XBinder C Common Runtime Functions

Version 3.0

Objective Systems, Inc.

December 2024

XBinder C Common Runtime Functions

Author's Contact Information. Comments, suggestions, and inquiries regarding XBinder or this document may be sent by electronic mail to `<info@obj-sys.com>`.

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 Documentation

Context Message Buffer Read/Write Functions

Detailed Description

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

Classes

- struct _OSRTBufLocDescr

Typedefs

- typedef struct _OSRTBufLocDescr OSRTBufLocDescr

Functions

- int rtxCheckBuffer (OSCTXT * pctxt, size_t nbytes)
- int rtxCheckOutputBuffer (OSCTXT * pctxt, OSSIZE nbytes)
- int rtxDecrBufferByteIndex (OSCTXT * pctxt, OSSIZE nbytes)
- OSBOOL rtxIsOutputBufferFlushable (OSCTXT * pctxt)
- int rtxFlushOutputBuffer (OSCTXT * pctxt)
- int rtxExpandOutputBuffer (OSCTXT * pctxt, size_t nbytes)
- int rtxCheckInputBuffer (OSCTXT * pctxt, size_t nbytes)
- int rtxCopyAsciiText (OSCTXT * pctxt, const char * text)
- int rtxCopyUTF8Text (OSCTXT * pctxt, const OSUTF8CHAR * text)
- int rtxCopyUnicodeText (OSCTXT * pctxt, const OSUNICHAR * text)
- int rtxLoadInputBuffer (OSCTXT * pctxt, OSSIZE nbytes)
- int rtxPeekByte (OSCTXT * pctxt, OSOCTET * pbyte)
- int rtxPeekBytes (OSCTXT * pctxt, OSOCTET * pdata, OSSIZE bufsize, OSSIZE nocts, OSSIZE * pactual)
- int rtxReadBytesSafe (OSCTXT * pctxt, OSOCTET * buffer, size_t bufsize, size_t nocts)
- int rtxReadBytes (OSCTXT * pctxt, OSOCTET * pdata, size_t nocts)
- int rtxReadBytesDynamic (OSCTXT * pctxt, OSOCTET ** ppdata, size_t nocts, OSBOOL * pMemAlloc)
- int rtxWriteBytes (OSCTXT * pctxt, const OSOCTET * pdata, size_t nocts)
- int rtxWriteIndent (OSCTXT * pctxt)

- void rtxIndentDecr (OSCTXT * pctxt)
- void rtxIndentIncr (OSCTXT * pctxt)
- void rtxIndentReset (OSCTXT * pctxt)
- size_t rtxGetIndentLevels (OSCTXT * pctxt)
- OSBOOL rtxCanonicalSort (OSOCTET * refPoint, OSRTSList * pList, OSBOOL normal)
- int rtxEncCanonicalSort (OSCTXT * pctxt, OSCTXT * pMemCtxt, OSRTSList * pList)
- void rtxGetBufLocDescr (OSCTXT * pctxt, OSRTBufLocDescr * pDescr)
- void rtxAddBufLocDescr (OSCTXT * pctxt, OSRTSList * pElemList, OSRTBufLocDescr * pDescr)

