

  Pop all containers from the container stack.
- EXTERNRT void rtxMemHeapSetFlags (OSCTXT ∗pctxt, OSUINT32 flags)

  This function sets flags to a heap.

- EXTERNRT void rtxMemHeapClearFlags (OSCTXT ∗pctxt, OSUINT32 flags)

  This function clears memory heap flags.
- EXTERNRT int rtxCtxtMarkBitPos (OSCTXT ∗pctxt, OSSIZE ∗ppos)

  This function saves the current bit position in a message buffer.
- EXTERNRT int rtxCtxtResetToBitPos (OSCTXT ∗pctxt, OSSIZE pos)

  This function resets a message buffer back to the given bit position.
- EXTERNRT int rtxMarkPos (OSCTXT ∗pctxt, OSSIZE ∗ppos)

  This function saves the current position in a message buffer or stream.
- EXTERNRT int rtxResetToPos (OSCTXT ∗pctxt, OSSIZE pos)

  This function resets a message buffer or stream back to the given position.
- EXTERNRT const char ∗ rtxCtxtGetExpDateStr (OSCTXT ∗pctxt, char ∗buf, OSSIZE bufsiz)

  This function will get the license expiration date for a time-limited license.
- EXTERNRT void rtxLicenseClose (void)

  Finish with current license and free internal resources.

7.13.1 Detailed Description

Common run-time context definitions.

7.14 rtxCtype.h File Reference

7.15 rtxDateTime.h File Reference

Common runtime functions for converting to and from various standard date/time formats.

```
#include <time.h>
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT 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).
- EXTERNRT 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]).
- EXTERNRT 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]).
- EXTERNRT int rtxGYearToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian year value to a string (CCYY).
- EXTERNRT int rtxGYearMonthToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian year and month value to a string (CCYY-MM).

- EXTERNRT int rtxGMonthToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian month value to a string (MM).
- EXTERNRT int rtxGMonthDayToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian month and day value to a string (MM-DD).
- EXTERNRT int rtxGDayToString (const OSNumDateTime ∗pvalue, OSUTF8CHAR ∗buffer, size_t bufsize)

  This function formats a gregorian day value to a string (DD).
- EXTERNRT int rtxGetCurrDateTime (OSNumDateTime ∗pvalue)

  This function reads the system date and time and stores the value in the given OSNumDateTime structure variable.
- EXTERNRT int rtxGetCurrDateTimeString (char ∗buffer, OSSIZE bufsize, OSBOOL local)

  This function reads the current system date and time and returns it as a formatted string.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxCmpDateTime (const OSNumDateTime ∗pvalue1, const OSNumDateTime ∗pvalue2)

  This function compares two OSNumDateTime structures and returns the result of the comparison.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT 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 OSNumDate←↩ Time to the decoded value.
- EXTERNRT 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.

- EXTERNRT 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.
- EXTERNRT int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR ∗buf, OSUINT32 bufsize)

  This function converts millisecs to a duration string with format "PnYnMnDTnHnMnS".
- EXTERNRT int rtxDurationToMSecs (OSUTF8CHAR ∗buf, OSUINT32 bufsize, OSINT32 ∗msecs)

  This function converts a duration string to milliseconds.
- EXTERNRT int rtxSetDateTime (OSNumDateTime ∗pvalue, struct tm ∗timeStruct)

  This function converts a structure of type 'struct tm' to an OSNumDateTime structure.
- EXTERNRT int rtxGetGMTime (struct tm ∗pvalue, time_t timeMs)

  This function gets GM time.
- EXTERNRT int rtxGetLocalTime (struct tm ∗pvalue, time_t timeMs)

  This function gets local time.
- EXTERNRT int rtxSetLocalDateTime (OSNumDateTime ∗pvalue, time_t timeMs)

  This function converts a local date and time value to an OSNumDateTime structure.
- EXTERNRT int rtxSetUtcDateTime (OSNumDateTime ∗pvalue, time_t timeMs)

  This function converts a UTC date and time value to an OSNumDateTime structure.
- EXTERNRT 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.
- EXTERNRT OSBOOL rtxDateIsValid (const OSNumDateTime ∗pvalue)

  This function verifies that date members (year, month, day, timezone) of the OSNumDateTime structure contains valid values.
- EXTERNRT OSBOOL rtxTimeIsValid (const OSNumDateTime ∗pvalue)

  This function verifies that time members (hour, minute, second, timezone) of the OSNumDateTime structure contains valid values.
- EXTERNRT OSBOOL rtxDateTimeIsValid (const OSNumDateTime ∗pvalue)

  This function verifies that all members of the OSNumDateTime structure contains valid values.
- EXTERNRT int rtxAscTime (char ∗buffer, OSSIZE bufsize, struct tm ∗pvalue)

  This function returns a string representation of the given date/time structure.

7.15.1 Detailed Description

Common runtime functions for converting to and from various standard date/time formats.

7.16 rtxDecimal.h File Reference

Common runtime functions for working with xsd:decimal numbers.

```
#include "rtxsrc/rtxContext.h"
```

7.16.1 Detailed Description

Common runtime functions for working with xsd:decimal numbers.

7.17 rtxDiag.h File Reference

Common runtime functions for diagnostic tracing and debugging.

```
#include <stdarg.h>
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT OSBOOL rtxDiagEnabled (OSCTXT ∗pctxt)

  This function is used to determine if diagnostic tracing is currently enabled for the specified context.
- EXTERNRT 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.
- EXTERNRT OSBOOL rtxSetGlobalDiag (OSBOOL value)

  This function is used to turn diagnostic tracing on or off at run-time on a global basis.
- EXTERNRT void rtxDiagPrint (OSCTXT ∗pctxt, const char ∗fmtspec,...)

  This function is used to print a diagnostics message to `stdout`*.*
- EXTERNRT void rtxDiagStream (OSCTXT ∗pctxt, const char ∗fmtspec,...)

  This function conditionally outputs diagnostic trace messages to an output stream defined within the context.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT void rtxDiagPrintChars (OSCTXT ∗pctxt, const char ∗data, size_t nchars)

  This function is used to print a given number of characters to standard output.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT void rtxDiagSetTraceLevel (OSCTXT ∗pctxt, OSRTDiagTraceLevel level)

  This function is used to set the maximum trace level for diagnostic trace messages.
- EXTERNRT OSBOOL rtxDiagTraceLevelEnabled (OSCTXT ∗pctxt, OSRTDiagTraceLevel level)

  This function tests if a given trace level is enabled.

7.17.1 Detailed Description

Common runtime functions for diagnostic tracing and debugging.

7.18 rtxDiagBitTrace.h File Reference

Common runtime functions for tracing bit patterns written to or read from a stream.

```
#include <stdarg.h>
#include "rtxsrc/rtxMemBuf.h"
#include "rtxsrc/rtxSList.h"
#include "rtxsrc/rtxPrintToStream.h"
```

Macros

- #define RTDIAG_GETCTXTBITOFFSET(pctxt) (((pctxt)->buffer.byteIndex ∗ 8) + (8 - (pctxt)->buffer.bitOffset))

  This macro calculates the relative bit offset to the current buffer position.

Functions

- EXTERNRT int rtxDiagCtxtBitFieldListInit (OSCTXT ∗pctxt)

  This function initializes the standard bit field list structure within the context.
- EXTERNRT void rtxDiagBitFieldListInit (OSCTXT ∗pctxt, OSRTDiagBitFieldList ∗pBFList)

  This function initializes a bit field list structure.
- EXTERNRT void rtxDiagInsBitFieldLen (OSRTDiagBitFieldList ∗pBFList)

  This function inserts a special length field before the current record in the bit field list.
- EXTERNRT OSRTDiagBitField ∗ rtxDiagNewBitField (OSRTDiagBitFieldList ∗pBFList, const char ∗nameSuffix)

  This function allocates a new bit field structure and adds it to the bit field list.
- EXTERNRT void rtxDiagSetBitFldOffset (OSRTDiagBitFieldList ∗pBFList)

  This function is used to set the bit offset in the current bit field structure.
- EXTERNRT void rtxDiagSetBitFldCount (OSRTDiagBitFieldList ∗pBFList)

  This function is used to set the bit count in the current bit field structure.
- EXTERNRT void rtxDiagSetBitFldNameSuffix (OSRTDiagBitFieldList ∗pBFList, const char ∗nameSuffix)

  This function is used to set the name suffix in the current bit field structure.
- EXTERNRT OSBOOL rtxDiagSetBitFldDisabled (OSRTDiagBitFieldList ∗pBFList, OSBOOL value)

  This function increments or decrements the disabled count.
- EXTERNRT void rtxDiagBitTracePrint (OSRTDiagBitFieldList ∗pBFList, const char ∗varname)

  This function prints the bit field list to a an output stream.
- EXTERNRT void rtxDiagBitTracePrintHTML (const char ∗filename, OSRTDiagBitFieldList ∗pBFList, const char ∗varname)

  This function prints the bit field list to a an HTML document.
- EXTERNRT void rtxDiagBitFldAppendNamePart (OSRTDiagBitFieldList ∗pBFList, const char ∗namePart)

  This function appends the given name part to the element name in the bit field.

7.18.1 Detailed Description

Common runtime functions for tracing bit patterns written to or read from a stream.

7.18.2 Function Documentation

7.18.2.1 rtxDiagBitFieldListInit()

```
EXTERNRT void rtxDiagBitFieldListInit (
             OSCTXT * pctxt,
             OSRTDiagBitFieldList * pBFList )
```

This function initializes a bit field list structure.

Parameters

pctxt	Pointer to a context structure.
pBFList	Pointer to bit field list structure.

7.18.2.2 rtxDiagBitFldAppendNamePart()

```
EXTERNRT void rtxDiagBitFldAppendNamePart (
             OSRTDiagBitFieldList * pBFList,
             const char * namePart )
```

This function appends the given name part to the element name in the bit field.

A dot (.) separator character is added after the existing name and before the name part.

Parameters

pBFList	Pointer to bit field list structure.
namePart	A name part that is appended to the field.

7.18.2.3 rtxDiagBitTracePrint()

```
EXTERNRT void rtxDiagBitTracePrint (
             OSRTDiagBitFieldList * pBFList,
             const char * varname )
```

This function prints the bit field list to a an output stream.

By default, the output goes to stdout; but this can be changed by creating a print output stream within the context (see rtxPrintStream).

Parameters

pBFList	Pointer to bit field list structure.
varname	A variable name that is prepended to each field.

7.18.2.4 rtxDiagBitTracePrintHTML()

```
EXTERNRT void rtxDiagBitTracePrintHTML (
             const char * filename,
             OSRTDiagBitFieldList * pBFList,
             const char * varname )
```

This function prints the bit field list to a an HTML document.

Parameters

filename	Name of HTML file to be written.
pBFList	Pointer to bit field list structure.
varname	A variable name that is prepended to each field.

7.18.2.5 rtxDiagCtxtBitFieldListInit()

```
EXTERNRT int rtxDiagCtxtBitFieldListInit (
             OSCTXT * pctxt )
```

This function initializes the standard bit field list structure within the context.

Parameters

pctxt	Pointer to a context structure.

7.18.2.6 rtxDiagInsBitFieldLen()

```
EXTERNRT void rtxDiagInsBitFieldLen (
             OSRTDiagBitFieldList * pBFList )
```

This function inserts a special length field before the current record in the bit field list.

Parameters

pBFList	Pointer to bit field list structure.

7.18.2.7 rtxDiagNewBitField()

```
EXTERNRT OSRTDiagBitField* rtxDiagNewBitField (
            OSRTDiagBitFieldList * pBFList,
            const char * nameSuffix )
```

This function allocates a new bit field structure and adds it to the bit field list.

Parameters

pBFList	Pointer to bit field list structure.
nameSuffix	Suffix to append to the bit field name.

Returns

Allocated bit field structure.

7.18.2.8 rtxDiagSetBitFldCount()

```
EXTERNRT void rtxDiagSetBitFldCount (
            OSRTDiagBitFieldList * pBFList )
```

This function is used to set the bit count in the current bit field structure.

Parameters

pBFList	Pointer to bit field list structure.

7.18.2.9 rtxDiagSetBitFldDisabled()

```
EXTERNRT OSBOOL rtxDiagSetBitFldDisabled (
            OSRTDiagBitFieldList * pBFList,
            OSBOOL value )
```

This function increments or decrements the disabled count.

This allows the list to be temporaily disabled to allow collection of more bits to form larger, aggregate fields.

Parameters

pBFList	Pointer to bit field list structure.
value	Indicates if disabled count should be incremented (TRUE) or decremented (FALSE).

Returns

TRUE if field operations are still disabled.

7.18.2.10 rtxDiagSetBitFldNameSuffix()

```
EXTERNRT void rtxDiagSetBitFldNameSuffix (
             OSRTDiagBitFieldList * pBFList,
             const char * nameSuffix )
```

This function is used to set the name suffix in the current bit field structure.

This text is printed after the element name when the field is displayed.

Parameters

pBFList	Pointer to bit field list structure.
nameSuffix	Suffix to append to the bit field name.

7.18.2.11 rtxDiagSetBitFldOffset()

```
EXTERNRT void rtxDiagSetBitFldOffset (
             OSRTDiagBitFieldList * pBFList )
```

This function is used to set the bit offset in the current bit field structure.

Parameters

pBFList	Pointer to bit field list structure.

7.19 rtxDList.h File Reference

Doubly-Linked List Utility Functions.

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxCommonDefs.h"
```

Classes

- struct OSRTDListNode

  This structure is used to hold a single data item within the list.
- struct OSRTDList

  This is the main list structure.

Functions

- EXTERNRT void rtxDListInit (OSRTDList ∗pList)

  This function initializes a doubly linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListAppend (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pData)

  This function appends an item to the linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListAppendCharArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, size_t length, char ∗pData)

  This function appends an item to the linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListAppendNode (OSRTDList ∗pList, OSRTDListNode ∗pListNode)

  This function appends an `OSRTDListNode` to the linked list structure.
- EXTERNRT OSRTDListNode ∗ rtxDListInsert (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSSIZE idx, void ∗pData)

  This function inserts an item into the linked list structure.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT OSRTDListNode ∗ rtxDListFindByIndex (const OSRTDList ∗pList, OSSIZE idx)

  This function will return the node pointer of the indexed entry in the list.
- EXTERNRT 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.
- EXTERNRT OSRTDListNode ∗ rtxDListFindFirstData (const OSRTDList ∗pList)

  This function will return the node pointer of the first non-null data item within the list or NULL if there is no node that has non-null data.
- EXTERNRT 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.
- EXTERNRT void rtxDListFreeNode (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node)

  This function will remove the given node from the list and free memory.
- EXTERNRT void rtxDListRemove (OSRTDList ∗pList, OSRTDListNode ∗node)

  This function will remove the given node from the list.
- EXTERNRT void rtxDListFreeNodes (struct OSCTXT ∗pctxt, OSRTDList ∗pList)

  This function will free all of the dynamic memory used to hold the list node pointers.

- EXTERNRT 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.
- EXTERNRT int rtxDListToArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗∗ppArray, OSSIZE ∗pElem←↩ Count, OSSIZE elemSize)

  This function converts a doubly linked list of ∗T to a dynamically (or pre-allocated) array of T.
- EXTERNRT int rtxDListToPointerArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗∗∗ppArray, OSSIZE ∗p←↩ ElemCount)

  This function converts a doubly linked list of ∗T to a dynamically (or pre-allocated) array of ∗T.
- EXTERNRT int rtxDListAppendArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pArray, OSSIZE num←↩ Elements, OSSIZE elemSize)

  This function appends pointers to items in the given array to a doubly linked list structure.
- EXTERNRT int rtxDListAppendArrayCopy (struct OSCTXT ∗pctxt, OSRTDList ∗pList, const void ∗pArray, OSS←↩ IZE numElements, OSSIZE elemSize)

  This function appends a copy of each item in the given array to a doubly linked list structure.
- EXTERNRT 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.

7.19.1 Detailed Description

Doubly-Linked List Utility Functions.

7.20 rtxDynBitSet.h File Reference

Implementation of a dynamic bit set similar to the Java BitSet class.

```
#include "rtxsrc/rtxBitString.h"
```

Functions

- EXTERNRT int rtxDynBitSetInit (OSCTXT ∗pctxt, OSRTDynBitSet ∗pbitset, OSUINT16 segNBytes)

  This function initializes a dynamic bit set structure.
- EXTERNRT void rtxDynBitSetFree (OSCTXT ∗pctxt, OSRTDynBitSet ∗pbitset)

  This function frees dynamic memory held by the bit set.
- EXTERNRT int rtxDynBitSetCopy (OSCTXT ∗pctxt, const OSRTDynBitSet ∗pSrcBitSet, OSRTDynBitSet ∗p←↩ DestBitSet)

  This function creates a deep copy of the given bit set.
- EXTERNRT int rtxDynBitSetSetBit (OSCTXT ∗pctxt, OSRTDynBitSet ∗pbitset, OSUINT32 idx)

  This function sets the bit at the given index.
- EXTERNRT int rtxDynBitSetClearBit (OSRTDynBitSet ∗pbitset, OSUINT32 idx)

  This function clears the bit at the given index.
- EXTERNRT OSBOOL rtxDynBitSetTestBit (const OSRTDynBitSet ∗pbitset, OSUINT32 idx)

  This function tests the bit at the given index.
- EXTERNRT int rtxDynBitSetSetBitToValue (OSCTXT ∗pctxt, OSRTDynBitSet ∗pbitset, OSUINT32 idx, OSBOOL value)

  This function sets the bit at the given index to the give value.
- EXTERNRT int rtxDynBitSetInsertBit (OSCTXT ∗pctxt, OSRTDynBitSet ∗pbitset, OSUINT32 idx, OSBOOL value)

  This function inserts a bit with the given value at the given index.

7.20.1 Detailed Description

Implementation of a dynamic bit set similar to the Java BitSet class.

7.20.2 Function Documentation

7.20.2.1 rtxDynBitSetClearBit()

```
EXTERNRT int rtxDynBitSetClearBit (
             OSRTDynBitSet * pbitset,
             OSUINT32 idx )
```

This function clears the bit at the given index.

The bit set will be not be expanded if the given index is outside the currently allocated range. The bit will be assumed to already be clear since it is undefined.

Parameters

pbitset	Pointer to bit set structure.
idx	Index of bit to be clear.

Returns

Status of operation: zero if success or a negative error code if failure.

7.20.2.2 rtxDynBitSetCopy()

```
EXTERNRT int rtxDynBitSetCopy (
             OSCTXT * pctxt,
             const OSRTDynBitSet * pSrcBitSet,
             OSRTDynBitSet * pDestBitSet )
```

This function creates a deep copy of the given bit set.

Parameters

pctxt	Pointer to a context structure.
pSrcBitSet	Pointer to bit set structure to be copied.
pDestBitSet	Pointer to bit set structure to recieve copied data.

Returns

Status of operation: zero if success or a negative error code if failure.

7.20.2.3 rtxDynBitSetFree()