Macros

- #define OSRTPUTCHAR rtxWriteBytes (pctxt, (OSOCTET*)&ch, 1)
- #define OSRTPUTCHARREV (pctxt)->buffer.data[--(pctxt)->buffer.byteIndex]=(OSOCTET)ch;
- #define OSRTZTERM (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]=(OSOCTET)0;
- #define OSRTSAFEZTERM do { \ if (rtxCheckOutputBuffer (pctxt, 1) == 0) \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]=(OSOCTET)0; \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEPUTCHAR do { \ if (rtxCheckOutputBuffer (pctxt, 1) == 0) \ (pctxt)->lastChar= \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex++]=(OSOCTET)ch; \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEPUTCHAR_ZTERM do { \ if (rtxCheckOutputBuffer (pctxt, 2) == 0) { \ (pctxt)->lastChar= \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex++]=(OSOCTET)ch; \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]=(OSOCTET)0; } \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEPUTCHAR1 do { \ OSOCTET b = (OSOCTET)ch; \ rtxWriteBytes (pctxt, &b, 1); \ } while (0)
- #define OSRTMEMCPY do { \ OSCRTLSAFEMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], \ (pctxt)->buffer.size-(pctxt)->buffer.byteIndex, bdata, len); \ (pctxt)->buffer.byteIndex += len; \ (pctxt)->lastChar = (pctxt)->buffer.data[(pctxt)->buffer.byteIndex-1]; \ } while (0)
- #define OSRTMEMCPYREV do { \ (pctxt)->buffer.byteIndex -= len; \ OSCRTLSAFEMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], \ (pctxt)->buffer.size-(pctxt)->buffer.byteIndex, bdata, len); \ } while (0)
- #define OSRTSAFEMEMCPY do { \ if (rtxCheckOutputBuffer (pctxt, len) == 0) { \ OSCRTLMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], bdata, len); \ (pctxt)->buffer.byteIndex += len; \ (pctxt)->lastChar = (pctxt)->buffer.data[(pctxt)->buffer.byteIndex-1]; } \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEMEMCPY1 do { \ if (rtxCheckOutputBuffer (pctxt, len) == 0) { \ OSCRTLMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], bdata, len); \ (pctxt)->buffer.byteIndex += len; \ (pctxt)->lastChar = (pctxt)->buffer.data[(pctxt)->buffer.byteIndex-1]; \ stat = 0; } \ else stat = RTERR_BUFOVFLW; \ } while (0)
- #define OSRTGETBUFUTF8LEN rtxCalcUTF8Len (OSRTBUFPTR (pctxt), OSRTBUFSIZE (pctxt))
- #define OSRTCHKBUFUTF8LEN do { size_t nchars = OSRTGETBUFUTF8LEN (pctxt); \ stat = (nchars >= lower && nchars <= upper) ? 0 : RTERR_CONSVIO; } while(0)

- #define OSRTENDOFBUF ((pctxt)->buffer.byteIndex >= (pctxt)->buffer.size)
- #define OSRTByteAlign if ((pctxt)->buffer.bitOffset != 8) { \ (pctxt)->buffer.byteIndex++; \ (pctxt)->buffer.bitOffset = 8; } \

Typedef Documentation

typedef struct _OSRTBufLocDescr OSRTBufLocDescr

Buffer location descriptor

Function Documentation

int rtxCheckOutputBuffer (OSCTXT *pctxt, OSSIZE 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.

Table 2.1. Parameters

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

Returns: . 0 on success, or less than zero on failure.

int rtxDecrBufferByteIndex (OSCTXT *pctxt, OSSIZE nbytes)

This function decrements the buffer byte index by the specified amount. The index will not be decremented if an underflow would occur.

Table 2.2. Parameters

pctxt	Pointer to a context structure.

Returns: . 0 if the buffer index is descremented; error status if not.

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.

Table 2.3. Parameters

pctxt	Pointer to a context structure.

Returns: . TRUE if the buffer can be flushed; FALSE if not.

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.

Table 2.4. Parameters

pctxt	Pointer to a context structure.

Returns: . 0 on success, or less than zero on failure.

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.

Table 2.5. Parameters

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

Returns: . 0 on success, or less than zero on failure.

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.

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.

Table 2.6. 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.

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.

Table 2.7. 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.

int rtxPeekBytes (OSCTXT *pctxt, OSOCTET *pdata, OSSIZE buf-size, 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 pc-txt->buffer.byteIndex.

Table 2.8. 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.

int rtxReadBytesSafe (OSCTXT *pctxt, OSOCTET *buffer, size_t buf-size, 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.

Table 2.9. 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.

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

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

Table 2.10. 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.

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.

Table 2.11. 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.

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

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

Table 2.12. 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.

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.

Table 2.13. Parameters

pctxt	Pointer to context block structure.

Returns: . Completion status of operation:

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

void rtxIndentDecr (OSCTXT *pctxt)

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

See also: . rtxGetIndentLevels

Table 2.14. Parameters

pctxt	Pointer to context block structure.

void rtxIndentIncr (OSCTXT *pctxt)

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

See also: . rtxGetIndentLevels

Table 2.15. Parameters

pctxt	Pointer to context block structure.

void rtxIndentReset (OSCTXT *pctxt)

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

See also: . rtxGetIndentLevels

Table 2.16. Parameters

pctxt	Pointer to context block structure.

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.

Table 2.17. Parameters

pctxt	Pointer to context block structure.

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.

Table 2.18. 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).

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.

Table 2.19. 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.

void rtxGetBufLocDescr (OSCTXT *pctxt, OSRTBufLocDescr *pDescr)

Set the buffer location description's offset (pDescr->offset) to the current position in pCtxt's buffer.

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

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

Table 2.20. 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.

Character string functions

Detailed Description

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

Functions

- int rtxStricmp (const char * str1, const char * str2)
- int rtxStrnicmp (const char * str1, const char * str2, size_t count)
- char * rtxStrcat (char * dest, size_t bufsiz, const char * src)
- char * rtxStrncat (char * dest, size_t bufsiz, const char * src, size_t nchars)
- char * rtxStrcpy (char * dest, size_t bufsiz, const char * src)
- char * rtxStrncpy (char * dest, size_t bufsiz, const char * src, size_t nchars)
- char * rtxStrdup (OSCTXT * pctxt, const char * src)
- char * rtxStrndup (OSCTXT * pctxt, const char * src, OSSIZE nchars)
- const char * rtxStrJoin (char * dest, size_t bufsiz, const char * str1, const char * str2, const char * str3, const char * str4, const char * str5)
- char * rtxStrDynJoin (OSCTXT * pctxt, const char * str1, const char * str2, const char * str3, const char * str4, const char * str5)
- char * rtxStrTrimEnd (char * s)

- int rtxValidateConstrainedStr (OSCTXT * pctxt, const char * pvalue, const char * pCharSet)
- int rtxIntToCharStr (OSINT32 value, char * dest, size_t bufsiz, char padchar)
- int rtxUIntToCharStr (OSUINT32 value, char * dest, size_t bufsiz, char padchar)
- int rtxInt64ToCharStr (OSINT64 value, char * dest, size_t bufsiz, char padchar)
- int rtxUInt64ToCharStr (OSUINT64 value, char * dest, size_t bufsiz, char padchar)
- int rtxSizeToCharStr (size_t value, char * dest, size_t bufsiz, char padchar)
- int rtxHexCharsToBinCount (const char * hexstr, size_t nchars)
- int rtxHexCharsToBin (const char * hexstr, size_t nchars, OSOCTET * binbuf, size_t bufsize)
- int rtxCharStrToInt (const char * cstr, OSINT32 * pvalue)
- int rtxCharStrnToInt (const char * cstr, OSSIZE ndigits, OSINT32 * pvalue)
- int rtxCharStrToInt8 (const char * cstr, OSINT8 * pvalue)
- int rtxCharStrToInt16 (const char * cstr, OSINT16 * pvalue)
- int rtxCharStrToInt64 (const char * cstr, OSINT64 * pvalue)
- int rtxCharStrToUInt (const char * cstr, OSUINT32 * pvalue)
- int rtxCharStrToUInt8 (const char * cstr, OSUINT8 * pvalue)
- int rtxCharStrToUInt16 (const char * cstr, OSUINT16 * pvalue)
- int rtxCharStrToUInt64 (const char * cstr, OSUINT64 * pvalue)

Function Documentation

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.

Table 2.21. 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.

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.

Table 2.22. 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.

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.

Table 2.23. 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.

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.

Table 2.24. 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.

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.

Table 2.25. 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.

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.

Table 2.26. 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.

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.

Table 2.27. 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.

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.

Table 2.28. 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.

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.

Table 2.29. 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.

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.

Table 2.30. 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.

char* rtxStrTrimEnd (char *s)

This function trims whitespace from the end of a string.

Table 2.31. Parameters

s	Pointer to string to be trimmed.

Returns: . Point to string s.

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.

Table 2.32. 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.

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.

Table 2.33. 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.

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.

Table 2.34. 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.

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.

Table 2.35. 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.

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.

Table 2.36. 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.

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.

Table 2.37. 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.

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.

Table 2.38. 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.

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.

Table 2.39. 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.

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.

Table 2.40. 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.

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.

Table 2.41. 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.

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.

Table 2.42. 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.

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.

Table 2.43. 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.

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.

Table 2.44. 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.

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.

Table 2.45. 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.

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.

Table 2.46. 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.

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.

Table 2.47. 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.

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.

Table 2.48. 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.