```
EXTERNRT void rtxDynBitSetFree (
            OSCTXT * pctxt,
            OSRTDynBitSet * pbitset )
```

This function frees dynamic memory held by the bit set.

Parameters

pctxt	Pointer to a context structure.
pbitset	Pointer to bit set structure to be freed.

7.20.2.4 rtxDynBitSetInit()

```
EXTERNRT int rtxDynBitSetInit (
            OSCTXT * pctxt,
            OSRTDynBitSet * pbitset,
            OSUINT16 segNBytes )
```

This function initializes a dynamic bit set structure.

Memory is allocated for the initial segment.

Parameters

pctxt	Pointer to a context structure.
pbitset	Pointer to bit set structure to be initialized.
segNBytes	Number of bytes per segment expansion. If zero, the default value is used.

Returns

Status of operation: zero if success or a negative error code if failure.

7.20.2.5 rtxDynBitSetInsertBit()

```
EXTERNRT int rtxDynBitSetInsertBit (
             OSCTXT * pctxt,
             OSRTDynBitSet * pbitset,
             OSUINT32 idx,
             OSBOOL value )
```

This function inserts a bit with the given value at the given index.

All other bits are shifted to the right one position. If the maximum set bit number is at the end of the allocated range, the set is expanded.

Parameters

pctxt	Pointer to a context structure.
pbitset	Pointer to bit set structure.
idx	Index of position where bit is to be inserted.
value	Boolean value of the bit.

Returns

Status of operation: zero if success or a negative error code if failure.

7.20.2.6 rtxDynBitSetSetBit()

```
EXTERNRT int rtxDynBitSetSetBit (
             OSCTXT * pctxt,
             OSRTDynBitSet * pbitset,
             OSUINT32 idx )
```

This function sets the bit at the given index.

The bit set will be expanded if the given index is outside the currently allocated range.

Parameters

pctxt	Pointer to a context structure.
pbitset	Pointer to bit set structure.
idx	Index of bit to be set.

Returns

Status of operation: zero if success or a negative error code if failure.

7.20.2.7 rtxDynBitSetSetBitToValue()

```
EXTERNRT int rtxDynBitSetSetBitToValue (
            OSCTXT * pctxt,
            OSRTDynBitSet * pbitset,
            OSUINT32 idx,
            OSBOOL value )
```

This function sets the bit at the given index to the give value.

The bit set will be expanded if the given index is outside the currently allocated range.

Parameters

pctxt	Pointer to a context structure.
pbitset	Pointer to bit set structure.
idx	Index of bit to be set.
value	Boolean value to which bit is to be set.

Returns

Status of operation: zero if success or a negative error code if failure.

7.20.2.8 rtxDynBitSetTestBit()

```
EXTERNRT OSBOOL rtxDynBitSetTestBit (
            const OSRTDynBitSet * pbitset,
            OSUINT32 idx )
```

This function tests the bit at the given index.

If the index is outside the range of the currently allocated set, the bit is assumed to be clear; otherwise, the state of the bit in the set is tested.

Parameters

pbitset	Pointer to bit set structure.
idx	Index of bit to be tested.

Returns

Boolean result: true if bit set; false if clear.

7.21 rtxDynPtrArray.h File Reference

Implementation of a dynamic pointer array.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxDynPtrArrayInit (OSCTXT ∗pctxt, OSRTDynPtrArray ∗pArray, OSUINT16 initialSize)

  This function initializes a new dynamic pointer array structure.
- EXTERNRT int rtxDynPtrArrayAppend (OSCTXT ∗pctxt, OSRTDynPtrArray ∗pArray, void ∗ptr)

  This function adds a pointer to the end of the array and expands the array if necessary.

7.21.1 Detailed Description

Implementation of a dynamic pointer array.

7.21.2 Function Documentation

7.21.2.1 rtxDynPtrArrayAppend()

```
EXTERNRT int rtxDynPtrArrayAppend (
            OSCTXT * pctxt,
            OSRTDynPtrArray * pArray,
            void * ptr )
```

This function adds a pointer to the end of the array and expands the array if necessary.

Parameters

pctxt	Pointer to a context structure.
pArray	Pointer to dynamic pointer array structure.
ptr	Pointer to be added to the array.

Returns

Status of operation: zero if success or a negative error code if failure.

7.21.2.2 rtxDynPtrArrayInit()

```
EXTERNRT int rtxDynPtrArrayInit (
             OSCTXT * pctxt,
             OSRTDynPtrArray * pArray,
             OSUINT16 initialSize )
```

This function initializes a new dynamic pointer array structure.

Memory is allocated for the initial capacity of pointers.

Parameters

pctxt	Pointer to a context structure.
pArray	Pointer to dynamic pointer array structure.
initialSize	Initial capacilty of the array. The size will doubled on each expansion. If zero is provided, a default size will be used.

Returns

Status of operation: zero if success or a negative error code if failure.

7.22 rtxEnum.h File Reference

Common runtime types and functions for performing operations on enumerated data items.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT OSINT32 rtxLookupEnum (const OSUTF8CHAR *strValue, size_t strValueSize, const OSEnumItem enumTable[ ], OSUINT16 enumTableSize)

  This function will return the numeric value for the given enumerated identifier string.
- EXTERNRT OSINT32 rtxLookupEnumU32 (const OSUTF8CHAR *strValue, size_t strValueSize, const OSEnumItemU32 enumTable[ ], OSUINT16 enumTableSize)

  This function will return the numeric value for the given enumerated identifier string.
- EXTERNRT OSINT32 rtxLookupBigEnum (const OSUTF8CHAR *strValue, size_t strValueSize, const OSBigEnumItem enumTable[ ], OSUINT16 enumTableSize)

  This function will return the numeric value for the given enumerated identifier string.
- EXTERNRT OSINT32 rtxLookupEnumByValue (OSINT32 value, const OSEnumItem enumTable[ ], size_t enumTableSize)

  Lookup enum by integer value.
- EXTERNRT OSINT32 rtxLookupEnumU32ByValue (OSUINT32 value, const OSEnumItemU32 enumTable[ ], size_t enumTableSize)

  Lookup enum by integer value (Unsiged 32-bit integer).

- EXTERNRT OSINT32 rtxLookupBigEnumByValue (const char ∗value, const OSBigEnumItem enumTable[ ], size_t enumTableSize)

  Lookup enum by stringified version of value.
- EXTERNRT int rtxTestNumericEnum (OSINT32 ivalue, const OSNumericEnumItem enumTable[ ], OSUINT16 enumTableSize)

  This function determines if the given numeric enumerated value is within the defined numeration set.

7.22.1 Detailed Description

Common runtime types and functions for performing operations on enumerated data items.

7.23 rtxErrCodes.h File Reference

List of numeric status codes that can be returned by common run-time functions and generated code.

Macros

- #define RT_OK 0

  Normal completion status.
- #define RT_OK_FRAG 2

  Message fragment return status.
- #define RTERR_BUFOVFLW -1

  Encode buffer overflow.
- #define RTERR_ENDOFBUF -2

  Unexpected end-of-buffer.
- #define RTERR_IDNOTFOU -3

  Expected identifier not found.
- #define RTERR_INVENUM -4

  Invalid enumerated identifier.
- #define RTERR_SETDUPL -5

  Duplicate element in set.
- #define RTERR_SETMISRQ -6

  Missing required element in set.
- #define RTERR_NOTINSET -7

  Element not in set.
- #define RTERR_SEQOVFLW -8

  Sequence overflow.
- #define RTERR_INVOPT -9

  Invalid option in choice.
- #define RTERR_NOMEM -10

  No dynamic memory available.
- #define RTERR_INVHEXS -11

  Invalid hexadecimal string.
- #define RTERR_INVREAL -12

Invalid real number value.

- #define RTERR_STROVFLW -13

  String overflow.
- #define RTERR_BADVALUE -14

  Bad value.
- #define RTERR_TOODEEP -15

  Nesting level too deep.
- #define RTERR_CONSVIO -16

  Constraint violation.
- #define RTERR_ENDOFFILE -17

  Unexpected end-of-file error.
- #define RTERR_INVUTF8 -18

  Invalid UTF-8 character encoding.
- #define RTERR_OUTOFBND -19

  Array index out-of-bounds.
- #define RTERR_INVPARAM -20

  Invalid parameter passed to a function or method.
- #define RTERR_INVFORMAT -21

  Invalid value format.
- #define RTERR_NOTINIT -22

  Context not initialized.
- #define RTERR_TOOBIG -23

  Value will not fit in target variable.
- #define RTERR_INVCHAR -24

  Invalid character.
- #define RTERR_XMLSTATE -25

  XML state error.
- #define RTERR_XMLPARSE -26

  XML parser error.
- #define RTERR_SEQORDER -27

  Sequence order error.
- #define RTERR_FILNOTFOU -28

  File not found.
- #define RTERR_READERR -29

  Read error.
- #define RTERR_WRITEERR -30

  Write error.
- #define RTERR_INVBASE64 -31

  Invalid Base64 encoding.
- #define RTERR_INVSOCKET -32

  Invalid socket.
- #define RTERR_INVATTR -33

  Invalid attribute.
- #define RTERR_REGEXP -34

  Invalid regular expression.
- #define RTERR_PATMATCH -35

  Pattern match error.

- #define RTERR_ATTRMISRQ -36

  Missing required attribute.
- #define RTERR_HOSTNOTFOU -37

  Host name could not be resolved.
- #define RTERR_HTTPERR -38

  HTTP protocol error.
- #define RTERR_SOAPERR -39

  SOAP error.
- #define RTERR_EXPIRED -40

  Evaluation license expired.
- #define RTERR_UNEXPELEM -41

  Unexpected element encountered.
- #define RTERR_INVOCCUR -42

  Invalid number of occurrences.
- #define RTERR_INVMSGBUF -43

  Invalid message buffer has been passed to decode or validate method.
- #define RTERR_DECELEMFAIL -44

  Element decode failed.
- #define RTERR_DECATTRFAIL -45

  Attribute decode failed.
- #define RTERR_STRMINUSE -46

  Stream in-use.
- #define RTERR_NULLPTR -47

  Null pointer.
- #define RTERR_FAILED -48

  General failure.
- #define RTERR_ATTRFIXEDVAL -49

  Attribute fixed value mismatch.
- #define RTERR_MULTIPLE -50

  Multiple errors occurred during an encode or decode operation.
- #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.
- #define RTERR_ADDRINUSE -52

  Address already in use.
- #define RTERR_CONNRESET -53

  Remote connection was reset.
- #define RTERR_UNREACHABLE -54

  Network failure.
- #define RTERR_NOCONN -55

  Not connected.
- #define RTERR_CONNREFUSED -56

  Connection refused.
- #define RTERR_INVSOCKOPT -57

  Invalid option.
- #define RTERR_SOAPFAULT -58

  This error is returned when decoded SOAP envelope is fault message.

- #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.
- #define RTERR_NOTSUPP -60 /∗ feature is not supported ∗/

  Feature is not supported.
- #define RTERR_UNBAL -61

  Unbalanced structure.
- #define RTERR_EXPNAME -62

  Expected name.
- #define RTERR_UNICODE -63

  Invalid Unicode sequence.
- #define RTERR_INVBOOL -64

  Invalid boolean keyword.
- #define RTERR_INVNULL -65

  Invalid null keyword.
- #define RTERR_INVLEN -66

  Invalid length.
- #define RTERR_UNKNOWNIE -67

  Unknown information element.
- #define RTERR_NOTALIGNED -68

  Not aligned error.
- #define RTERR_EXTRDATA -69

  Extraneous data.
- #define RTERR_INVMAC -70

  Invalid Message Authentication Code.
- #define RTERR_NOSECPARAMS -71

  No security parameters provided.
- #define RTERR_COPYFAIL -72

  Copy failed.
- #define RTERR_PARSEFAIL -73

  Parse failed.
- #define RTERR_VALCMPERR -74

  Value comparison error.
- #define RTERR_BUFCMPERR -75

  Buffer comparison error.
- #define RTERR_INVBITS -76

  Invalid bit string error.
- #define RTERR_RLM -77

  RLM error encounterd.
- #define RTERR_NOCODEC -78

  No codec.
- #define RTERR_WOULDBLOCK -79

  A non-blocking socket could not return any data without blocking.
- #define RTERR_NODATA -80

  CDR or message file contains no data records.
- #define RTERR_MBEDTLS -81

  Error returned from MBEDTLS function.
- #define RTERR_INTOVFLOW -82

  Integer value overflow.
- #define RTERR_ABORT_REQUESTED -83

  User event handler calls for abort.

7.23.1 Detailed Description

List of numeric status codes that can be returned by common run-time functions and generated code.

7.24 rtxError.h File Reference

Error handling function and macro definitions.

```
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxErrCodes.h"
#include <errno.h>
```

Macros

- #define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat,__FILE__,__LINE__)

  This macro is used to log a run-time error in the context.
- #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }

  This macro is used to check an assertion.
- #define OSRTCHECKPARAM(condition) if (condition) { /∗ do nothing ∗/ }

  This macro check a condition but takes no action.
- #define LOG_RTERR_AND_FREE_MEM(ctxt_p, stat, mem_p) rtxMemFreePtr ((ctxt_p),(mem_p)), LOG_RTE←RR(ctxt_p, stat)

  This logs an error to a global context and frees a memory pointer allocated for encoding or decoding.

Functions

- EXTERNRT OSBOOL rtxErrAddCtxtBufParm (OSCTXT ∗pctxt)

  This function adds the contents of the context buffer to the error information structure in the context.
- EXTERNRT OSBOOL rtxErrAddDoubleParm (OSCTXT ∗pctxt, double errParm)

  This function adds a double parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddErrorTableEntry (const char ∗const ∗ppStatusText, OSINT32 minErrCode, OS←INT32 maxErrCode)

  This function adds a set of error codes to the global error table.
- EXTERNRT OSBOOL rtxErrAddElemNameParm (OSCTXT ∗pctxt)

  This function adds an element name parameter to the context error information structure.
- EXTERNRT OSBOOL rtxErrAddIntParm (OSCTXT ∗pctxt, int errParm)

  This function adds an integer parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddInt64Parm (OSCTXT ∗pctxt, OSINT64 errParm)

  This function adds a 64-bit integer parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddSizeParm (OSCTXT ∗pctxt, OSSIZE errParm)

  This function adds a size-typed parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddStrParm (OSCTXT ∗pctxt, const char ∗pErrParm)

  This function adds a character string parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddStrnParm (OSCTXT ∗pctxt, const char ∗pErrParm, OSSIZE nchars)

This function adds a given number of characters from a character string parameter to an error information structure.

- EXTERNRT OSBOOL rtxErrAddUIntParm (OSCTXT ∗pctxt, unsigned int errParm)

  This function adds an unsigned integer parameter to an error information structure.
- EXTERNRT OSBOOL rtxErrAddUInt64Parm (OSCTXT ∗pctxt, OSUINT64 errParm)

  This function adds an unsigned 64-bit integer parameter to an error information structure.
- EXTERNRT void rtxErrAssertionFailed (const char ∗conditionText, int lineNo, const char ∗fileName)

  This function is used to record an assertion failure.
- EXTERNRT const char ∗ rtxErrFmtMsg (OSRTErrInfo ∗pErrInfo, char ∗bufp, OSSIZE bufsiz)

  This function formats a given error structure from the context into a finished status message including substituted parameters.
- EXTERNRT void rtxErrFreeParms (OSCTXT ∗pctxt)

  This function is used to free dynamic memory that was used in the recording of error parameters.
- EXTERNRT char ∗ rtxErrGetText (OSCTXT ∗pctxt, char ∗pBuf, OSSIZE ∗pBufSize)

  This function returns error text in a memory buffer.
- EXTERNRT char ∗ rtxErrGetTextBuf (OSCTXT ∗pctxt, char ∗pbuf, OSSIZE bufsiz)

  This function returns error text in the given fixed-size memory buffer.
- EXTERNRT char ∗ rtxErrGetMsgText (OSCTXT ∗pctxt)

  This function returns error message text in a memory buffer.
- EXTERNRT char ∗ rtxErrGetMsgTextBuf (OSCTXT ∗pctxt, char ∗pbuf, OSSIZE bufsiz)

  This function returns error message text in a static memory buffer.
- EXTERNRT OSRTErrInfo ∗ rtxErrNewNode (OSCTXT ∗pctxt)

  This function creates a new empty error record for the passed context.
- EXTERNRT void rtxErrInit (OSVOIDARG)

  This function is a one-time initialization function that must be called before any other error processing functions can be called.
- EXTERNRT int rtxErrReset (OSCTXT ∗pctxt)

  This function is used to reset the error state recorded in the context to successful.
- EXTERNRT int rtxErrResetErrInfo (OSCTXT ∗pctxt)

  This function is used to reset the error state recorded in the context to successful.
- EXTERNRT void rtxErrLogUsingCB (OSCTXT ∗pctxt, OSErrCbFunc cb, void ∗cbArg_p)

  This function allows error information to be logged using a user-defined callback routine.
- EXTERNRT void rtxErrPrint (OSCTXT ∗pctxt)

  This function is used to print the error information stored in the context to the standard output device.
- EXTERNRT void rtxErrPrintElement (OSRTErrInfo ∗pErrInfo)

  This function is used to print the error information stored in the error information element to the standard output device.
- EXTERNRT int rtxErrSetData (OSCTXT ∗pctxt, int status, const char ∗module, int lineno)

  This function is used to record an error in the context structure.
- EXTERNRT int rtxErrSetNewData (OSCTXT ∗pctxt, int status, const char ∗module, int lineno)

  This function is used to record an error in the context structure.
- EXTERNRT void rtxErrSetNonFatal (OSCTXT ∗pctxt)

  This marks the last error recorded in the context as non-fatal.
- EXTERNRT int rtxErrCheckNonFatal (OSCTXT ∗pctxt)

  This function checks the context structure for non-fatal errors.
- EXTERNRT int rtxErrGetFirstError (const OSCTXT ∗pctxt)

  This function returns the error code, stored in the first error record.
- EXTERNRT int rtxErrGetLastError (const OSCTXT ∗pctxt)

  This function returns the error code, stored in the last error record.

- EXTERNRT OSSIZE rtxErrGetErrorCnt (const OSCTXT ∗pctxt)

  This function returns the total number of error records, including non-fatal errors.
- EXTERNRT int rtxErrGetStatus (const OSCTXT ∗pctxt)

  This function returns the status value from the context.
- EXTERNRT int rtxErrResetLastErrors (OSCTXT ∗pctxt, OSSIZE errorsToReset)

  This function resets last 'errorsToReset' errors in the context.
- EXTERNRT int rtxErrCopy (OSCTXT ∗pDestCtxt, const OSCTXT ∗pSrcCtxt)

  This function copies error information from one context into another.
- EXTERNRT int rtxErrAppend (OSCTXT ∗pDestCtxt, const OSCTXT ∗pSrcCtxt)

  This function appends error information from one context into another.
- EXTERNRT int rtxErrInvUIntOpt (OSCTXT ∗pctxt, OSUINT32 ident)

  This function create an 'invalid option' error (RTERR_INVOPT) in the context using an unsigned integer parameter.

7.24.1 Detailed Description

Error handling function and macro definitions.

7.25 rtxExternDefs.h File Reference

Common definitions of external function modifiers used to define the scope of functions used in DLL's (Windows only).

7.25.1 Detailed Description

Common definitions of external function modifiers used to define the scope of functions used in DLL's (Windows only).

7.26 rtxFile.h File Reference

Common runtime functions for reading from or writing to files.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT OSBOOL rtxFileExists (const char ∗filePath)

  This function tests if a file exists.
- EXTERNRT OSBOOL rtxFileIsDirectory (const char ∗filePath)

  This function tests if a file is actually a directory.
- EXTERNRT time_t rtxFileLastModified (const char ∗filePath)

  This function returns the last modified time for a given file.
- EXTERNRT int rtxFileOpen (FILE ∗∗ppFile, const char ∗filePath, const char ∗access)

  This function opens a file for read, write, or append access.
- EXTERNRT int rtxFileReadBinary (OSCTXT ∗pctxt, const char ∗filePath, OSOCTET ∗∗ppMsgBuf, size_t ∗p←Length)

  This function reads the entire contents of a binary file into memory.
- EXTERNRT int rtxFileReadBinary2 (OSCTXT ∗pctxt, FILE ∗pFile, OSOCTET ∗∗ppMsgBuf, size_t ∗pLength)

  This function reads the entire contents of a binary file into memory.
- EXTERNRT int rtxFileReadBinToSysMem (OSCTXT ∗pctxt, const char ∗filePath, OSOCTET ∗∗ppMsgBuf, size←_t ∗pLength)

  This function reads the entire contents of a binary file into memory.
- EXTERNRT int rtxFileReadText (OSCTXT ∗pctxt, const char ∗filePath, OSOCTET ∗∗ppMsgBuf, size_t ∗pLength)

  This function reads the entire contents of an ASCII text file into memory.
- EXTERNRT int rtxFileWriteBinary (const char ∗filePath, const OSOCTET ∗pMsgBuf, size_t length)

  This function writes binary data from memory to the given file.
- EXTERNRT int rtxFileWriteText (const char ∗filePath, const char ∗pMsgBuf)

  This function writes text data from memory to the given file.
- EXTERNRT int rtxFileCopyTextFile (const char ∗srcFilePath, const char ∗destFilePath)

  This function copies the contents on one text file to another.

7.26.1 Detailed Description

Common runtime functions for reading from or writing to files.

7.26.2 Function Documentation

7.26.2.1 rtxFileCopyTextFile()