Date/time conversion functions

Detailed Description

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

Functions

- int rtxDateToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxTimeToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxDateTimeToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGYearToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGYearMonthToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGMonthToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGMonthDayToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGDayToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGetCurrDateTime (OSNumDateTime * pvalue)
- int rtxGetCurrDateTimeString (char * buffer, OSSIZE bufsize, OSBOOL local)
- int rtxCmpDate (const OSNumDateTime * pvalue1, const OSNumDateTime * pvalue2)
- int rtxCmpDate2 (const OSNumDateTime * pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpTime (const OSNumDateTime * pvalue1, const OSNumDateTime * pvalue2)
- int rtxCmpTime2 (const OSNumDateTime * pvalue, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpDateTime (const OSNumDateTime * pvalue1, const OSNumDateTime * pvalue2)
- int rtxCmpDateTime2 (const OSNumDateTime * pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxParseDateString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseTimeString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseDateTimeString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGYearString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

- int rtxParseGYearMonthString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGMonthString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGMonthDayString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGDayString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR * buf, OSUINT32 bufsize)
- int rtxDurationToMSecs (OSUTF8CHAR * buf, OSUINT32 bufsize, OSINT32 * msecs)
- int rtxSetDateTime (OSNumDateTime * pvalue, struct tm * timeStruct)
- int rtxGetGMTime (struct tm * pvalue, time_t timeMs)
- int rtxGetLocalTime (struct tm * pvalue, time_t timeMs)
- int rtxSetLocalDateTime (OSNumDateTime * pvalue, time_t timeMs)
- int rtxSetUtcDateTime (OSNumDateTime * pvalue, time_t timeMs)
- int rtxGetDateTime (const OSNumDateTime * pvalue, time_t * timeMs)
- OSBOOL rtxDateIsValid (const OSNumDateTime * pvalue)
- OSBOOL rtxTimeIsValid (const OSNumDateTime * pvalue)
- OSBOOL rtxDateTimeIsValid (const OSNumDateTime * pvalue)
- int rtxAscTime (char * buffer, OSSIZE bufsize, struct tm * pvalue)

Function Documentation

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).

Table 2.49. 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

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]).

Table 2.50. 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

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]).

Table 2.51. 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

int rtxGYearToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)

This function formats a gregorian year value to a string (CCYY).

Table 2.52. 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

int rtxGYearMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)

This function formats a gregorian year and month value to a string (CCYY-MM).

Table 2.53. 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

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

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

Table 2.54. 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

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

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

Table 2.55. 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

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

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

Table 2.56. 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

int rtxGetCurrDateTime (OSNumDateTime *pvalue)

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

Table 2.57. Parameters

pvalue	Pointer to OSNumDateTime structure.

Returns: . Completion status of operation:

- 0 in case success
- negative in case failure

int rtxGetCurrDateTimeString (char *buffer, OSSIZE bufsize, OS-BOOL 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.

Table 2.58. 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

int rtxCmpDate (const OSNumDateTime *pvalue1, const OSNum-DateTime *pvalue2)

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

Table 2.59. 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.

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.

Table 2.60. 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.

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.

Table 2.61. 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.

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.

Table 2.62. Parameters

pvalue	Pointer to OSNumDateTime structure.
hour	Hour (0..23)
min	Minutes (0..59)
sec	Seconds (0.0..59.(9))

tzflag	TRUE, if time zone offset is set (see tzo parameter).
tzo	Time zone offset (-840..840).

Returns: . Completion status of operation:

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

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

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

Table 2.63. 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.

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.

Table 2.64. 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.

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.

Table 2.65. 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

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.

Table 2.66. 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

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.

Table 2.67. 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

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.

Table 2.68. 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

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.

Table 2.69. 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

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.

Table 2.70. 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

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.

Table 2.71. 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

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.

Table 2.72. 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

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

Table 2.73. 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

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.

Table 2.74. 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

int rtxSetDateTime (OSNumDateTime *pvalue, struct tm *timeStruct)

This function converts a structure of type 'struct tm' to an OSNumDateTime structure.

Table 2.75. 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.

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.

Table 2.76. 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.

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.

Table 2.77. 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.

int rtxSetLocalDateTime (OSNumDateTime *pvalue, time_t timeMs)

This function converts a local date and time value to an OSNumDateTime structure.

Table 2.78. 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.

int rtxSetUtcDateTime (OSNumDateTime *pvalue, time_t timeMs)

This function converts a UTC date and time value to an OSNumDateTime structure.