```
EXTERNRT int rtxFileCopyTextFile (
             const char * srcFilePath,
             const char * destFilePath )
```

This function copies the contents on one text file to another.

Parameters

srcFilePath	Complete file path name of file to be copied.
destFilePath	Complete file path name of target file.

Returns

Completion status of operation:

- 0 (ASN_OK) = success,
- RTERR_FILNOTFOU = source file not found
- RTERR_FILEREAD = file read error (see errno)

7.26.2.2 rtxFileExists()

```
EXTERNRT OSBOOL rtxFileExists (
            const char * filePath )
```

This function tests if a file exists.

Parameters

filePath	Complete file path name of file to be tested.

Returns

TRUE if file exists or FALSE if does not exist or some other error occurred.

7.26.2.3 rtxFileIsDirectory()

```
EXTERNRT OSBOOL rtxFileIsDirectory (
            const char * filePath )
```

This function tests if a file is actually a directory.

Parameters

filePath	Complete path name of file to be tested.

Returns

TRUE if file is a directory or FALSE otherwise (including if the file doesn't exist).

7.26.2.4 rtxFileLastModified()

```
EXTERNRT time_t rtxFileLastModified (
            const char * filePath )
```

This function returns the last modified time for a given file.

If the file does not exist or some error occurs, 0 is returned.

7.26.2.5 rtxFileOpen()

```
EXTERNRT int rtxFileOpen (
            FILE ** ppFile,
            const char * filePath,
            const char * access )
```

This function opens a file for read, write, or append access.

It is basically a wrapper for the C run-time fopen function except in the case of Visual Studio, the more secure fopen_s function is used.

Parameters

ppFile	Pointer to FILE variable to receive file pointer.
filePath	Complete file path name of file to be opened.
access	File access string as defined for C fopen.

Returns

Completion status of operation:

- 0 = success or negative status code.

7.26.2.6 rtxFileReadBinary()

```
EXTERNRT int rtxFileReadBinary (
            OSCTXT * pctxt,
            const char * filePath,
            OSOCTET ** ppMsgBuf,
            size_t * pLength )
```

This function reads the entire contents of a binary file into memory.

A memory buffer is allocated for the file contents using the run-time memory management functions.

Parameters

pctxt	Pointer to context block structure.
filePath	Complete file path name of file to read.
ppMsgBuf	Pointer to message buffer to receive allocated memory pointer.
pLength	Pointer to integer to receive length of data read.

Returns

Completion status of operation:

- 0 (ASN_OK) = success,
- RTERR_FILNOTFOU = file not found
- RTERR_FILEREAD = file read error (see errno)

7.26.2.7 rtxFileReadBinary2()

```
EXTERNRT int rtxFileReadBinary2 (
            OSCTXT * pctxt,
            FILE * pFile,
            OSOCTET ** ppMsgBuf,
            size_t * pLength )
```

This function reads the entire contents of a binary file into memory.

A memory buffer is allocated for the file contents using the run-time memory management functions.

Parameters

pctxt	Pointer to context block structure.
pFile	Pointer to the open file.
ppMsgBuf	Pointer to message buffer to receive allocated memory pointer.
pLength	Pointer to integer to receive length of data read.

Returns

Completion status of operation:

- 0 (ASN_OK) = success,

- RTERR_FILNOTFOU = file not found
- RTERR_FILEREAD = file read error (see errno)

7.26.2.8 rtxFileReadBinToSysMem()

```
EXTERNRT int rtxFileReadBinToSysMem (
             OSCTXT * pctxt,
             const char * filePath,
             OSOCTET ** ppMsgBuf,
             size_t * pLength )
```

This function reads the entire contents of a binary file into memory.

A memory buffer is allocated for the file contents using the standard configured system memory allocation function (usually malloc).

Parameters

pctxt	Pointer to context block structure.
filePath	Complete file path name of file to read.
ppMsgBuf	Pointer to message buffer to receive allocated memory pointer.
pLength	Pointer to integer to receive length of data read.

Returns

Completion status of operation:

- 0 (ASN_OK) = success,
- RTERR_FILNOTFOU = file not found
- RTERR_FILEREAD = file read error (see errno)

7.26.2.9 rtxFileReadText()

```
EXTERNRT int rtxFileReadText (
             OSCTXT * pctxt,
             const char * filePath,
             OSOCTET ** ppMsgBuf,
             size_t * pLength )
```

This function reads the entire contents of an ASCII text file into memory.

A memory buffer is allocated for the file contents using the run-time memory management functions. This function is identical to rtxReadFileBinary except that a) the file is opened in text mode, and b) and extra byte is allocated at the end for a null-terminator character.

Parameters

pctxt	Pointer to context block structure.
filePath	Complete file path name of file to read.
ppMsgBuf	Pointer to message buffer to receive allocated memory pointer.
pLength	Pointer to integer to receive length of data read.

Returns

Completion status of operation:

- 0 (ASN_OK) = success,
- RTERR_FILNOTFOU = file not found
- RTERR_FILEREAD = file read error (see errno)

7.26.2.10 rtxFileWriteBinary()

```
EXTERNRT int rtxFileWriteBinary (
             const char * filePath,
             const OSOCTET * pMsgBuf,
             size_t length )
```

This function writes binary data from memory to the given file.

Parameters

filePath	Complete file path name of file to be written to.
pMsgBuf	Pointer to buffer containing data to be written.
length	Size (in bytes) of data to be written

Returns

Completion status of operation:

- 0 = success,
- negative status code if error

7.26.2.11 rtxFileWriteText()

```
EXTERNRT int rtxFileWriteText (
             const char * filePath,
             const char * pMsgBuf )
```

This function writes text data from memory to the given file.

The text is expected to be terminated by a null terminator character. This function will work with standard ASCII or UTF-8 encoded text.

Parameters

filePath	Complete file path name of file to be written to.
pMsgBuf	Pointer to buffer containing data to be written.

Returns

Completion status of operation:

- 0 = success,
- negative status code if error

7.27 rtxFloat.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
```

7.28 rtxGenValueType.h File Reference

Implementation of a generic value type for encoding and decoding values without schema.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxGenValueCompare (OSCTXT ∗pctxt, const OSRTGenValue ∗pvalue1, const OSRTGenValue ∗pvalue2)

  This function compares two generic values for equality.
- EXTERNRT const char ∗ rtxGenValueGetIdent (OSRTDataType id)

  This function returns a textual identifier for the given enumerated value type.

7.28.1 Detailed Description

Implementation of a generic value type for encoding and decoding values without schema.

7.28.2 Function Documentation

7.28.2.1 rtxGenValueCompare()

```
EXTERNRT int rtxGenValueCompare (
            OSCTXT * pctxt,
            const OSRTGenValue * pvalue1,
            const OSRTGenValue * pvalue2 )
```

This function compares two generic values for equality.

Information on values in the structure that are not equal are returned on the error list within the context.

Parameters

pctxt	Pointer to a context structure.
pvalue1	Pointer to first value to compare.
pvalue2	Pointer to second value to compare.

Returns

Status of the comparison operation. A negative value indicates the values are not equal. Printing the error results using rtxErrPrint or equivalent will show the specific items that don't match.

7.28.2.2 rtxGenValueGetIdent()

```
EXTERNRT const char* rtxGenValueGetIdent (
            OSRTDataType id )
```

This function returns a textual identifier for the given enumerated value type.

Parameters

id	Enumerated data type identifier (OSRTDataType)

Returns

Textual identifier for id or '<unknown>'

7.29 rtxHashMap.h File Reference

Generic hash map interface.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT void HASHMAPINITFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap, size_t capacity, OSUINT32(∗hashFunc)(HASHMAPKEYTYPE), OSBOOL(∗keyEqualsFunc)(HASHMAPKEYTYPE, HASHMAPKEYTYPE))

  This function initializes the hash map.
- EXTERNRT HASHMAPTYPENAME ∗ HASHMAPNEWFUNC (OSCTXT ∗pctxt, size_t capacity, OSUINT32(∗hashFunc)(HASHMAPKEYTYPE), OSBOOL(∗keyEqualsFunc)(HASHMAPKEYTYPE, HASHMAPKEYTYPE))

  This function creates a new hash map.
- EXTERNRT HASHMAPTYPENAME ∗ HASHMAPCOPYFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap)

  This function creates a copy of an existing hash map.
- EXTERNRT void HASHMAPFREEFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap)

  This function frees all entries within an existing hash map.
- EXTERNRT int HASHMAPINSERTFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE value)

  This function inserts an entry into the hash map.
- EXTERNRT OSBOOL HASHMAPSEARCHFUNC (HASHMAPTYPENAME ∗pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE ∗pvalue)

  This function searches for an entry in the hash map.
- EXTERNRT OSBOOL HASHMAPREMOVEFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE ∗pvalue)

  This function removes an entry from the hash map.
- EXTERNRT int HASHMAPPUTFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE value)

  This function inserts/replaces an entry into the hash map.
- EXTERNRT int HASHMAPSORTFUNC (OSCTXT ∗pctxt, HASHMAPTYPENAME ∗pHashMap, OSRTDList ∗pSortedList, int(∗compareFunc)(HASHMAPKEYTYPE key1, HASHMAPKEYTYPE key2))

  This function sorts the hash map in ascending order using the given key compare function.

7.29.1 Detailed Description

Generic hash map interface.

This relates a generic key structure (void∗) to a generic value (void∗). Based on "C Hash Table" public domain code (http://www.cl.cam.ac.uk/∼cwc22/hashtable/).

7.29.2 Function Documentation

7.29.2.1 HASHMAPCOPYFUNC()

```
EXTERNRT HASHMAPTYPENAME* HASHMAPCOPYFUNC (
            OSCTXT * pctxt,
            HASHMAPTYPENAME * pHashMap )
```

This function creates a copy of an existing hash map.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure to copy.

Returns

Allocated and copied hash map structure or NULL if insufficient dynamic memory is available to hold the structure.

7.29.2.2 HASHMAPFREEFUNC()

```
EXTERNRT void HASHMAPFREEFUNC (
            OSCTXT * pctxt,
            HASHMAPTYPENAME * pHashMap )
```

This function frees all entries within an existing hash map.

It does not free the structure itself.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure to free.

7.29.2.3 HASHMAPINITFUNC()

```
EXTERNRT void HASHMAPINITFUNC (
            OSCTXT * pctxt,
            HASHMAPTYPENAME * pHashMap,
            size_t capacity,
            OSUINT32(*)(HASHMAPKEYTYPE) hashFunc,
            OSBOOL(*)(HASHMAPKEYTYPE, HASHMAPKEYTYPE) keyEqualsFunc )
```

This function initializes the hash map.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure.
capacity	Capacity of the hash map or zero to use default.
hashFunc	Hash callback function.
keyEqualsFunc	Key equals callback function.

7.29.2.4 HASHMAPINSERTFUNC()

```
EXTERNRT int HASHMAPINSERTFUNC (
            OSCTXT * pctxt,
            HASHMAPTYPENAME * pHashMap,
            HASHMAPKEYTYPE key,
            HASHMAPVALUETYPE value )
```

This function inserts an entry into the hash map.

The table will be expanded if the insertion would take the ratio of entries to table size over the maximum load factor.

This function does not check for repeated insertions with a duplicate key. The value returned when using a duplicate key is undefined – when the hashtable changes size, the order of retrieval of duplicate key entries is reversed. If in doubt, remove before insert.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure.
key	Key value. Memory is owned by caller.
value	Value to insert. Memory is owned by caller.

Returns

Zero if insertion was successful, a negative status code otherwise.

7.29.2.5 HASHMAPNEWFUNC()

```
EXTERNRT HASHMAPTYPENAME* HASHMAPNEWFUNC (
            OSCTXT * pctxt,
            size_t capacity,
            OSUINT32(*)(HASHMAPKEYTYPE) hashFunc,
            OSBOOL(*)(HASHMAPKEYTYPE, HASHMAPKEYTYPE) keyEqualsFunc )
```

This function creates a new hash map.

Parameters

pctxt	Pointer to a context structure.
capacity	Capacity of the map or zero to use default.
hashFunc	Hash callback function.
keyEqualsFunc	Key equals callback function.

Returns

Allocated hash map structure or NULL if insufficient dynamic memory is available to hold the structure.

7.29.2.6 HASHMAPPUTFUNC()

```
EXTERNRT int HASHMAPPUTFUNC (
            OSCTXT * pctxt,
            HASHMAPTYPENAME * pHashMap,
            HASHMAPKEYTYPE key,
            HASHMAPVALUETYPE value )
```

This function inserts/replaces an entry into the hash map.

If the key already exists in the map, its value is updated. Otherwise, the key/value pair is inserted.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure.
key	Key value. Memory is owned by caller.
value	Value to insert/replace. Memory is owned by caller.

Returns

Zero if operation was successful, a negative status code otherwise.

7.29.2.7 HASHMAPREMOVEFUNC()

```
EXTERNRT OSBOOL HASHMAPREMOVEFUNC (
            OSCTXT * pctxt,
            HASHMAPTYPENAME * pHashMap,
            HASHMAPKEYTYPE key,
            HASHMAPVALUETYPE * pvalue )
```

This function removes an entry from the hash map.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure.
key	Key value. Memory is owned by caller.
pvalue	Pointer to value to receive search result value.

Returns

Boolean result: true if found and removed.

7.29.2.8 HASHMAPSEARCHFUNC()

```
EXTERNRT OSBOOL HASHMAPSEARCHFUNC (
            HASHMAPTYPENAME * pHashMap,
            HASHMAPKEYTYPE key,
            HASHMAPVALUETYPE * pvalue )
```

This function searches for an entry in the hash map.

Parameters

pHashMap	Pointer to hash map structure.
key	Key value. Memory is owned by caller.
pvalue	Pointer to value to receive search result value.

Returns

Boolean search result: true if found; false if not.

7.29.2.9 HASHMAPSORTFUNC()

```
EXTERNRT int HASHMAPSORTFUNC (
              OSCTXT * pctxt,
              HASHMAPTYPENAME * pHashMap,
              OSRTDList * pSortedList,
              int(*)(HASHMAPKEYTYPE key1, HASHMAPKEYTYPE key2) compareFunc )
```

This function sorts the hash map in ascending order using the given key compare function.

Parameters

pctxt	Pointer to a context structure.
pHashMap	Pointer to hash map structure.
compareFunc	Comparison function for key values.
pSortedList	Pointer to linked list structure to receive sorted values. Entries within the list are items from the hash map themselves, not copies. List memory may be freed by calling rtxDListFreeNodes.

Returns

Status of operation: 0 = success or negative status code.

7.30 rtxHashMapStr2Int.h File Reference

String-to-integer hash map interface.

```
#include "rtxsrc/rtxHashMapUndef.h"
#include "rtxsrc/rtxHashMap.h"
```

7.30.1 Detailed Description

String-to-integer hash map interface.

This relates a STRING key structure (const OSUTF8CHAR*) to a 32-bit signed integer value (OSINT32). It uses the rtxHashMap .h/.c file as a template.

7.31 rtxHashMapStr2UInt.h File Reference

String-to-unsigned integer hash map interface.

```
#include "rtxsrc/rtxHashMapUndef.h"
#include "rtxsrc/rtxHashMap.h"
```

7.31.1 Detailed Description

String-to-unsigned integer hash map interface.

This relates a string key structure (const OSUTF8CHAR∗) to a 32-bit unsigned integer value (OSUINT32). It uses the rtxHashMap .h/.c file as a template.

7.32 rtxHashMapUndef.h File Reference

Undefine all hash map symbols to allow reuse of the basic definitions in a different of the map.

7.32.1 Detailed Description

Undefine all hash map symbols to allow reuse of the basic definitions in a different of the map.

7.33 rtxHexDump.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxByteToHexChar (OSOCTET byte, char ∗buf, OSSIZE bufsize)

  This function converts a byte value into its hex string equivalent.
- EXTERNRT int rtxByteToHexCharWithPrefix (OSOCTET byte, char ∗buf, OSSIZE bufsize, const char ∗prefix)

  This function converts a byte value into its hex string equivalent.
- EXTERNRT int 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.
- EXTERNRT void rtxHexDumpToFile (FILE ∗fp, const OSOCTET ∗data, OSSIZE numocts)

  This function outputs a hexadecimal dump of the current buffer contents to a file.
- EXTERNRT void rtxHexDumpToFileEx (FILE ∗fp, const OSOCTET ∗data, OSSIZE numocts, OSSIZE bytesPer↩ Unit)

  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.
- EXTERNRT void rtxHexDumpToFileExNoAscii (FILE ∗fp, const OSOCTET ∗data, OSSIZE numocts, OSSIZE 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.
- EXTERNRT int rtxHexDumpToNamedFileNoAscii (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.
- EXTERNRT void rtxHexDump (const OSOCTET ∗data, OSSIZE numocts)

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

- EXTERNRT void rtxHexDumpEx (const OSOCTET ∗data, OSSIZE numocts, OSSIZE 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.
- EXTERNRT 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.
- EXTERNRT int rtxHexDumpToStringEx (const OSOCTET ∗data, OSSIZE numocts, char ∗buffer, OSSIZE bufferIndex, OSSIZE bufferSize, OSSIZE 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.
- EXTERNRT int rtxHexDumpFileContents (const char ∗inFilePath)

  This function outputs a hexadecimal dump of the contents of the named file to stdout.
- EXTERNRT int rtxHexDumpFileContentsToFile (const char ∗inFilePath, const char ∗outFilePath)

  This function outputs a hexadecimal dump of the contents of the named file to a text file.
- EXTERNRT char ∗ rtxHexDiffToDynString (OSCTXT ∗pctxt, const OSOCTET ∗pdata1, const OSOCTET ∗pdata2, OSSIZE numocts)

  This function generates a differences report between two binary data buffers.

7.34 rtxHttp.h File Reference

```
#include "rtxsrc/rtxArrayList.h"
#include "rtxsrc/rtxNetUtil.h"
```

Functions

- EXTERNRT int rtxHttpSendConnectRequest (OSRTNETCONN ∗pNetConn)

  This function executes a full synchronous HTTP CONNECT request to setup a connection through a proxy.
- EXTERNRT int rtxHttpGetTls (OSCTXT ∗pctxt, const char ∗url, OSRTHttpContent ∗pContent, tlsInit_t ∗f_tls_init, tlsClose_t ∗f_tls_close)

  TLS-enabled version of rtxHttpGet.
- EXTERNRT int rtxHttpGet (OSCTXT ∗pctxt, const char ∗url, OSRTHttpContent ∗pContent)

  This function executes a full synchronous HTTP GET request.
- EXTERNRT int rtxHttpSendGetRequest (OSRTNETCONN ∗pNetConn, const char ∗url)

  This function sends an HTTP GET request to a network connection.
- EXTERNRT int rtxHttpSendRequest (OSRTNETCONN ∗pNetConn, const char ∗method, const char ∗content, const char ∗contentType)

  This function sends an HTTP request to a network connection.
- EXTERNRT int rtxHttpRecvRespHdr (OSRTNETCONN ∗pNetConn, OSRTHttpHeader ∗pHeader)

  This function receives the initial header returned from an HTTP request.
- EXTERNRT int rtxHttpRecvContent (OSRTNETCONN ∗pNetConn, OSRTHttpHeader ∗pHeader, OSRTHttpContent ∗pContent)

  This function receives HTTP content.
- EXTERNRT int rtxHttpRecvRespContent (OSRTNETCONN ∗pNetConn, OSRTHttpHeader ∗pHeader, OSRTHttpContent ∗pContent)

  This function receives any HTTP content indicated by the response header.

7.34.1 Function Documentation

7.34.1.1 rtxHttpGet()

```
EXTERNRT int rtxHttpGet (
             OSCTXT * pctxt,
             const char * url,
             OSRTHttpContent * pContent )
```

This function executes a full synchronous HTTP GET request.

A network connection is opened and a GET request sent to the given URL. The response is then read and returned, after which the network connection is closed.

Parameters

pctxt	- Pointer to context structure.
url	- Full URL of get request.
pContent	- Pointer to content buffer to receive response.

Returns

- Operation status: 0 if success, negative code if error.

7.34.1.2 rtxHttpGetTls()

```
EXTERNRT int rtxHttpGetTls (
             OSCTXT * pctxt,
             const char * url,
             OSRTHttpContent * pContent,
             tlsInit_t * f_tls_init,
             tlsClose_t * f_tls_close )
```

TLS-enabled version of rtxHttpGet.

For internal use only. This optionally uses TLS.

This exists to avoid making the runtime dependent on TLS code.

Parameters

f_tls_init	If null, the connection does not use TLS. If the url specifies https and f_tls_init is null, this returns an error. If not null and the url specifies https, then this uses f_tls_init to create a TLS stream on top of the socket stram created by this function. If the url does not specify https, this argument is ignored.
f_tls_close	Must be not null if and only if f_tls_init is not null. This is the function for tearing down a TLS connection.

7.34.1.3 rtxHttpRecvContent()

```
EXTERNRT int rtxHttpRecvContent (
             OSRTNETCONN * pNetConn,
             OSRTHttpHeader * pHeader,
             OSRTHttpContent * pContent )
```

This function receives HTTP content.

All content associated with the response header is stored in the given memory buffer.

Parameters

pNetConn	- Pointer to network connection structure.
pHeader	- Pointer to response header structure describing content.
pContent	- Buffer to receive content. Dynamic memory is allocated for the content using the rtxMemAlloc function.

Returns

- Operation status: 0 if success, negative code if error.

7.34.1.4 rtxHttpRecvRespContent()

```
EXTERNRT int rtxHttpRecvRespContent (
             OSRTNETCONN * pNetConn,
             OSRTHttpHeader * pHeader,
             OSRTHttpContent * pContent )
```

This function receives any HTTP content indicated by the response header.

If the header does not indicate any additional content, the function does nothing.

Parameters

pNetConn	- Pointer to network connection structure.
pHeader	- Pointer to response header structure describing content.
pContent	- Buffer to receive content. Dynamic memory is allocated for the content using the rtxMemAlloc function.

Returns

- Operation status: 0 if success, negative code if error.

7.34.1.5 rtxHttpRecvRespHdr()

```
EXTERNRT int rtxHttpRecvRespHdr (
            OSRTNETCONN * pNetConn,
            OSRTHttpHeader * pHeader )
```

This function receives the initial header returned from an HTTP request.

The header response information is returned in the header structure.

Parameters

pNetConn	- Pointer to network connection structure.
pHeader	- Pointer to header structure to receive returned data.

Returns

- Operation status: 0 if success, negative code if error.

7.34.1.6 rtxHttpSendConnectRequest()

```
EXTERNRT int rtxHttpSendConnectRequest (
            OSRTNETCONN * pNetConn )
```

This function executes a full synchronous HTTP CONNECT request to setup a connection through a proxy.

Parameters

pNetConn	- Pointer to network connection structure.

Returns

- Operation status: 0 if success, negative code if error.

7.34.1.7 rtxHttpSendGetRequest()

```
EXTERNRT int rtxHttpSendGetRequest (
             OSRTNETCONN * pNetConn,
             const char * url )
```

This function sends an HTTP GET request to a network connection.

Parameters

pNetConn	- Pointer to network connection structure.
url	- Full URL of get request. May be set to NULL if URL was provided earlier in rtxNetInitConn function call.

Returns

- Operation status: 0 if success, negative code if error.

7.34.1.8 rtxHttpSendRequest()

```
EXTERNRT int rtxHttpSendRequest (
             OSRTNETCONN * pNetConn,
             const char * method,
             const char * content,
             const char * contentType )
```

This function sends an HTTP request to a network connection.

Parameters

pNetConn	- Pointer to network connection structure.
method	- HTTP method to be used for request (GET or POST)
content	- Content to be sent after header.
contentType	- Type of content.

Returns

- Operation status: 0 if success, negative code if error.

7.35 rtxIntDecode.h File Reference

General purpose integer decode functions.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxDecInt8 (OSCTXT ∗pctxt, OSINT8 ∗pvalue)

  This macro decodes an 8-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecInt16 (OSCTXT ∗pctxt, OSINT16 ∗pvalue, OSSIZE nbytes)

  This function decodes an 16-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecInt32 (OSCTXT ∗pctxt, OSINT32 ∗pvalue, OSSIZE nbytes)

  This function decodes an 32-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecInt64 (OSCTXT ∗pctxt, OSINT64 ∗pvalue, OSSIZE nbytes)

  This function decodes an 64-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecUInt8 (OSCTXT ∗pctxt, OSUINT8 ∗pvalue)

  This macro decodes an 8-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecUInt16 (OSCTXT ∗pctxt, OSUINT16 ∗pvalue, OSSIZE nbytes)

  This function decodes an 16-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecUInt32 (OSCTXT ∗pctxt, OSUINT32 ∗pvalue, OSSIZE nbytes)

  This function decodes an 32-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.
- EXTERNRT int rtxDecUInt64 (OSCTXT ∗pctxt, OSUINT64 ∗pvalue, OSSIZE nbytes)

  This function decodes a 64-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.

7.35.1 Detailed Description

General purpose integer decode functions.

These decode integer value contents that are encoded in big-endian form. This is a common format for a number of different encoding rules.

7.35.2 Function Documentation

7.35.2.1 rtxDecInt16()

```
EXTERNRT int rtxDecInt16 (
             OSCTXT * pctxt,
             OSINT16 * pvalue,
             OSSIZE nbytes )
```

This function decodes an 16-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to receive decoded 16-bit integer value. Pass null to discard the decoded value.
nbytes	Number of bytes to decode (2 or less).

Returns

Completion status of operation:

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

7.35.2.2 rtxDecInt32()

```
EXTERNRT int rtxDecInt32 (
             OSCTXT * pctxt,
             OSINT32 * pvalue,
             OSSIZE nbytes )
```

This function decodes an 32-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to receive decoded 32-bit integer value. Pass null to discard the decoded value.
nbytes	Number of bytes to decode (4 or less).

Returns

Completion status of operation:

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

7.35.2.3 rtxDecInt64()

```
EXTERNRT int rtxDecInt64 (
             OSCTXT * pctxt,
             OSINT64 * pvalue,
             OSSIZE nbytes )
```

This function decodes an 64-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to receive decoded 64-bit integer value. Pass null to discard the decoded value.
nbytes	Number of bytes to decode (8 or less).

Returns

Completion status of operation:

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

7.35.2.4 rtxDecInt8()

```
EXTERNRT int rtxDecInt8 (
             OSCTXT * pctxt,
             OSINT8 * pvalue )
```

This macro decodes an 8-bit signed integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to decoded 8-bit integer value.

Returns

Completion status of operation:

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

7.35.2.5 rtxDecUInt16()

```
EXTERNRT int rtxDecUInt16 (
            OSCTXT * pctxt,
            OSUINT16 * pvalue,
            OSSIZE nbytes )
```

This function decodes an 16-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to decoded 16-bit integer value.
nbytes	Number of bytes to decode (2 or less).

Returns

Completion status of operation:

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

7.35.2.6 rtxDecUInt32()

```
EXTERNRT int rtxDecUInt32 (
            OSCTXT * pctxt,
            OSUINT32 * pvalue,
            OSSIZE nbytes )
```

This function decodes an 32-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to decoded 32-bit integer value.
nbytes	Number of bytes to decode (4 or less).

Returns

Completion status of operation:

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

7.35.2.7 rtxDecUInt64()

```
EXTERNRT int rtxDecUInt64 (
             OSCTXT * pctxt,
             OSUINT64 * pvalue,
             OSSIZE nbytes )
```

This function decodes a 64-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to decoded 64-bit integer value.
nbytes	Number of bytes to decode (8 or less).

Returns

Completion status of operation:

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

7.35.2.8 rtxDecUInt8()

```
EXTERNRT int rtxDecUInt8 (
             OSCTXT * pctxt,
             OSUINT8 * pvalue )
```

This macro decodes an 8-bit unsigned integer at the current message buffer/stream location and advances the pointer to the next field.

Parameters

pctxt	Pointer to context block structure.
pvalue	Pointer to decoded 8-bit integer value.

Returns

Completion status of operation:

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

7.36 rtxIntEncode.h File Reference

General purpose integer encode functions.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxEncUInt32 (OSCTXT ∗pctxt, OSUINT32 value, OSSIZE size)

  This function will encode the given unsigned integer into big-endian form.

7.36.1 Detailed Description

General purpose integer encode functions.

These encode integer value contents into big-endian form which is a common format for a number of different encoding rules.

7.36.2 Function Documentation

7.36.2.1 rtxEncUInt32()

```
EXTERNRT int rtxEncUInt32 (
            OSCTXT * pctxt,
            OSUINT32 value,
            OSSIZE size )
```

This function will encode the given unsigned integer into big-endian form.

One, two, and four byte fixed sizes are supported.

Parameters

pctxt	Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
value	The value to be encoded.
size	Size of the field in bytes into which the value should be encoded (1, 2, or 4).

7.37 rtxIntStack.h File Reference

Simple FIFO stack for storing integer values.

```
#include "rtxsrc/rtxContext.h"
```

Classes

- struct _OSRTIntStack

  This is the main stack structure.

Macros

- #define OSRTISTK_DEFAULT_CAPACITY 100

  This is the default capacity that is used if zero is passed as the capacity argument to rtxIntStackInit.
- #define rtxIntStackIsEmpty(stack) (OSBOOL)((stack).index == 0)

  This macro tests if the stack is empty.

Functions

- EXTERNRT int rtxIntStackInit (OSCTXT ∗pctxt, OSRTIntStack ∗pstack, size_t capacity)

  This function initializes a stack structure.
- EXTERNRT int rtxIntStackPush (OSRTIntStack ∗pstack, OSINT32 value)

  This function pushes an item onto the stack.
- EXTERNRT int rtxIntStackPeek (OSRTIntStack ∗pstack, OSINT32 ∗pvalue)

  This functions returns the data item on the top of the stack.
- EXTERNRT int rtxIntStackPop (OSRTIntStack ∗pstack, OSINT32 ∗pvalue)

  This functions pops the data item on the top of the stack.

7.37.1 Detailed Description

Simple FIFO stack for storing integer values.

7.38 rtxLatin1.h File Reference

Utility functions for converting ISO 8859-1 strings to and from UTF-8.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxLatin1ToUTF8 (const OSUTF8CHAR ∗inbuf, int inlen, OSUTF8CHAR ∗outbuf, int outbufsize)

  This function converts an ISO 8859-1 encoded string into a UTF-8 string.
- EXTERNRT int rtxUTF8ToLatin1 (const OSUTF8CHAR ∗inbuf, int inlen, OSUTF8CHAR ∗outbuf, int outbufsize)

  This function converts a UTF-8 encoded byte stream into an ISO 8859-1 encoded string.
- EXTERNRT int rtxStreamUTF8ToLatin1 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗inbuf, size_t inlen)

  This function converts a UTF-8 encoded byte stream into an ISO 8859-1 encoded string, and write it to stream.

7.38.1 Detailed Description

Utility functions for converting ISO 8859-1 strings to and from UTF-8.

7.38.2 Function Documentation

7.38.2.1 rtxLatin1ToUTF8()

```
EXTERNRT int rtxLatin1ToUTF8 (
             const OSUTF8CHAR ∗ inbuf,
             int inlen,
             OSUTF8CHAR ∗ outbuf,
             int outbufsize )
```

This function converts an ISO 8859-1 encoded string into a UTF-8 string.

A buffer large enough to hold the converted UTF-8 characters must be provided. A buffer providing 4 bytes-per-character should be large enough to hold the largest possible UTF-8 conversion.

Parameters

inbuf	A pointer to an array of ISO 8859-1 characters.
inlen	Number of ISO 8859-1 characters to be converted.
outbuf	Buffer to hold converted string.
outbufsize	Size of output buffer.

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space

7.38.2.2 rtxStreamUTF8ToLatin1()

```
EXTERNRT int rtxStreamUTF8ToLatin1 (
              OSCTXT * pctxt,
              const OSUTF8CHAR * inbuf,
              size_t inlen )
```

This function converts a UTF-8 encoded byte stream into an ISO 8859-1 encoded string, and write it to stream.

Parameters

pctxt	Pointer to context block structure.
inbuf	A pointer to an array of UTF-8 string.
inlen	Number of bytes of the input string.

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.38.2.3 rtxUTF8ToLatin1()

```
EXTERNRT int rtxUTF8ToLatin1 (
              const OSUTF8CHAR * inbuf,
              int inlen,
              OSUTF8CHAR * outbuf,
              int outbufsize )
```

This function converts a UTF-8 encoded byte stream into an ISO 8859-1 encoded string.

A buffer large enough to hold the converted characters must be provided.

Parameters

inbuf	A pointer to an array of UTF-8 string.
inlen	Number of bytes of the input string.
outbuf	Buffer to hold converted string.
outbufsize	Size of output buffer.

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.39 rtxMemBuf.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxMemBufAppend (OSRTMEMBUF ∗pMemBuf, const OSOCTET ∗pdata, OSSIZE nbytes)

  This function appends the data to the end of a memory buffer.
- EXTERNRT int rtxMemBufCut (OSRTMEMBUF ∗pMemBuf, OSSIZE fromOffset, OSSIZE nbytes)

  This function cuts off the part of memory buffer.
- EXTERNRT void rtxMemBufFree (OSRTMEMBUF ∗pMemBuf)

  This function frees the memory buffer.
- EXTERNRT OSOCTET ∗ rtxMemBufGetData (const OSRTMEMBUF ∗pMemBuf, int ∗length)

  This function returns the pointer to the used part of a memory buffer.
- EXTERNRT OSOCTET ∗ rtxMemBufGetDataExt (const OSRTMEMBUF ∗pMemBuf, OSSIZE ∗length)

  This function returns the pointer to the used part of a memory buffer.
- EXTERNRT OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF ∗pMemBuf)

  This function returns the length of the used part of a memory buffer.
- EXTERNRT void rtxMemBufInit (OSCTXT ∗pCtxt, OSRTMEMBUF ∗pMemBuf, OSSIZE segsize)

  This function initializes a memory buffer structure.
- EXTERNRT void rtxMemBufInitBuffer (OSCTXT ∗pCtxt, OSRTMEMBUF ∗pMemBuf, OSOCTET ∗buf, OSSIZE bufsize, OSSIZE segsize)

  This function assigns a static buffer to the memory buffer structure.
- EXTERNRT int rtxMemBufPreAllocate (OSRTMEMBUF ∗pMemBuf, OSSIZE nbytes)

  This function allocates a buffer with a predetermined amount of space.
- EXTERNRT void rtxMemBufReset (OSRTMEMBUF ∗pMemBuf)

  This function resets the memory buffer structure.
- EXTERNRT int rtxMemBufSet (OSRTMEMBUF ∗pMemBuf, OSOCTET value, OSSIZE nbytes)

  This function sets part of a memory buffer to a specified octet value.
- EXTERNRT OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF ∗pMemBuf, OSBOOL isExpandable)

  This function sets "isExpandable" flag for the memory buffer object.
- EXTERNRT 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.
- EXTERNRT OSSIZE rtxMemBufTrimW (OSRTMEMBUF ∗pMemBuf)

  This function trims white space of the memory buffer.

7.40 rtxMemory.h File Reference

Memory management function and macro definitions.