Table 2.79. 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.

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.

Table 2.80. 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.

OSBOOL rtxDateIsValid (const OSNumDateTime *pvalue)

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

Table 2.81. Parameters

pvalue	Pointer to OSNumDateTime structure to be checked.

Returns: . Boolean result: true means data is valid.

OSBOOL rtxTimeIsValid (const OSNumDateTime *pvalue)

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

Table 2.82. Parameters

pvalue	Pointer to OSNumDateTime structure to be checked.

Returns: . Boolean result: true means data is valid.

OSBOOL rtxDateTimeIsValid (const OSNumDateTime *pvalue)

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

Table 2.83. Parameters

pvalue	Pointer to OSNumDateTime structure to be checked.

Returns: . Boolean result: true means data is valid.

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.

Table 2.84. 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.

Floating-point number utility functions

Detailed Description

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

Functions

- OSREAL rtxGetMinusInfinity (OSVOIDARG)
- OSREAL rtxGetMinusZero (OSVOIDARG)
- OSREAL rtxGetNaN (OSVOIDARG)
- OSREAL rtxGetPlusInfinity (OSVOIDARG)

- OSBOOL rtxIsMinusInfinity (OSREAL value)
- OSBOOL rtxIsMinusZero (OSREAL value)
- OSBOOL rtxIsNaN (OSREAL value)
- OSBOOL rtxIsPlusInfinity (OSREAL value)
- OSBOOL rtxIsApproximate (OSREAL a, OSREAL b, OSREAL delta)
- OSBOOL rtxIsApproximateAbs (OSREAL a, OSREAL b, OSREAL delta)

Function Documentation

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.

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.

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.

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.

OSBOOL rtxIsMinusInfinity (OSREAL value)

A utility function that compares the given input value to the IEEE 754 value for negative infinity.

Table 2.85. Parameters

value	The input real value.

OSBOOL rtxIsMinusZero (OSREAL value)

A utility function that compares the given input value to the IEEE 754 value for minus zero.

Table 2.86. Parameters

value	The input real value.

OSBOOL rtxIsNaN (OSREAL value)

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

Table 2.87. Parameters

value	The input real value.

OSBOOL rtxIsPlusInfinity (OSREAL value)

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

Table 2.88. Parameters

value	The input real value.

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.

Table 2.89. 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.

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.

Table 2.90. Parameters

a	The input real value.
b	The input real value.
delta	difference must be low than delta

Decimal number utility functions

Detailed Description

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

Functions

- const char * rtxNR3toDecimal (OSCTXT * pctxt, const char * object_p)

UTF-8 String Functions

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.

Functions

- long rtxUTF8ToUnicode (OSCTXT * pctxt, const OSUTF8CHAR * inbuf, OSUNICHAR * outbuf, size_t outbufsiz)
- long rtxUTF8ToUnicode32 (OSCTXT * pctxt, const OSUTF8CHAR * inbuf, OS32BITCHAR * outbuf, size_t outbufsiz)
- int rtxValidateUTF8 (OSCTXT * pctxt, const OSUTF8CHAR * inbuf)
- size_t rtxUTF8Len (const OSUTF8CHAR * inbuf)
- size_t rtxCalcUTF8Len (const OSUTF8CHAR * inbuf, size_t inbufBytes)
- size_t rtxUTF8LenBytes (const OSUTF8CHAR * inbuf)
- int rtxUTF8CharSize (OS32BITCHAR wc)
- int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET * buf, size_t bufsiz)
- int rtxUTF8DecodeChar (OSCTXT * pctxt, const OSUTF8CHAR * pinbuf, int * pInsize)
- OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR * buf, OSUINT32 * len)
- OSUTF8CHAR * rtxUTF8StrChr (OSUTF8CHAR * utf8str, OS32BITCHAR utf8char)
- OSUTF8CHAR * rtxUTF8Strdup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str)
- OSUTF8CHAR * rtxUTF8Strndup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, size_t nbytes)
- OSUTF8CHAR * rtxUTF8StrRefOrDup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str)
- OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2)
- OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2, size_t count)
- int rtxUTF8Strcmp (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2)
- int rtxUTF8Strncmp (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2, size_t count)
- OSUTF8CHAR * rtxUTF8Strcpy (OSUTF8CHAR * dest, size_t bufsiz, const OSUTF8CHAR * src)
- OSUTF8CHAR * rtxUTF8Strncpy (OSUTF8CHAR * dest, size_t bufsiz, const OSUTF8CHAR * src, size_t nchars)

- OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR * str)
- const OSUTF8CHAR * rtxUTF8StrJoin (OSCTXT * pctxt, const OSUTF8CHAR * str1, const OSUTF8CHAR * str2, const OSUTF8CHAR * str3, const OSUTF8CHAR * str4, const OSUTF8CHAR * str5)
- int rtxUTF8StrToBool (const OSUTF8CHAR * utf8str, OSBOOL * pvalue)
- int rtxUTF8StrnToBool (const OSUTF8CHAR * utf8str, size_t nbytes, OSBOOL * pvalue)
- int rtxUTF8StrToDouble (const OSUTF8CHAR * utf8str, OSREAL * pvalue)
- int rtxUTF8StrnToDouble (const OSUTF8CHAR * utf8str, size_t nbytes, OSREAL * pvalue)
- int rtxUTF8StrToInt (const OSUTF8CHAR * utf8str, OSINT32 * pvalue)
- int rtxUTF8StrnToInt (const OSUTF8CHAR * utf8str, size_t nbytes, OSINT32 * pvalue)
- int rtxUTF8StrToUInt (const OSUTF8CHAR * utf8str, OSUINT32 * pvalue)
- int rtxUTF8StrnToUInt (const OSUTF8CHAR * utf8str, size_t nbytes, OSUINT32 * pvalue)
- int rtxUTF8StrToSize (const OSUTF8CHAR * utf8str, size_t * pvalue)
- int rtxUTF8StrnToSize (const OSUTF8CHAR * utf8str, size_t nbytes, size_t * pvalue)
- int rtxUTF8StrToInt64 (const OSUTF8CHAR * utf8str, OSINT64 * pvalue)
- int rtxUTF8StrnToInt64 (const OSUTF8CHAR * utf8str, size_t nbytes, OSINT64 * pvalue)
- int rtxUTF8StrToUInt64 (const OSUTF8CHAR * utf8str, OSUINT64 * pvalue)
- int rtxUTF8StrnToUInt64 (const OSUTF8CHAR * utf8str, size_t nbytes, OSUINT64 * pvalue)
- int rtxUTF8ToDynUniStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OSUNICHAR ** ppdata, OSUINT32 * pnchars)
- int rtxUTF8ToDynUniStr32 (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OS32BITCHAR ** ppdata, OSUINT32 * pnchars)
- int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR * utf8instr, size_t nbytes, const OSUTF8CHAR ** putf8outstr)
- int rtxUTF8StrToDynHexStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, OSDynOctStr * pvalue)
- int rtxUTF8StrnToDynHexStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, size_t nbytes, OSDynOctStr * pvalue)
- int rtxUTF8StrToNamedBits (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OSBitMapItem * pBitMap, OSOCTET * pvalue, OSUINT32 * pnbits, OSUINT32 bufsize)
- const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR * utf8str, OSUTF8CHAR ** ppNext)

Macros

- #define rtxUTF8StrToInt32 rtxUTF8StrToInt
- #define rtxUTF8StrToUInt32 rtxUTF8StrToUInt
- #define RTUTF8STRCMPL rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)

- #define OSRTCHKUTF8LEN do { size_t nchars = rtxUTF8Len (str); \ stat = (nchars >= lower && nchars <= upper) ? 0 : RTERR_CONSVIO; } while(0)

Function Documentation

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).

Table 2.91. 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.

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).

Table 2.92. 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.

int rtxValidateUTF8 (OSCTXT *pctxt, const OSUTF8CHAR *inbuf)

This function will validate a UTF-8 encoded string to ensure that it is encoded correctly.

Table 2.93. 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.

size_t rtxUTF8Len (const OSUTF8CHAR *inbuf)

This function will return the length (in characters) of a null-terminated UTF-8 encoded string.

Table 2.94. 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.

size_t rtxUTF8LenBytes (const OSUTF8CHAR *inbuf)

This function will return the length (in bytes) of a null-terminated UTF-8 encoded string.

Table 2.95. Parameters

inbuf	A pointer to the null-terminated UTF-8 encoded string.