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define OSRTALLOCTYPE(pctxt, type) (type∗) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))

  This macro allocates a single element of the given type.
- #define OSRTALLOCTYPEZ(pctxt, type) (type∗) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))

  This macro allocates and zeros a single element of the given type.
- #define OSRTREALLOCARRAY(pctxt, pseqof, type)

  Reallocate an array.
- #define rtxMemAlloc(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap,nbytes)

  Allocate memory.
- #define rtxMemSysAlloc(pctxt, nbytes) rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)

  This macro makes a direct call to the configured system memory allocation function.
- #define rtxMemAllocZ(pctxt, nbytes) rtxMemHeapAllocZ(&(pctxt)->pMemHeap,nbytes)

  Allocate and zero memory.
- #define rtxMemSysAllocZ(pctxt, nbytes) rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)

  Allocate and zero memory.
- #define rtxMemRealloc(pctxt, mem_p, nbytes) rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, nbytes)

  Reallocate memory.
- #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.
- #define rtxMemFreePtr(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemSysFreePtr(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  This macro makes a direct call to the configured system memory free function.
- #define rtxMemAllocType(pctxt, ctype) (ctype∗)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type.
- #define rtxMemSysAllocType(pctxt, ctype) (ctype∗)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type.
- #define rtxMemAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type and zero memory.
- #define rtxMemSysAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))

  Allocate type and zero memory.
- #define rtxMemFreeType(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemSysFreeType(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemAllocArray(pctxt, n, type) (type∗)rtxMemAllocArray2 (pctxt, n, sizeof(type), 0)

  Allocate a dynamic array.
- #define rtxMemSysAllocArray(pctxt, n, type) (type∗)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_SYSALLOC)

  Allocate a dynamic array.
- #define rtxMemAllocArrayZ(pctxt, n, type) (type∗)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_ZEROARRAY)

  Allocate a dynamic array and zero memory.
- #define rtxMemFreeArray(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.

- #define rtxMemSysFreeArray(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Free memory pointer.
- #define rtxMemReallocArray(pctxt, mem_p, n, type) (type∗)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, sizeof(type)∗n)

  Reallocate memory.
- #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.
- #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void∗)(ptr))

  This function increments the auto-pointer reference count.
- #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void∗)(ptr))

  This function decrements the auto-pointer reference count.
- #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void∗)(ptr))

  This function returns the reference count of the given pointer.
- #define rtxMemCheckPtr(pctxt, mem_p) rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void∗)mem_p)

  Check memory pointer.
- #define rtxMemCheck(pctxt) rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)

  Check memory heap.
- #define rtxMemPrint(pctxt) rtxMemHeapPrint(&(pctxt)->pMemHeap)

  Print memory heap structure to stderr.
- #define rtxMemPrintWithFree(pctxt) rtxMemHeapPrintWithFree(&(pctxt)->pMemHeap)

  Print memory heap structure to stderr, the same as rtxMemPrint, but add details about the memory free list.
- #define rtxMemSetProperty(pctxt, propId, pProp) rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, p↵ Prop)

  Set memory heap property.

Functions

- EXTERNRT void rtxMemHeapPrint (void ∗∗ppvMemHeap)

  Print details about the memory heap.
- EXTERNRT void rtxMemHeapPrintWithFree (void ∗∗ppvMemHeap)

  Print the same details about the memory heap as rtxMemHeapPrint but add deatils about the free memory list.
- EXTERNRT int rtxMemHeapCreate (void ∗∗ppvMemHeap)

  This function creates a standard memory heap.
- EXTERNRT int rtxMemHeapCreateExt (void ∗∗ppvMemHeap, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

  This function creates a standard memory heap and sets the low-level memory functions to the specified values.
- EXTERNRT int rtxMemStaticHeapCreate (void ∗∗ppvMemHeap, void ∗pmem, OSSIZE memsize)

  This function creates a static memory heap.
- EXTERNRT int rtxMemHeapConvertStatic (void ∗∗ppvMemHeap, void ∗pmem, OSSIZE memsize)

  This function converts a standard memory heap to a static memory heap.
- EXTERNRT void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

  This function sets the pointers to standard allocation functions.
- EXTERNRT OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT ∗pctxt)

  This function returns the actual granularity of memory blocks in the context.
- EXTERNRT void rtxMemSetDefBlkSize (OSUINT32 blkSize)

This function sets the minimum size and the granularity of memory blocks for newly created memory heaps.

- EXTERNRT OSUINT32 rtxMemGetDefBlkSize (OSVOIDARG)

  This function returns the actual granularity of memory blocks.
- EXTERNRT OSBOOL rtxMemHeapIsEmpty (OSCTXT ∗pctxt)

  This function determines if the memory heap defined in the give context is empty (i.e.
- EXTERNRT OSBOOL rtxMemIsZero (const void ∗pmem, OSSIZE memsiz)

  This helper function determines if an arbitrarily sized block of memory is set to zero.
- EXTERNRT void rtxMemFree (OSCTXT ∗pctxt)

  Free memory associated with a context.
- EXTERNRT void rtxMemReset (OSCTXT ∗pctxt)

  Reset memory associated with a context.

7.40.1 Detailed Description

Memory management function and macro definitions.

7.41 rtxMemStatic.h File Reference

Simplistic low-level memory manager which allocates memory from out of given block of memory.

```
#include "osSysTypes.h"
```

Functions

- void rtxMemSetStaticAlloc (OSOCTET ∗staticHeap, OSSIZE staticHeapSize)

  Install a simplistic low-level memory manager, by calling rtxMemSetAllocFuncs (which see).
- void rtxMemFreeStatic ()

  Frees all memory made by the static allocator (which is set by rtxMemSetStaticAlloc).

7.41.1 Detailed Description

Simplistic low-level memory manager which allocates memory from out of given block of memory.

7.41.2 Function Documentation

7.41.2.1 rtxMemFreeStatic()

```
void rtxMemFreeStatic ( )
```

Frees all memory made by the static allocator (which is set by rtxMemSetStaticAlloc).

This makes all of the memory, given to rtxMemSetStaticAlloc, available for allocation again. Before calling this function, all contexts must have been freed by calling rtFreeContext for each context.

7.41.2.2 rtxMemSetStaticAlloc()

```
void rtxMemSetStaticAlloc (
             OSOCTET * staticHeap,
             OSSIZE staticHeapSize )
```

Install a simplistic low-level memory manager, by calling rtxMemSetAllocFuncs (which see).

The low-level manager is responsible for allocating memory to the high-level manager and to code which does not have access to a high-level manager (e.g. during context initialization).

The installed manager has these properties:

- Allocations are made out of the given block of memory.
- Does not support memory reallocation.
- Does not support freeing individually allocated blocks.
- Is not thread-safe; not suitable for multi-threaded use of Objective Systems code.

This manager is intended for use in applications where the system memory allocator cannot be used to dynamically allocate memory, to allow for some limited dynamic memory allocations required by the Objective Systems' runtime.

The built-in (high-level) memory manager uses the low-level manager to obtain more memory when needed. The high-level manager may itself be configured to satisfy memory requests to it by carving up a given block of memory; this is done by calling rtxMemStaticHeapCreate or rtxMemHeapConvertStatic. When the high-level manager is set to use a static heap, fewer requests will be made of the low-level manager.

This function must be called before any memory allocations are made, which means it must be called before any calls are made to initialize a context.

This function may be called to provide a new staticHeap any time that rtxMemFreeStatic could be called.

7.42 rtxNetUtil.h File Reference

```
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxSocket.h"
```

Functions

- EXTERNRT OSRTNETCONN ∗ rtxNetCreateConn (OSCTXT ∗pctxt, const char ∗url)

  This function creates a new network connection to the given URL.
- EXTERNRT int rtxNetCloseConn (OSRTNETCONN ∗pNetConn)

  This function closes a network connection.
- EXTERNRT int rtxNetConnect (OSRTNETCONN ∗pNetConn)

  This function creates a network connection.
- EXTERNRT int rtxNetInitConn (OSCTXT ∗pctxt, OSRTNETCONN ∗pNetConn, const char ∗url)

  This function initializes a network connection structure.
- EXTERNRT int rtxNetParseURL (OSRTNETCONN ∗pNetConn, const char ∗url)

  This function parses a Universal Resource Locator (URL) into the components defined in the network connection structure.

7.42.1 Function Documentation

7.42.1.1 rtxNetCloseConn()

```
EXTERNRT int rtxNetCloseConn (
            OSRTNETCONN ∗ pNetConn )
```

This function closes a network connection.

Parameters

pNetConn	- Pointer to network connection structure. This is assumed to have been created using the rtxNetCreateNewConn.

Returns

- Operation status: 0 if success, negative code if error.

7.42.1.2 rtxNetConnect()

```
EXTERNRT int rtxNetConnect (
            OSRTNETCONN ∗ pNetConn )
```

This function creates a network connection.

The network entity is described by a network connection structure. The network structure may have been created from a URL using the `rtxNetCreateConn` function or may have been initialized manually.

If a proxy is specified in the proxy field, this connects to the proxy.

Parameters

pNetConn	- Pointer to network connection structure.

Returns

- Operation status: 0 if success, negative code if error.

See also

rtxNetCreateConn

7.42.1.3 rtxNetCreateConn()

```
EXTERNRT OSRTNETCONN* rtxNetCreateConn (
            OSCTXT * pctxt,
            const char * url )
```

This function creates a new network connection to the given URL.

Parameters

pctxt	- Pointer to run-time context structure.
url	- URL to which to connect.

Returns

- Pointer to allocated network connection object or null if error. If error, error information is stored in context variable and can be accessed using rtxErr∗ functions. Allocated memory will be freed when rtxNetCloseConn is called.

7.42.1.4 rtxNetInitConn()

```
EXTERNRT int rtxNetInitConn (
            OSCTXT * pctxt,
            OSRTNETCONN * pNetConn,
            const char * url )
```

This function initializes a network connection structure.

The given URL is parsed into components and stored in the structure.

Parameters

pctxt	- Pointer to run-time context structure.
pNetConn	- Pointer to network connection structure to be initialized.
url	- URL to be parsed.

Returns

- Status of initialization operation. 0 == success, negative status code = error.

7.42.1.5 rtxNetParseURL()

```
EXTERNRT int rtxNetParseURL (
             OSRTNETCONN * pNetConn,
             const char * url )
```

This function parses a Universal Resource Locator (URL) into the components defined in the network connection structure.

Parameters

pNetConn	- Pointer to network connection structure.
url	- URL to be parsed.

Returns

- Status of the parse operation. 0 == success, negative status code = error.

7.43 rtxPattern.h File Reference

Pattern matching functions.

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT OSBOOL rtxMatchPattern (OSCTXT ∗pctxt, const OSUTF8CHAR ∗text, const OSUTF8CHAR ∗pattern)

  This function compares the given string to the given pattern.
- EXTERNRT void rtxFreeRegexpCache (OSCTXT ∗pctxt)

  This function frees the memory associated with the regular expression cache.

7.43.1 Detailed Description

Pattern matching functions.

7.44 rtxPcapFile.h File Reference

Data structures and functions for processing PCAP files.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxDecPcapHdr (OSCTXT ∗pctxt, pcap_hdr_t ∗pPcapHdr, OSBOOL ∗pswapped)

  This function decodes a PCAP global header structure.
- EXTERNRT int rtxDecPcapRecHdr (OSCTXT ∗pctxt, pcaprec_hdr_t ∗pPcapRecHdr, OSBOOL swapped)

  This function decodes a PCAP record header structure.
- EXTERNRT int rtxDecEtherHdr (OSCTXT ∗pctxt, ether_header_t ∗pEtherHdr, OSBOOL swapped)

  This function decodes an Ethernet header structure.
- EXTERNRT void rtxPrintPcapHdr (const pcap_hdr_t ∗pPcapHdr)

  This function prints the contents of a PCAP global header structure to stdout.
- EXTERNRT void rtxPrintPcapRecHdr (const pcaprec_hdr_t ∗pPcapRecHdr)

  This function prints the contents of a PCAP record header structure to stdout.

7.44.1 Detailed Description

Data structures and functions for processing PCAP files.

7.44.2 Function Documentation

7.44.2.1 rtxDecEtherHdr()

```
EXTERNRT int rtxDecEtherHdr (
             OSCTXT ∗ pctxt,
             ether_header_t ∗ pEtherHdr,
             OSBOOL swapped )
```

This function decodes an Ethernet header structure.

Parameters

pctxt	Pointer to context holding message to decode.
pEtherHdr	Pointer to structure to receive decoded header. If pointer is NULL, header record will be skipped.
swapped	Boolean indicating if byte swapping needs to be done when decoding integer values.

Returns

Status of decode operation: 0 = success, negative value = error

7.44.2.2 rtxDecPcapHdr()

```
EXTERNRT int rtxDecPcapHdr (
             OSCTXT * pctxt,
             pcap_hdr_t * pPcapHdr,
             OSBOOL * pswapped )
```

This function decodes a PCAP global header structure.

Parameters

pctxt	Pointer to context holding message to decode.
pPcapHdr	Pointer to structure to receive decoded header. If pointer is NULL, header record will be skipped.
pswapped	Boolean to receive results of magic number analysis to determine bytes need to be swapped when decoding integer values.

Returns

Status of decode operation: 0 = success, negative value = error

7.44.2.3 rtxDecPcapRecHdr()

```
EXTERNRT int rtxDecPcapRecHdr (
             OSCTXT * pctxt,
             pcaprec_hdr_t * pPcapRecHdr,
             OSBOOL swapped )
```

This function decodes a PCAP record header structure.

Parameters

pctxt	Pointer to context holding message to decode.
pPcapRecHdr	Pointer to structure to receive decoded header. If pointer is NULL, header record will be skipped.
swapped	Boolean indicating if byte swapping needs to be done when decoding integer values.

Returns

Status of decode operation: 0 = success, negative value = error

7.44.2.4 rtxPrintPcapHdr()

```
EXTERNRT void rtxPrintPcapHdr (
            const pcap_hdr_t * pPcapHdr )
```

This function prints the contents of a PCAP global header structure to stdout.

Parameters

pPcapHdr	Pointer to structure to print.

7.44.2.5 rtxPrintPcapRecHdr()

```
EXTERNRT void rtxPrintPcapRecHdr (
            const pcaprec_hdr_t * pPcapRecHdr )
```

This function prints the contents of a PCAP record header structure to stdout.

Parameters

pPcapRecHdr	Pointer to structure to print.

7.45 rtxPrint.h File Reference

```
#include <stdio.h>
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxHexDump.h"
```

Functions

- EXTERNRT void rtxPrintBoolean (const char ∗name, OSBOOL value)

    Prints a boolean value to stdout.
- EXTERNRT void rtxPrintDate (const char ∗name, const OSNumDateTime ∗pvalue)

    Prints a date value to stdout.
- EXTERNRT void rtxPrintTime (const char ∗name, const OSNumDateTime ∗pvalue)

    Prints a time value to stdout.
- EXTERNRT void rtxPrintDateTime (const char ∗name, const OSNumDateTime ∗pvalue)

    Prints a dateTime value to stdout.
- EXTERNRT void rtxPrintInteger (const char ∗name, OSINT32 value)

    Prints an integer value to stdout.
- EXTERNRT void rtxPrintInt64 (const char ∗name, OSINT64 value)

    Prints a 64-bit integer value to stdout.
- EXTERNRT void rtxPrintIpv4Addr (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    This function prints the value of a binary string in IPv4 address format to standard output.
- EXTERNRT void rtxPrintIpv6Addr (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    This function prints the value of a binary string in IPv6 address format to standard output.
- EXTERNRT void rtxPrintTBCDStr (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    This function prints the value of a binary string in TBCD format to standard output.
- EXTERNRT void rtxPrintText (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    This function prints the value of a binary string in ASCII text format to standard output.
- EXTERNRT void rtxPrintUnsigned (const char ∗name, OSUINT32 value)

    Prints an unsigned integer value to stdout.
- EXTERNRT void rtxPrintUInt64 (const char ∗name, OSUINT64 value)

    Prints an unsigned 64-bit integer value to stdout.
- EXTERNRT 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.
- EXTERNRT void rtxPrintHexStrPlain (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintHexStrNoAscii (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintHexBinary (const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

    Prints an octet string value in hex binary format to stdout.
- EXTERNRT void rtxPrintCharStr (const char ∗name, const char ∗cstring)

    Prints an ASCII character string value to stdout.
- EXTERNRT void rtxPrintUTF8CharStr (const char ∗name, const OSUTF8CHAR ∗cstring)

    Prints a UTF-8 encoded character string value to stdout.
- EXTERNRT void rtxPrintUnicodeCharStr (const char ∗name, const OSUNICHAR ∗str, int nchars)

    This function prints a Unicode string to standard output.
- EXTERNRT void rtxPrintReal (const char ∗name, OSREAL value)

    Prints a REAL (float, double, decimal) value to stdout.
- EXTERNRT void rtxPrintNull (const char ∗name)

    Prints a NULL value to stdout.
- EXTERNRT void rtxPrintNVP (const char ∗name, const OSUTF8NVP ∗value)

    Prints a name-value pair to stdout.
- EXTERNRT void rtxPrintArrayNVP (const char ∗name, OSSIZE subscript, const OSUTF8NVP ∗value)

Prints a name-value pair to stdout.

- EXTERNRT int rtxPrintFile (const char ∗filename)

  This function prints the contents of a text file to stdout.
- EXTERNRT void rtxPrintIndent (OSVOIDARG)

  This function prints indentation spaces to stdout.
- EXTERNRT void rtxPrintIncrIndent (OSVOIDARG)

  This function increments the current indentation level.
- EXTERNRT void rtxPrintDecrIndent (OSVOIDARG)

  This function decrements the current indentation level.
- EXTERNRT void rtxPrintCloseBrace (OSVOIDARG)

  This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.
- EXTERNRT 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.
- EXTERNRT const char ∗ rtxGetArrayElemName (char ∗buffer, OSSIZE bufsize, const char ∗name, OSSIZE subscript)

  This function returns an array element name in the form of 'name[subscript]' in the given character array buffer.

7.46 rtxPrintStream.h File Reference

Functions that allow printing diagnostic message to a stream using a callback function.

```
#include <stdarg.h>
#include "rtxsrc/rtxContext.h"
```

Classes

- struct OSRTPrintStream

  Structure to hold information about a global PrintStream.

Typedefs

- typedef void(∗ rtxPrintCallback) (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)

  Callback function definition for print stream.
- typedef struct OSRTPrintStream OSRTPrintStream

  Structure to hold information about a global PrintStream.

Functions

- EXTERNRT int rtxSetPrintStream (OSCTXT *pctxt, rtxPrintCallback myCallback, void *pStrmInfo)

  This function is for setting the callback function for a PrintStream.
- EXTERNRT int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void *pStrmInfo)

  This function is for setting the callback function for a PrintStream.
- EXTERNRT 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.
- EXTERNRT int rtxDiagToStream (OSCTXT *pctxt, const char *fmtspec, va_list arglist)

  Diagnostics print-to-stream function.
- EXTERNRT int rtxPrintStreamRelease (OSCTXT *pctxt)

  This function releases the memory held by PrintStream in the context.
- EXTERNRT void rtxPrintStreamToStdoutCB (void *pPrntStrmInfo, const char *fmtspec, va_list arglist)

  Standard callback function for use with print-to-stream for writing to stdout.
- EXTERNRT void rtxPrintStreamToFileCB (void *pPrntStrmInfo, const char *fmtspec, va_list arglist)

  Standard callback function for use with print-to-stream for writing to a file.
- EXTERNRT void rtxPrintStreamToStringCB (void *pPrntStrmInfo, const char *fmtspec, va_list arglist)

  Standard callback function for use with print-to-stream for writing to a string.

Variables

- OSRTPrintStream g_PrintStream

  Global PrintStream.

7.46.1 Detailed Description

Functions that allow printing diagnostic message to a stream using a callback function.

7.47 rtxPrintToStream.h File Reference

```
#include <stdio.h>
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT void rtxPrintToStreamBoolean (OSCTXT ∗pctxt, const char ∗name, OSBOOL value)

  Prints a boolean value to a print stream.
- EXTERNRT void rtxPrintToStreamDate (OSCTXT ∗pctxt, const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a date value to a print stream.
- EXTERNRT void rtxPrintToStreamTime (OSCTXT ∗pctxt, const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a time value to a print stream.
- EXTERNRT void rtxPrintToStreamDateTime (OSCTXT ∗pctxt, const char ∗name, const OSNumDateTime ∗pvalue)

  Prints a dateTime value to a print stream.
- EXTERNRT void rtxPrintToStreamInteger (OSCTXT ∗pctxt, const char ∗name, OSINT32 value)

  Prints an integer value to a print stream.
- EXTERNRT void rtxPrintToStreamInt64 (OSCTXT ∗pctxt, const char ∗name, OSINT64 value)

  Prints a 64-bit integer value to a print stream.
- EXTERNRT void rtxPrintToStreamUnsigned (OSCTXT ∗pctxt, const char ∗name, OSUINT32 value)

  Prints an unsigned integer value to a print stream.
- EXTERNRT void rtxPrintToStreamUInt64 (OSCTXT ∗pctxt, const char ∗name, OSUINT64 value)

  Prints an unsigned 64-bit integer value to a print stream.
- EXTERNRT void rtxPrintToStreamHexStr (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OSOC↩ TET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintToStreamHexStrPlain (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const O↩ SOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintToStreamHexStrNoAscii (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OSOCTET ∗data)

  This function prints the value of a binary string in hex format to standard output.
- EXTERNRT void rtxPrintToStreamHexBinary (OSCTXT ∗pctxt, const char ∗name, OSSIZE numocts, const OS↩ OCTET ∗data)

  Prints an octet string value in hex binary format to a print stream.
- EXTERNRT void rtxPrintToStreamCharStr (OSCTXT ∗pctxt, const char ∗name, const char ∗cstring)

  Prints an ASCII character string value to a print stream.
- EXTERNRT void rtxPrintToStreamUTF8CharStr (OSCTXT ∗pctxt, const char ∗name, const OSUTF8CHAR ∗cstring)

  Prints a UTF-8 encoded character string value to a print stream.
- EXTERNRT void rtxPrintToStreamUnicodeCharStr (OSCTXT ∗pctxt, const char ∗name, const OSUNICHAR ∗str, int nchars)

  This function prints a Unicode string to standard output.
- EXTERNRT void rtxPrintToStreamReal (OSCTXT ∗pctxt, const char ∗name, OSREAL value)

  Prints a REAL (float, double, decimal) value to a print stream.
- EXTERNRT void rtxPrintToStreamNull (OSCTXT ∗pctxt, const char ∗name)

  Prints a NULL value to a print stream.
- EXTERNRT void rtxPrintToStreamNVP (OSCTXT ∗pctxt, const char ∗name, const OSUTF8NVP ∗value)

  Prints a name-value pair to a print stream.
- EXTERNRT int rtxPrintToStreamFile (OSCTXT ∗pctxt, const char ∗filename)

  This function prints the contents of a text file to a print stream.
- EXTERNRT void rtxPrintToStreamIndent (OSCTXT ∗pctxt)

This function prints indentation spaces to a print stream.

- EXTERNRT void rtxPrintToStreamResetIndent (OSCTXT ∗pctxt)

  This function resets the current indentation level to zero.
- EXTERNRT void rtxPrintToStreamIncrIndent (OSCTXT ∗pctxt)

  This function increments the current indentation level.
- EXTERNRT void rtxPrintToStreamDecrIndent (OSCTXT ∗pctxt)

  This function decrements the current indentation level.
- EXTERNRT void rtxPrintToStreamCloseBrace (OSCTXT ∗pctxt)

  This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.
- EXTERNRT 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.
- EXTERNRT void rtxHexDumpToStream (OSCTXT ∗pctxt, const OSOCTET ∗data, OSSIZE numocts)

  This function outputs a hexadecimal dump of the current buffer contents to a print stream.
- EXTERNRT void rtxHexDumpToStreamEx (OSCTXT ∗pctxt, const OSOCTET ∗data, OSSIZE numocts, OSSIZE 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.
- EXTERNRT void rtxHexDumpToStreamExNoAscii (OSCTXT ∗pctxt, const OSOCTET ∗data, OSSIZE numocts, OSSIZE bytesPerUnit)

  This function outputs a formatted hexadecimal dump of the current buffer to a print stream.

7.48 rtxReal.h File Reference

Common runtime functions for working with floating-point numbers.

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
```

Functions

- EXTERNRT OSREAL rtxGetMinusInfinity (OSVOIDARG)

  Returns the IEEE negative infinity value.
- EXTERNRT OSREAL rtxGetMinusZero (OSVOIDARG)

  Returns the IEEE minus zero value.
- EXTERNRT OSREAL rtxGetNaN (OSVOIDARG)

  Returns the IEEE Not-A-Number (NaN) value.
- EXTERNRT OSREAL rtxGetPlusInfinity (OSVOIDARG)

  Returns the IEEE posative infinity value.
- EXTERNRT OSBOOL rtxIsMinusInfinity (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for negative infinity.
- EXTERNRT OSBOOL rtxIsMinusZero (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for minus zero.
- EXTERNRT OSBOOL rtxIsNaN (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for Not-A-Number (NaN).

- EXTERNRT OSBOOL rtxIsPlusInfinity (OSREAL value)

  A utility function that compares the given input value to the IEEE 754 value for positive infinity.
- EXTERNRT 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.
- EXTERNRT 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.

7.48.1 Detailed Description

Common runtime functions for working with floating-point numbers.

7.49 rtxScalarDList.h File Reference

Doubly-linked list utility functions to hold scalar data variables.

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
```

Classes

- struct OSRTScalarDListNode

  This structure is used to hold a single data item within the list.
- struct OSRTScalarDList

  This is the main list structure.

Functions

- EXTERNRT void rtxScalarDListInit (OSRTScalarDList *pList)

  This function initializes a doubly linked list structure.
- EXTERNRT OSRTScalarDListNode * rtxScalarDListAppendDouble (struct OSCTXT *pctxt, OSRTScalarDList *pList, OSDOUBLE value)

  This set of functions appends an item of the given scalar type to the linked list structure.
- EXTERNRT OSRTScalarDListNode * rtxScalarDListAppendNode (OSRTScalarDList *pList, OSRTScalarDListNode *pListNode)

  This function is used to append a node to the linked list.
- EXTERNRT OSRTScalarDListNode * rtxScalarDListInsertNode (OSRTScalarDList *pList, OSUINT32 idx, OSRTScalarDListNode *pListNode)

  This function is used to insert a node into the linked list.
- EXTERNRT OSRTScalarDListNode * rtxScalarDListFindByIndex (const OSRTScalarDList *pList, OSUINT32 idx)

  This function will return the node pointer of the indexed entry in the list.
- EXTERNRT void rtxScalarDListFreeNode (struct OSCTXT *pctxt, OSRTScalarDList *pList, OSRTScalarDListNode *node)

  This function will remove the given node from the list and free memory.
- EXTERNRT void rtxScalarDListRemove (OSRTScalarDList *pList, OSRTScalarDListNode *node)

  This function will remove the given node from the list.
- EXTERNRT void rtxScalarDListFreeNodes (struct OSCTXT *pctxt, OSRTScalarDList *pList)

  This function will free all of the dynamic memory used to hold the list node pointers.

7.49.1 Detailed Description

Doubly-linked list utility functions to hold scalar data variables.

7.50 rtxSOAP.h File Reference

: common SOAP socket communications functions

```
#include "rtxsrc/rtxCommon.h"
#include "rtxsrc/rtxSocket.h"
```

Functions

- EXTERNRT int rtxSoapInitConn (OSSOAPCONN ∗pSoapConn, OSCTXT ∗pctxt, OSSoapVersion soapv, const char ∗url)

  This function initializes a connection to a SOAP endpoint.
- EXTERNRT int rtxSoapAcceptConn (OSRTSOCKET listenSocket, OSSOAPCONN ∗pSoapConn)

  This function accepts an incoming connection request and sets up a stream on which to receive messages.
- EXTERNRT int rtxSoapConnect (OSSOAPCONN ∗pSoapConn)

  This function creates a connection to a SOAP endpoint.
- EXTERNRT int rtxSoapRecvHttp (OSSOAPCONN ∗pSoapConn)

  This function receives the initial header returned from an HTTP request.
- EXTERNRT int rtxSoapRecvHttpContent (OSSOAPCONN ∗pSoapConn, OSOCTET ∗∗ppbuf)

  This function receives a complete HTTP response from a SOAP connection.
- EXTERNRT int rtxSoapSendHttpResponse (OSSOAPCONN ∗pSoapConn, const OSUTF8CHAR ∗soapMsg)

  This function sends a SOAP message as an HTTP response.
- EXTERNRT int rtxSoapSendHttp (OSSOAPCONN ∗pSoapConn, const OSUTF8CHAR ∗soapMsg)

  This function sends a complete HTTP POST request to a SOAP connection.
- EXTERNRT int rtxSoapSetReadTimeout (OSSOAPCONN ∗pSoapConn, OSUINT32 nsecs)

  This function sets the read timeout value to the given number of seconds.

7.50.1 Detailed Description

: common SOAP socket communications functions

7.50.2 Function Documentation

7.50.2.1 rtxSoapAcceptConn()

```
EXTERNRT int rtxSoapAcceptConn (
            OSRTSOCKET listenSocket,
            OSSOAPCONN ∗ pSoapConn )
```

This function accepts an incoming connection request and sets up a stream on which to receive messages.

Parameters

listenSocket	- Listener socket
pSoapConn	- Pointer to SOAP connection structure.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.2 rtxSoapConnect()

```
EXTERNRT int rtxSoapConnect (
             OSSOAPCONN * pSoapConn )
```

This function creates a connection to a SOAP endpoint.

The endpoint is described by a SOAP connection structure which must have been initialized using the rtxSoapInitConn function.

Parameters

pSoapConn	- Pointer to SOAP connection structure.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.3 rtxSoapInitConn()

```
EXTERNRT int rtxSoapInitConn (
             OSSOAPCONN * pSoapConn,
             OSCTXT * pctxt,
             OSSoapVersion soapv,
             const char * url )
```

This function initializes a connection to a SOAP endpoint.

Parameters

pSoapConn	- Pointer to SOAP connection structure.
pctxt	- Pointer to an XBinder run-time context structure.
soapv	- SOAP version that is to be used.
url	- URL to which to connect.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.4 rtxSoapRecvHttp()

```
EXTERNRT int rtxSoapRecvHttp (
             OSSOAPCONN * pSoapConn )
```

This function receives the initial header returned from an HTTP request.

The header response information including content length and whether the response is 'chunked' is stored in the connection structure.

Parameters

pSoapConn	- Pointer to SOAP connection structure.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.5 rtxSoapRecvHttpContent()

```
EXTERNRT int rtxSoapRecvHttpContent (
             OSSOAPCONN * pSoapConn,
             OSOCTET ** ppbuf )
```

This function receives a complete HTTP response from a SOAP connection.

The response if stored in a dynamic memory buffer which is returned via the buffer pointer argument. Memory is allocated for the response using XBinder memory management, so it will be freed when the context is freed or the rtx↩MemFree function is called. This buffer can now be used in a decode function call to parse the received XML message into a program structure.

Parameters

pSoapConn	- Pointer to SOAP connection structure.
ppbuf	- Pointer to pointer to receive content buffer.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.6 rtxSoapSendHttp()

```
EXTERNRT int rtxSoapSendHttp (
            OSSOAPCONN * pSoapConn,
            const OSUTF8CHAR * soapMsg )
```

This function sends a complete HTTP POST request to a SOAP connection.

The request is stored in the XBinder context buffer. If an XML encode operation was just completed, then calling this function will send the encoded XML message to the SOAP endpoint.

Parameters

pSoapConn	- Pointer to SOAP connection structure.
soapMsg	- SOAP XML message to be sent.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.7 rtxSoapSendHttpResponse()

```
EXTERNRT int rtxSoapSendHttpResponse (
            OSSOAPCONN * pSoapConn,
            const OSUTF8CHAR * soapMsg )
```

This function sends a SOAP message as an HTTP response.

The SOAP message may be held in pSoapConn's context buffer.

Parameters

pSoapConn	- Pointer to SOAP connection structure.
soapMsg	- SOAP XML message to be sent.

Returns

- Operation status: 0 if success, negative code if error.

7.50.2.8 rtxSoapSetReadTimeout()

```
EXTERNRT int rtxSoapSetReadTimeout (
            OSSOAPCONN * pSoapConn,
            OSUINT32 nsecs )
```

This function sets the read timeout value to the given number of seconds.

Any read operation attempted on the SOAP connection 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.

7.51 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"
```

Typedefs

- typedef int OSRTSOCKET

  Socket handle type definition.
- typedef unsigned long OSIPADDR

  The IP address represented as unsigned long value.

Functions

- EXTERNRT int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET ∗pNewSocket, OSIPADDR ∗destAddr, int ∗destPort)

  This function permits an incoming connection attempt on a socket.
- EXTERNRT int rtxSocketAddrToStr (OSIPADDR ipAddr, char ∗pbuf, size_t bufsize)

  This function converts an IP address to its string representation.
- EXTERNRT int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)

  This function associates a local address with a socket.
- EXTERNRT int rtxSocketClose (OSRTSOCKET socket)

  This function closes an existing socket.
- EXTERNRT int rtxSocketConnect (OSRTSOCKET socket, const char ∗host, int port)

  This function establishes a connection to a specified socket.
- EXTERNRT int rtxSocketConnectTimed (OSRTSOCKET socket, const char ∗host, int port, int nsecs)

  This function establishes a connection to a specified socket.
- EXTERNRT int rtxSocketCreate (OSRTSOCKET ∗psocket)

  This function creates a TCP socket.
- EXTERNRT int rtxSocketGetHost (const char ∗host, struct in_addr ∗inaddr)

  This function resolves the given host name to an IP address.
- EXTERNRT int rtxSocketsInit (OSVOIDARG)

  This function initiates use of sockets by an application.
- EXTERNRT int rtxSocketListen (OSRTSOCKET socket, int maxConnection)

  This function places a socket a state where it is listening for an incoming connection.
- EXTERNRT 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.
- EXTERNRT int rtxSocketRecv (OSRTSOCKET socket, OSOCTET ∗pbuf, size_t bufsize)

  This function receives data from a connected socket.
- EXTERNRT int rtxSocketRecvTimed (OSRTSOCKET socket, OSOCTET ∗pbuf, size_t bufsize, OSUINT32 secs)

  This function receives data from a connected socket on a timed basis.
- EXTERNRT 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.
- EXTERNRT int rtxSocketSend (OSRTSOCKET socket, const OSOCTET ∗pdata, size_t size)

  This function sends data on a connected socket.
- EXTERNRT int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)

  This function turns blocking mode for a socket on or off.
- EXTERNRT int rtxSocketStrToAddr (const char ∗pIPAddrStr, OSIPADDR ∗pIPAddr)

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

7.52 rtxStream.h File Reference

Input/output data stream type definitions and function prototypes.

```
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxMemBuf.h"
```

Classes

- struct OSRTSTREAM

  The stream control block.

Macros

- #define rtxStreamBlockingRead rtxStreamRead

  rtxStreamBlockingRead is deprecated.

Typedefs

- typedef long(∗ OSRTStreamReadProc) (struct OSRTSTREAM ∗pStream, OSOCTET ∗pbuffer, size_t nocts)

  Stream read function pointer type.
- typedef OSRTStreamReadProc OSRTStreamBlockingReadProc

  OSRTStreamBlockingReadProc is deprecated.
- typedef long(∗ OSRTStreamWriteProc) (struct OSRTSTREAM ∗pStream, const OSOCTET ∗data, size_t numocts)

  Stream write function pointer type.
- typedef int(∗ OSRTStreamFlushProc) (struct OSRTSTREAM ∗pStream)

  Stream flush function pointer type.
- typedef int(∗ OSRTStreamCloseProc) (struct OSRTSTREAM ∗pStream)

  Stream close function pointer type.
- typedef int(∗ OSRTStreamSkipProc) (struct OSRTSTREAM ∗pStream, size_t skipBytes)

  Stream skip function pointer type.
- typedef int(∗ OSRTStreamMarkProc) (struct OSRTSTREAM ∗pStream, size_t readAheadLimit)

  Stream mark function pointer type.
- typedef int(∗ OSRTStreamResetProc) (struct OSRTSTREAM ∗pStream)

  Stream reset function pointer type.
- typedef int(∗ OSRTStreamGetPosProc) (struct OSRTSTREAM ∗pStream, size_t ∗ppos)

  Stream get position function pointer type.
- typedef int(∗ OSRTStreamSetPosProc) (struct OSRTSTREAM ∗pStream, size_t pos)

  Stream set position function pointer type.
- typedef struct OSRTSTREAM OSRTSTREAM

  The stream control block.

Functions

- EXTERNRT int rtxStreamClose (OSCTXT ∗pctxt)

  This function closes the input or output stream and releases any system resources associated with the stream.
- EXTERNRT int rtxStreamFlush (OSCTXT ∗pctxt)

  This function flushes the output stream and forces any buffered output octets to be written out.
- EXTERNRT int rtxStreamLoadInputBuffer (OSCTXT ∗pctxt, OSSIZE nbytes)

  This is for meant for internal use by the runtime.
- EXTERNRT int rtxStreamInit (OSCTXT ∗pctxt)

This function initializes a stream part of the context block.

- EXTERNRT int rtxStreamInitCtxtBuf (OSCTXT ∗pctxt)

  This function initializes a stream to use the context memory buffer for stream buffering.
- EXTERNRT int rtxStreamRemoveCtxtBuf (OSCTXT ∗pctxt)

  This function removes the use of a context memory buffer from a stream.
- EXTERNRT long rtxStreamRead (OSCTXT ∗pctxt, OSOCTET ∗pbuffer, size_t nocts)

  Read up to nocts bytes of data from the input stream into an array of octets.
- EXTERNRT long rtxStreamRead2 (OSCTXT ∗pctxt, OSOCTET ∗pbuffer, size_t nocts, size_t bufSize)

  Attempt to read nocts bytes from the input stream into an array of octets.
- EXTERNRT int rtxStreamSkip (OSCTXT ∗pctxt, size_t skipBytes)

  This function skips over and discards the specified amount of data octets from this input stream.
- EXTERNRT 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.
- EXTERNRT int rtxStreamGetIOBytes (OSCTXT ∗pctxt, size_t ∗pPos)

  This function returns the number of processed octets.
- EXTERNRT int rtxStreamMark (OSCTXT ∗pctxt, size_t readAheadLimit)

  Marks the current position in this input stream.
- EXTERNRT int rtxStreamReset (OSCTXT ∗pctxt)

  Repositions this stream to the position recorded by the last call to the rtxStreamMark function.
- EXTERNRT OSBOOL rtxStreamMarkSupported (OSCTXT ∗pctxt)

  Tests if this input stream supports the mark and reset methods.
- EXTERNRT OSBOOL rtxStreamIsOpened (OSCTXT ∗pctxt)

  Tests if this stream opened (for reading or writing).
- EXTERNRT OSBOOL rtxStreamIsReadable (OSCTXT ∗pctxt)

  Tests if this stream opened for reading.
- EXTERNRT OSBOOL rtxStreamIsWritable (OSCTXT ∗pctxt)

  Tests if this stream opened for writing.
- EXTERNRT int rtxStreamRelease (OSCTXT ∗pctxt)

  This function releases the stream's resources.
- EXTERNRT void rtxStreamSetCapture (OSCTXT ∗pctxt, OSRTMEMBUF ∗pmembuf)

  This function sets a capture buffer for the stream.
- EXTERNRT OSRTMEMBUF ∗ rtxStreamGetCapture (OSCTXT ∗pctxt)

  This function returns the capture buffer currently assigned to the stream.
- EXTERNRT int rtxStreamGetPos (OSCTXT ∗pctxt, size_t ∗ppos)

  Get the current position in input stream.
- EXTERNRT int rtxStreamSetPos (OSCTXT ∗pctxt, size_t pos)

  Set the current position in input stream.

7.52.1 Detailed Description

Input/output data stream type definitions and function prototypes.

7.53 rtxStreamBuffered.h File Reference

```
#include "rtxsrc/rtxStream.h"
```

7.54 rtxStreamFile.h File Reference

```
#include <stdio.h>
#include "rtxsrc/rtxStream.h"
```

Functions

- EXTERNRT int rtxStreamFileAttach (OSCTXT *pctxt, FILE *pFile, OSUINT16 flags)

  Attaches the existing file structure pointer to the stream.
- EXTERNRT int rtxStreamFileOpen (OSCTXT *pctxt, const char *pFilename, OSUINT16 flags)

  Opens a file stream.
- EXTERNRT int rtxStreamFileCreateReader (OSCTXT *pctxt, const char *pFilename)

  This function creates an input file stream using the specified file name.
- EXTERNRT int rtxStreamFileCreateWriter (OSCTXT *pctxt, const char *pFilename)

  This function creates an output file stream using the file name.

7.55 rtxStreamHexText.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxStream.h"
```

Functions

- EXTERNRT int rtxStreamHexTextAttach (OSCTXT *pctxt, OSUINT16 flags)

  This function initializes a hexText stream and attaches it to the existing stream defined within the context.

7.55.1 Function Documentation

7.55.1.1 rtxStreamHexTextAttach()

```
EXTERNRT 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.

7.56 rtxStreamMemory.h File Reference

```
#include "rtxsrc/rtxStream.h"
```

Functions

- EXTERNRT int rtxStreamMemoryCreate (OSCTXT ∗pctxt, OSUINT16 flags)

  Opens a memory stream.
- EXTERNRT int rtxStreamMemoryAttach (OSCTXT ∗pctxt, OSOCTET ∗pMemBuf, size_t bufSize, OSUINT16 flags)

  Opens a memory stream using the specified memory buffer.
- EXTERNRT OSOCTET ∗ rtxStreamMemoryGetBuffer (OSCTXT ∗pctxt, size_t ∗pSize)

  This function returns the memory buffer and its size for the given memory stream.
- EXTERNRT int rtxStreamMemoryCreateReader (OSCTXT ∗pctxt, OSOCTET ∗pMemBuf, size_t bufSize)

  This function creates an input memory stream using the specified buffer.
- EXTERNRT int rtxStreamMemoryCreateWriter (OSCTXT ∗pctxt, OSOCTET ∗pMemBuf, size_t bufSize)

  This function creates an output memory stream using the specified buffer.
- EXTERNRT int rtxStreamMemoryResetWriter (OSCTXT ∗pctxt)

  This function resets the output memory stream internal buffer to allow it to be overwritten with new data.

7.57 rtxStreamSocket.h File Reference

```
#include "rtxsrc/rtxStream.h"
#include "rtxsrc/rtxSocket.h"
```

Functions

- EXTERNRT int rtxStreamSocketAttach (OSCTXT ∗pctxt, OSRTSOCKET socket, OSUINT16 flags)

  Attaches the existing socket handle to the stream.
- EXTERNRT int rtxStreamSocketClose (OSCTXT ∗pctxt)

  This function closes a socket stream.
- EXTERNRT int rtxStreamSocketCreateWriter (OSCTXT ∗pctxt, const char ∗host, int port)

  This function opens a socket stream for writing.
- EXTERNRT int rtxStreamSocketSetOwnership (OSCTXT ∗pctxt, OSBOOL ownSocket)

  This function transfers ownership of the socket to or from the stream instance.
- EXTERNRT int rtxStreamSocketSetReadTimeout (OSCTXT ∗pctxt, OSUINT32 nsecs)

  This function sets the read timeout value to the given number of seconds.

7.58 rtxSysInfo.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
```

Functions

- EXTERNRT int rtxGetPID ()

  This function return the process ID of the currently running process.
- EXTERNRT char ∗ rtxEnvVarDup (const char ∗name)

  This function make a duplicate copy of an environment variable.
- EXTERNRT OSBOOL rtxEnvVarIsSet (const char ∗name)

  This function tests if an environment variable is set.
- EXTERNRT int rtxEnvVarSet (const char ∗name, const char ∗value, int overwrite)

  This function sets an environment variable to the given value.

7.58.1 Function Documentation

7.58.1.1 rtxEnvVarDup()

```
EXTERNRT char∗ rtxEnvVarDup (
             const char ∗ name )
```

This function make a duplicate copy of an environment variable.

The variable should be used using the standard C RTL free function (note: the OSCRTLFREE macro may be used to abstract the free function).

Parameters

name	Name of environment variable to duplicate.

Returns

The duplicated environment variable value.

7.58.1.2 rtxEnvVarIsSet()

```
EXTERNRT OSBOOL rtxEnvVarIsSet (
             const char * name )
```

This function tests if an environment variable is set.

Parameters

name	Name of environment variable to test.

Returns

True if environmental variable is set; false otherwise.

7.58.1.3 rtxEnvVarSet()

```
EXTERNRT int rtxEnvVarSet (
             const char * name,
             const char * value,
             int overwrite )
```

This function sets an environment variable to the given value.

Parameters

name	Name of environment variable to test.
value	Value to which variable should be set.
overwrite	If non-zero, overwrite existing variable with value.

Returns

Status of operation, 0 = success.

7.58.1.4 rtxGetPID()

```
EXTERNRT int rtxGetPID ( )
```

This function return the process ID of the currently running process.

Returns

Process ID of currently running process.

7.59 rtxTBCD.h File Reference

Telephony binary-decimal conversion functions.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxQ825TBCDToString (OSSIZE numocts, const OSOCTET ∗data, char ∗buffer, OSSIZE bufsiz)

  This function converts a Q.825 TBCD value to a standard null-terminated string.
- EXTERNRT int rtxDecQ825TBCDString (OSCTXT ∗pctxt, OSSIZE numocts, char ∗buffer, OSSIZE bufsiz)

  This function decodes a Q.825 TBCD value to a standard null-terminated string.
- EXTERNRT int rtxEncQ825TBCDString (OSCTXT ∗pctxt, const char ∗str)

  This function encodes a null-terminated string Q.825 TBCD string.
- EXTERNRT int rtxTBCDBinToChar (OSUINT8 bcdDigit, char ∗pdigit)

  This function converts a TBCD binary character into its ASCII equivalent.
- EXTERNRT int rtxTBCDCharToBin (char digit, OSUINT8 ∗pbyte)

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

7.59.1 Detailed Description

Telephony binary-decimal conversion functions.

7.60 rtxText.h File Reference

```
#include "rtxContext.h"
```

Functions

- EXTERNRT int rtxTxtMatchChar (OSCTXT ∗pctxt, OSUTF8CHAR ch, OSBOOL skipWs)

  This function matches the given character or logs and returns an error.
- EXTERNRT int rtxTxtMatchChars (OSCTXT ∗pctxt, const OSUTF8CHAR ∗chars, OSBOOL skipWs)

  This function matches the given characters or logs and returns an error.
- EXTERNRT int rtxTxtPeekChar (OSCTXT ∗pctxt, OSUTF8CHAR ∗pch, OSBOOL skipWs)

  This function determines the next character in the input.
- EXTERNRT char rtxTxtPeekChar2 (OSCTXT ∗pctxt, OSBOOL skipWs)

  This function determines the next character in the input.
- EXTERNRT int rtxTxtSkipWhitespace (OSCTXT ∗pctxt)

  This function skips any whitespace in the input.
- EXTERNRT int rtxTxtReadBigInt (OSCTXT ∗pctxt, char ∗∗ppvalue)

  This function reads an integer, using standard decimal notation, into a character string.
- EXTERNRT int rtxTxtReadInt8 (OSCTXT ∗pctxt, OSINT8 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 8-bit signed integer.
- EXTERNRT int rtxTxtReadInt16 (OSCTXT ∗pctxt, OSINT16 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 16-bit signed integer.
- EXTERNRT int rtxTxtReadInt32 (OSCTXT ∗pctxt, OSINT32 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 32-bit signed integer.
- EXTERNRT int rtxTxtReadInt64 (OSCTXT ∗pctxt, OSINT64 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 64-bit signed integer.
- EXTERNRT int rtxTxtReadUInt8 (OSCTXT ∗pctxt, OSUINT8 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 8-bit unsigned integer.
- EXTERNRT int rtxTxtReadUInt16 (OSCTXT ∗pctxt, OSUINT16 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 16-bit unsigned integer.
- EXTERNRT int rtxTxtReadUInt32 (OSCTXT ∗pctxt, OSUINT32 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 32-bit unsigned integer.
- EXTERNRT int rtxTxtReadUInt64 (OSCTXT ∗pctxt, OSUINT64 ∗pvalue)

  This function reads an integer, using standard decimal notation, into an 64-bit unsigned integer.
- EXTERNRT int rtxTxtWriteInt (OSCTXT ∗pctxt, OSINT32 value)

  This writes a 32-bit signed integer to the buffer using standard decimal (base 10) notation.
- EXTERNRT int rtxTxtWriteInt64 (OSCTXT ∗pctxt, OSINT64 value)

  This writes a 64-bit signed integer to the buffer using standard decimal (base 10) notation.
- EXTERNRT int rtxTxtWriteUInt (OSCTXT ∗pctxt, OSUINT32 value)

  This writes a 32-bit unsigned integer to the buffer using standard decimal (base 10) notation.
- EXTERNRT int rtxTxtWriteUInt64 (OSCTXT ∗pctxt, OSUINT64 value)

  This writes a 64-bit unsigned integer to the buffer using standard decimal (base 10) notation.

7.61 rtxUnicode.h File Reference

This is an open source header file derived from the libxml2 project.

```
#include <stdio.h>
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT long rtxUCSToUTF8 (OSCTXT ∗pctxt, const OSUNICHAR ∗inbuf, size_t inlen, OSOCTET ∗outbuf, size_t outbufsiz)

    This function converts a Unicode string into a UTF-8 string.
- EXTERNRT const OSUTF8CHAR ∗ rtxUCSToDynUTF8 (OSCTXT ∗pctxt, const OSUNICHAR ∗inbuf)

    This function converts a null-terminated Unicode string into a UTF-8 string.
- EXTERNRT OSBOOL rtxUCSIsChar (OS32BITCHAR c)

    rtxUCSIsChar:
- EXTERNRT OSBOOL rtxUCSIsBlank (OS32BITCHAR c)

    rtxUCSIsBlank:
- EXTERNRT OSBOOL rtxUCSIsBaseChar (OS32BITCHAR c)

    rtxUCSIsBaseChar:
- EXTERNRT OSBOOL rtxUCSIsDigit (OS32BITCHAR c)

    rtxUCSIsDigit:
- EXTERNRT OSBOOL rtxUCSIsCombining (OS32BITCHAR c)

    rtxUCSIsCombining:
- EXTERNRT OSBOOL rtxUCSIsExtender (OS32BITCHAR c)

    rtxUCSIsExtender:
- EXTERNRT OSBOOL rtxUCSIsIdeographic (OS32BITCHAR c)

    rtxUCSIsIdeographic:
- EXTERNRT OSBOOL rtxUCSIsLetter (OS32BITCHAR c)

    rtxUCSIsLetter:
- EXTERNRT OSBOOL rtxUCSIsPubidChar (OS32BITCHAR c)

    rtxUCSIsPubidChar:
- EXTERNRT OSBOOL rtxErrAddUniStrParm (OSCTXT ∗pctxt, const OSUNICHAR ∗pErrParm)

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

7.61.1 Detailed Description

This is an open source header file derived from the libxml2 project.

It defines UNICODE data types and macros. See the header file for further details.

7.61.2 Function Documentation

7.61.2.1 rtxErrAddUniStrParm()

```
EXTERNRT OSBOOL rtxErrAddUniStrParm (
            OSCTXT ∗ pctxt,
            const OSUNICHAR ∗ pErrParm )
```

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

Parameters

pctxt	A pointer to a context structure.
pErrParm	The Unicode string error parameter.

Returns

The status of the operation (TRUE if the parameter was sucessfully added).

7.61.2.2 rtxUCSIsBaseChar()

```
EXTERNRT OSBOOL rtxUCSIsBaseChar (
            OS32BITCHAR c )
```

rtxUCSIsBaseChar:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [85] BaseChar ::= ... long list see REC ...

Returns 0 if not, non-zero otherwise

7.61.2.3 rtxUCSIsBlank()

```
EXTERNRT OSBOOL rtxUCSIsBlank (
            OS32BITCHAR c )
```

rtxUCSIsBlank:

Parameters

c	a UNICODE character (int)

Check whether the character is allowed by the production [3] S ::= (#x20 | #x9 | #xD | #xA)+ Also available as a macro IS_BLANK()

Returns 0 if not, non-zero otherwise

7.61.2.4 rtxUCSIsChar()

```
EXTERNRT OSBOOL rtxUCSIsChar (
            OS32BITCHAR c )
```

rtxUCSIsChar:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF] any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. Also available as a macro IS_CHAR()

Returns 0 if not, non-zero otherwise

7.61.2.5 rtxUCSIsCombining()

```
EXTERNRT OSBOOL rtxUCSIsCombining (
            OS32BITCHAR c )
```

rtxUCSIsCombining:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [87] CombiningChar ::= ... long list see REC ...

Returns 0 if not, non-zero otherwise

7.61.2.6 rtxUCSIsDigit()

```
EXTERNRT OSBOOL rtxUCSIsDigit (
            OS32BITCHAR c )
```

rtxUCSIsDigit:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [88] Digit ::= ... long list see REC ...

Returns 0 if not, non-zero otherwise

7.61.2.7 rtxUCSIsExtender()

```
EXTERNRT OSBOOL rtxUCSIsExtender (
            OS32BITCHAR c )
```

rtxUCSIsExtender:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [89] Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46 | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]

Returns 0 if not, non-zero otherwise

7.61.2.8 rtxUCSIsIdeographic()

```
EXTERNRT OSBOOL rtxUCSIsIdeographic (
            OS32BITCHAR c )
```

rtxUCSIsIdeographic:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

Returns 0 if not, non-zero otherwise

7.61.2.9 rtxUCSIsLetter()

```
EXTERNRT OSBOOL rtxUCSIsLetter (
            OS32BITCHAR c )
```

rtxUCSIsLetter:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [84] Letter ::= BaseChar | Ideographic

Returns 0 if not, non-zero otherwise

7.61.2.10 rtxUCSIsPubidChar()

```
EXTERNRT OSBOOL rtxUCSIsPubidChar (
            OS32BITCHAR c )
```

rtxUCSIsPubidChar:

Parameters

c	an unicode character (int)

Check whether the character is allowed by the production [13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./↩
:=?;!∗#@$_%]

Returns 0 if not, non-zero otherwise

7.61.2.11 rtxUCSToDynUTF8()

```
EXTERNRT const OSUTF8CHAR∗ rtxUCSToDynUTF8 (
            OSCTXT ∗ pctxt,
            const OSUNICHAR ∗ inbuf )
```

This function converts a null-terminated Unicode string into a UTF-8 string.

Memory is allocated for the output string using the built-in memory management functions.

Parameters

pctxt	Pointer to context structure.
inbuf	Null-terminated Unicode string to convert.

Returns

Converted UTF-8 character string.

7.61.2.12 rtxUCSToUTF8()

```
EXTERNRT long rtxUCSToUTF8 (
            OSCTXT ∗ pctxt,
            const OSUNICHAR ∗ inbuf,
            size_t inlen,
            OSOCTET ∗ outbuf,
            size_t outbufsiz )
```

This function converts a Unicode string into a UTF-8 string.

A buffer large enough to hold the converted UTF-8 characters must be provided. A buffer providing 4 bytes-per-character should be large enough to hold the largest possible UTF-8 conversion. The output UTF-8 string is null-terminated.

Parameters

pctxt	Pointer to context structure.
inbuf	Unicode string to convert. Does not need to be null-terminated.
inlen	Number of characters in inbuf.
outbuf	Buffer to hold converted string.
outbufsiz	Size of output buffer.

Returns

Total number of bytes in converted string or a negative status code if error.

7.62 rtxUTF16.h File Reference

Utility functions for converting UTF-16(LE|BE) strings to and from UTF-8.

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT int rtxUTF16LEToUTF8 (unsigned char ∗out, int outlen, const unsigned char ∗inb, int inlenb)

  This function converts a UTF-16LE string into a UTF-8 string.
- EXTERNRT int rtxUTF8ToUTF16LE (unsigned char ∗outb, int outlen, const unsigned char ∗in, int inlen)

  This function converts a UTF-8 string into a UTF-16LE string.
- EXTERNRT int rtxUTF8ToUTF16 (unsigned char ∗outb, int outlen, const unsigned char ∗in, int inlen)

  This function converts a UTF-8 string into a UTF-16 string.
- EXTERNRT int rtxUTF16BEToUTF8 (unsigned char ∗out, int outlen, const unsigned char ∗inb, int inlenb)

  This function converts a UTF-16BE string into a UTF-8 string.
- EXTERNRT int rtxUTF8ToUTF16BE (unsigned char ∗outb, int outlen, const unsigned char ∗in, int inlen)

  This function converts a UTF-8 string into a UTF-16BE string.
- EXTERNRT int rtxStreamUTF8ToUTF16 (OSCTXT ∗pctxt, const unsigned char ∗in, size_t inlen)

  This function takes a block of UTF-8 chars in and try to convert it to an UTF-16 block of chars, and write the converted chars to stream.
- EXTERNRT int rtxStreamUTF8ToUTF16LE (OSCTXT ∗pctxt, const unsigned char ∗in, size_t inlen)

  This function takes a block of UTF-8 chars in and try to convert it to an UTF-16LE block of chars, and write the converted chars to stream.
- EXTERNRT int rtxStreamUTF8ToUTF16BE (OSCTXT ∗pctxt, const unsigned char ∗in, size_t inlen)

  This function takes a block of UTF-8 chars in and try to convert it to an UTF-16BE block of chars, and write the converted chars to stream.

7.62.1 Detailed Description

Utility functions for converting UTF-16(LE|BE) strings to and from UTF-8.

7.62.2 Function Documentation

7.62.2.1 rtxStreamUTF8ToUTF16()

```
EXTERNRT int rtxStreamUTF8ToUTF16 (
            OSCTXT * pctxt,
            const unsigned char * in,
            size_t inlen )
```

This function takes a block of UTF-8 chars in and try to convert it to an UTF-16 block of chars, and write the converted chars to stream.

Parameters

pctxt	Pointer to context block structure.
in	A pointer to an array of UTF-8 chars.
inlen	The length of .

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.62.2.2 rtxStreamUTF8ToUTF16BE()

```
EXTERNRT int rtxStreamUTF8ToUTF16BE (
            OSCTXT * pctxt,
            const unsigned char * in,
            size_t inlen )
```

This function takes a block of UTF-8 chars in and try to convert it to an UTF-16BE block of chars, and write the converted chars to stream.

Parameters

pctxt	Pointer to context block structure.
in	A pointer to an array of UTF-8 chars.
inlen	The length of .

Returns

> Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.62.2.3 rtxStreamUTF8ToUTF16LE()

```
EXTERNRT int rtxStreamUTF8ToUTF16LE (
             OSCTXT * pctxt,
             const unsigned char * in,
             size_t inlen )
```

This function takes a block of UTF-8 chars in and try to convert it to an UTF-16LE block of chars, and write the converted chars to stream.

Parameters

pctxt	Pointer to context block structure.
in	A pointer to an array of UTF-8 chars.
inlen	The length of .

Returns

> Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.62.2.4 rtxUTF16BEToUTF8()

```
EXTERNRT int rtxUTF16BEToUTF8 (
             unsigned char * out,
             int outlen,
             const unsigned char * inb,
             int inlenb )
```

This function converts a UTF-16BE string into a UTF-8 string.

A buffer large enough to hold the converted UTF-8 characters must be provided. A buffer providing 4 bytes-per-character should be large enough to hold the largest possible UTF-8 conversion. This function assumes the endian property is the same between the native type of this machine and the inputed one.

Parameters

out	Buffer to hold converted string.
outlen	Size of output buffer.
inb	A pointer to an array of UTF-16LE passed as a byte array.
inlenb	The length of in UTF-16LE characters.

Returns

> Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails (if ∗inb is not a valid utf16 string)

7.62.2.5 rtxUTF16LEToUTF8()

```
EXTERNRT int rtxUTF16LEToUTF8 (
             unsigned char ∗ out,
             int outlen,
             const unsigned char ∗ inb,
             int inlenb )
```

This function converts a UTF-16LE string into a UTF-8 string.

A buffer large enough to hold the converted UTF-8 characters must be provided. A buffer providing 4 bytes-per-character should be large enough to hold the largest possible UTF-8 conversion. This function assumes the endian property is the same between the native type of this machine and the inputed one.

Parameters

out	Buffer to hold converted string.
outlen	Size of output buffer.
inb	A pointer to an array of UTF-16LE passed as a byte array.
inlenb	The length of in UTF-16LE characters.

Returns

> Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails (if ∗inb is not a valid utf16 string)

7.62.2.6 rtxUTF8ToUTF16()

```
EXTERNRT int rtxUTF8ToUTF16 (
             unsigned char ∗ outb,
             int outlen,
             const unsigned char ∗ in,
             int inlen )
```

This function converts a UTF-8 string into a UTF-16 string.

A buffer large enough to hold the converted UTF-16 characters must be provided.

Parameters

outb	Buffer to hold converted string.
outlen	Size of output buffer.
in	A pointer to an array of UTF-8 chars
inlen	The length of

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.62.2.7 rtxUTF8ToUTF16BE()

```
EXTERNRT int rtxUTF8ToUTF16BE (
             unsigned char * outb,
             int outlen,
             const unsigned char * in,
             int inlen )
```

This function converts a UTF-8 string into a UTF-16BE string.

A buffer large enough to hold the converted UTF-16 characters must be provided.

Parameters

outb	Buffer to hold converted string.
outlen	Size of output buffer.
in	A pointer to an array of UTF-8 chars
inlen	The length of

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.62.2.8 rtxUTF8ToUTF16LE()

```
EXTERNRT int rtxUTF8ToUTF16LE (
             unsigned char * outb,
             int outlen,
             const unsigned char * in,
             int inlen )
```

This function converts a UTF-8 string into a UTF-16LE string.

A buffer large enough to hold the converted UTF-16 characters must be provided.

Parameters

outb	Buffer to hold converted string.
outlen	Size of output buffer.
in	A pointer to an array of UTF-8 chars
inlen	The length of

Returns

Total number of bytes in converted string or a negative status code if error: -1 if lack of space, or -2 if the transcoding fails

7.63 rtxUTF8.h File Reference

Utility functions for handling UTF-8 strings.

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR∗)lstr)

  Compare UTF-8 string to a string literal.

Functions

- EXTERNRT 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).
- EXTERNRT long rtxUTF8ToUnicode32 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗inbuf, OS32BITCHAR ∗outbuf, size_t outbufsiz)

  This function converts a UTF-8 string to a Unicode string (UTF-32).
- EXTERNRT int rtxValidateUTF8 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗inbuf)

  This function will validate a UTF-8 encoded string to ensure that it is encoded correctly.
- EXTERNRT size_t rtxUTF8Len (const OSUTF8CHAR ∗inbuf)

  This function will return the length (in characters) of a null-terminated UTF-8 encoded string.
- EXTERNRT size_t rtxUTF8LenBytes (const OSUTF8CHAR ∗inbuf)

  This function will return the length (in bytes) of a null-terminated UTF-8 encoded string.

- EXTERNRT 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.
- EXTERNRT int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET ∗buf, size_t bufsiz)

  This function will convert a wide character into an encoded UTF-8 character byte string.
- EXTERNRT 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.
- EXTERNRT OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR ∗buf, OSUINT32 ∗len)

  Thia function will convert a UTF-8 encoded character value into a wide character.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8StrChr (OSUTF8CHAR ∗utf8str, OS32BITCHAR utf8char)

  This function finds a character in the given UTF-8 character string.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8Strdup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str)

  This function creates a duplicate copy of the given UTF-8 character string.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8Strndup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, size_t nbytes)

  This function creates a duplicate copy of the given UTF-8 character string.
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8StrRefOrDup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str)

  This function check to see if the given UTF8 string pointer exists on the memory heap.
- EXTERNRT OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR ∗utf8str1, const OSUTF8CHAR ∗utf8str2)

  This function compares two UTF-8 string values for equality.
- EXTERNRT OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR ∗utf8str1, const OSUTF8CHAR ∗utf8str2, size_t count)

  This function compares two UTF-8 string values for equality.
- EXTERNRT 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).
- EXTERNRT 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).
- EXTERNRT OSUTF8CHAR ∗ rtxUTF8Strcpy (OSUTF8CHAR ∗dest, size_t bufsiz, const OSUTF8CHAR ∗src)

  This function copies a null-terminated UTF-8 string to a target buffer.
- EXTERNRT 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.
- EXTERNRT OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR ∗str)

  This function computes a hash code for the given string value.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8StrToBool (const OSUTF8CHAR ∗utf8str, OSBOOL ∗pvalue)

  This function converts the given null-terminated UTF-8 string to a boolean (true/false) value.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8StrnToDouble (const OSUTF8CHAR ∗utf8str, size_t nbytes, OSREAL ∗pvalue)

  This function converts the given part of UTF-8 string to a double value.
- EXTERNRT int rtxUTF8StrToInt (const OSUTF8CHAR ∗utf8str, OSINT32 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to an integer value.
- EXTERNRT int rtxUTF8StrnToInt (const OSUTF8CHAR ∗utf8str, size_t nbytes, OSINT32 ∗pvalue)

  This function converts the given part of UTF-8 string to an integer value.

- EXTERNRT int rtxUTF8StrToUInt (const OSUTF8CHAR ∗utf8str, OSUINT32 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to an unsigned integer value.
- EXTERNRT 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.
- EXTERNRT 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).
- EXTERNRT 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).
- EXTERNRT int rtxUTF8StrToInt64 (const OSUTF8CHAR ∗utf8str, OSINT64 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to a 64-bit integer value.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8StrToUInt64 (const OSUTF8CHAR ∗utf8str, OSUINT64 ∗pvalue)

  This function converts the given null-terminated UTF-8 string to an unsigned 64-bit integer value.
- EXTERNRT 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.
- EXTERNRT int rtxUTF8ToDynUniStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, const OSUNICHAR ∗∗ppdata, OSUINT32 ∗pnchars)

  This function converts the given UTF-8 string to a Unicode string (UTF-16).
- EXTERNRT int rtxUTF8ToDynUniStr32 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, const OS32BITCHAR ∗∗ppdata, OSUINT32 ∗pnchars)

  This function converts the given UTF-8 string to a Unicode string (UTF-32).
- EXTERNRT int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR ∗utf8instr, size_t nbytes, const OSUTF8CHAR ∗∗putf8outstr)

  This function removes leading and trailing whitespace from a string.
- EXTERNRT int rtxUTF8StrToDynHexStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, OSDynOctStr ∗pvalue)

  This function converts the given null-terminated UTF-8 string to a octet string value.
- EXTERNRT 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.
- EXTERNRT 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.
- EXTERNRT const OSUTF8CHAR ∗ rtxUTF8StrNextTok (OSUTF8CHAR ∗utf8str, OSUTF8CHAR ∗∗ppNext)

  This function returns the next whitespace-separated token from the input string.

7.63.1 Detailed Description

Utility functions for handling UTF-8 strings.

7.64 rtxXmlQName.h File Reference

XML QName type definition and associated utility functions.

```
#include "rtxsrc/rtxContext.h"
```

Classes

- struct OSXMLFullQName

    This version of QName contains complete namespace info (prefix + URI)
- struct OSXMLQNameValue

    This version of QName models the value space for XML Schema QNames, which consists of just a namespace URI and a local name.

Functions

- EXTERNRT OSXMLFullQName ∗ rtxNewFullQName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗localName, const OSUTF8CHAR ∗prefix, const OSUTF8CHAR ∗nsuri)

    This function creates a new full QName structure given the parts.
- EXTERNRT OSXMLFullQName ∗ rtxNewFullQNameDeepCopy (OSCTXT ∗pctxt, const OSXMLFullQName ∗pqname)

    This function allocates a new QName instance and makes a deep copy of the given QName including the strings inside.
- EXTERNRT void rtxQNameDeepCopy (OSCTXT ∗pctxt, OSXMLFullQName ∗pdest, const OSXMLFullQName ∗psrc)

    This function makes a deep copy of the given QName including the strings inside.
- EXTERNRT void rtxQNameFreeMem (OSCTXT ∗pctxt, OSXMLFullQName ∗pqname, OSBOOL dynamic)

    This function frees all memory within a QName structure,.
- EXTERNRT OSUINT32 rtxQNameHash (const OSXMLFullQName ∗pqname)

    This function computes a hash code for the given QName.
- EXTERNRT OSBOOL rtxQNamesEqual (const OSXMLFullQName ∗pqname1, const OSXMLFullQName ∗pqname2)

    This function tests 2 QNames for equality.
- EXTERNRT const OSUTF8CHAR ∗ rtxQNameToString (const OSXMLFullQName ∗pqname, OSUTF8CHAR ∗buffer, OSUINT32 bufsiz)

    This function returns the QName in the following stringified format: {uri}/localName.
- EXTERNRT OSSIZE rtxQNameValueArraySearch (OSXMLQNameValue ∗pValue, OSXMLQNameValue ∗pArray, OSSIZE arraySize)

    Function to do a binary search on a sorted array of QNames.

7.64.1 Detailed Description

XML QName type definition and associated utility functions.

7.64.2 Function Documentation

7.64.2.1 rtxNewFullQName()

```
EXTERNRT OSXMLFullQName* rtxNewFullQName (
            OSCTXT * pctxt,
            const OSUTF8CHAR * localName,
            const OSUTF8CHAR * prefix,
            const OSUTF8CHAR * nsuri )
```

This function creates a new full QName structure given the parts.

Memory is allocated for the structure using rtxMemAlloc. Copies are not made of the string variables - the pointers are stored.

Parameters

pctxt	Pointer to a context structure.
localName	Element local name.
prefix	Namespace prefix.
nsuri	Namespace URI.

Returns

QName value. Memory for the value will have been allocated by rtxMemAlloc and thus must be freed using one of the rtxMemFree functions. The value will be NULL if no dynamic memory was available.

7.64.2.2 rtxNewFullQNameDeepCopy()

```
EXTERNRT OSXMLFullQName* rtxNewFullQNameDeepCopy (
            OSCTXT * pctxt,
            const OSXMLFullQName * pqname )
```

This function allocates a new QName instance and makes a deep copy of the given QName including the strings inside.

Parameters

pctxt	Pointer to a context structure.
pqname	Pointer to QName to be copied.

Returns

Deep copy of QName structure.

7.64.2.3 rtxQNameDeepCopy()

```
EXTERNRT void rtxQNameDeepCopy (
             OSCTXT * pctxt,
             OSXMLFullQName * pdest,
             const OSXMLFullQName * psrc )
```

This function makes a deep copy of the given QName including the strings inside.

Parameters

pctxt	Pointer to a context structure.
pdest	Pointer to QName to receive copied data.
psrc	Pointer to QName to be copied.

7.64.2.4 rtxQNameFreeMem()

```
EXTERNRT void rtxQNameFreeMem (
             OSCTXT * pctxt,
             OSXMLFullQName * pqname,
             OSBOOL dynamic )
```

This function frees all memory within a QName structure,.

Parameters

pctxt	Pointer to a context structure.
pqname	Pointer to QName in which memory will be freed.
dynamic	Boolean indicating if pqname is dynamic. If true, the memory for pqname is freed.

7.64.2.5 rtxQNameHash()

```
EXTERNRT OSUINT32 rtxQNameHash (
             const OSXMLFullQName * pqname )
```

This function computes a hash code for the given QName.

Parameters

pqname	Pointer to QName structure.

Returns

Computed hash code.

7.64.2.6 rtxQNamesEqual()

```
EXTERNRT OSBOOL rtxQNamesEqual (
             const OSXMLFullQName * pqname1,
             const OSXMLFullQName * pqname2 )
```

This function tests 2 QNames for equality.

Parameters

pqname1	Pointer to QName structure.
pqname2	Pointer to QName structure.

Returns

True if names equal; false, otherwise.

7.64.2.7 rtxQNameToString()

```
EXTERNRT const OSUTF8CHAR* rtxQNameToString (
             const OSXMLFullQName * pqname,
             OSUTF8CHAR * buffer,
             OSUINT32 bufsiz )
```

This function returns the QName in the following stringified format: {uri}/localName.

Parameters

pqname	Pointer to QName structure.
buffer	Buffer into which to return name.
bufsiz	Size of buffer into which name is to be returned. If name will not fit in buffer, it is truncated.

Returns

Pointer to string (address of 'buffer' argument).

7.64.2.8 rtxQNameValueArraySearch()

```
EXTERNRT OSSIZE rtxQNameValueArraySearch (
             OSXMLQNameValue * pValue,
             OSXMLQNameValue * pArray,
             OSSIZE arraySize )
```

Function to do a binary search on a sorted array of QNames.

Parameters

pValue	Value to search for.
pArray	Array to search. It must be sorted by localName then nsURI, with null nsURI (no namespace) being less than non-null nsURI (non empty namespace).
arraySize	Size of the array

Returns

index of matching element. If multiple elements match pValue, the index of the smallest such element is returned. If there is no match, (OSSIZE)-1 is returned.

7.65 rtxXmlStr.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- EXTERNRT OSXMLSTRING ∗ rtxCreateXmlStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗pStr, OSBOOL cdata)

  This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.
- EXTERNRT OSXMLSTRING ∗ rtxCreateCopyXmlStr (OSCTXT ∗pctxt, const OSUTF8CHAR ∗pStr, OSBOOL cdata)

  This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

7.65.1 Function Documentation

7.65.1.1 rtxCreateCopyXmlStr()

```
EXTERNRT OSXMLSTRING* rtxCreateCopyXmlStr (
            OSCTXT * pctxt,
            const OSUTF8CHAR * pStr,
            OSBOOL cdata )
```

This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

In contrary to `rtxCreateXmlStr` function, the string value is copied. This function uses `rtxMemAlloc` to allocate the memory for the OSXMLSTRING structure and for the string value being copied. To free memory, `rtxMemFreePtr` function may be used for both value and structure itself:

OSXMLSTRING pStr = rtxCreateCopyXmlStr (....);*

....

rtxMemFreePtr (pctxt, pStr->value);

rtxMemFreePtr (pctxt, pStr);

Parameters

pctxt	Pointer to a context block
pStr	Pointer to a character string to be copied.
cdata	This indicates if this string should be encoded as a CDATA section in an XML document.

Returns

The allocated and initialized instance of OSXMLSTRING type.

7.65.1.2 rtxCreateXmlStr()

```
EXTERNRT OSXMLSTRING* rtxCreateXmlStr (
            OSCTXT * pctxt,
            const OSUTF8CHAR * pStr,
            OSBOOL cdata )
```

This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

This function uses `rtxMemAlloc` to allocate the memory for the OSXMLSTRING structure. String `pStr` is not copied: the pointer will be assigned to "value" member of OSXMLSTRING structure. To free memory, `rtxMemFreePtr` function may be used:

OSXMLSTRING pStr = rtxCreateXmlStr (....);*

....

rtxMemFreePtr (pctxt, pStr);

Note, user is responsible to free *pStr->value* if it is not static and was allocated dynamically.

Parameters

pctxt	Pointer to a context block
pStr	Pointer to a character string to be assigned.
cdata	This indicates if this string should be encoded as a CDATA section in an XML document.

Returns

The allocated and initialized instance of OSXMLSTRING type.

Index