Returns: . Number of bytes in the string.

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.

Table 2.96. Parameters

wc	32-bit wide character value.

Returns: . Number of bytes needed to encode as UTF-8.

int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET *buf, size_t bufsiz)

This function will convert a wide character into an encoded UTF-8 character byte string.

Table 2.97. 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.

int rtxUTF8DecodeChar (OSCTXT *pctxt, const OSUTF8CHAR *pin-buf, int *pInsize)

This function will convert an encoded UTF-8 character byte string into a wide character value.

Table 2.98. 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.

OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR *buf, OSUINT32 *len)

Thia function will convert a UTF-8 encoded character value into a wide character.

Table 2.99. 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.

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.

Table 2.100. 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.

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.

Table 2.101. Parameters

pctxt	A pointer to a context structure.
utf8str	Null-terminated UTF-8 string to be duplicated.

Returns: . Pointer to duplicated string value.

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.

Table 2.102. 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.

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.

Table 2.103. 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.

OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)

This function compares two UTF-8 string values for equality.

Table 2.104. Parameters

utf8str1	UTF-8 string to be compared.
utf8str2	UTF-8 string to be compared.

Returns: . TRUE if equal, FALSE if not.

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.

Table 2.105. 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.

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.

Table 2.106. 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.

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.

Table 2.107. 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.

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.

Table 2.108. 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.

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

Table 2.109. 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.

OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR *str)

This function computes a hash code for the given string value.

Table 2.110. Parameters

str	Pointer to string.

Returns: . Hash code value.

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.

Table 2.111. 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.

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'.

Table 2.112. Parameters

utf8str	Null-terminated UTF-8 string to convert
pvalue	Pointer to boolean value to receive result

Returns: . Status: 0 = OK, negative value = error

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'.

Table 2.113. 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

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.

Table 2.114. Parameters

utf8str	Null-terminated UTF-8 string to convert
pvalue	Pointer to double to receive result

Returns: . Status: 0 = OK, negative value = error

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.

Table 2.115. 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

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.

Table 2.116. Parameters

utf8str	Null-terminated UTF-8 string to convert
pvalue	Pointer to integer to receive result

Returns: . Status: 0 = OK, negative value = error

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.

Table 2.117. 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

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.

Table 2.118. Parameters

utf8str	Null-terminated UTF-8 string to convert
pvalue	Pointer to integer to receive result

Returns: . Status: 0 = OK, negative value = error

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.

Table 2.119. 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

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.

Table 2.120. 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

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.

Table 2.121. 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

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.

Table 2.122. Parameters

utf8str	Null-terminated UTF-8 string to convert
pvalue	Pointer to integer to receive result

Returns: . Status: 0 = OK, negative value = error

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.

Table 2.123. 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

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.

Table 2.124. Parameters

utf8str	Null-terminated UTF-8 string to convert
pvalue	Pointer to integer to receive result

Returns: . Status: 0 = OK, negative value = error

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.

Table 2.125. 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

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.

Table 2.126. 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

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.

Table 2.127. 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

int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR *utf8instr, size_t nbytes, const OSUTF8CHAR **putf8outstr)

This function removes leading and trailing whitespace from a string.

Table 2.128. 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.

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.

Table 2.129. 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.

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.

Table 2.130. 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.

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.

Table 2.131. 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

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.

Table 2.132. 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.

Macro Definition Documentation

#define RTUTF8STRCMPL

Compare UTF-8 string to a string literal.

Table 2.133. Parameters

name	UTF-8 string variable.
lstr	C string literal value (quoted contant such as "a")

Definition at line 567 of file rtxUTF8.h

The Documentation for this define was generated from the following file:

- rtxUTF8.h

Bit String Functions

Detailed Description

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

Functions

- OSUINT32 rtxGetBitCount (OSUINT32 value)
- int rtxSetBit (OSOCTET * pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)
- int rtxClearBit (OSOCTET * pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSBOOL rtxTestBit (const OSOCTET * pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSSIZE rtxLastBitSet (const OSOCTET * pBits, OSSIZE numbits)
- int rtxCheckBitBounds (OSCTXT * pctxt, OSOCTET ** ppBits, OSSIZE * pNumocts, OSSIZE minRequiredBits, OSSIZE preferredLimitBits)

- int rtxZeroUnusedBits (OSOCTET * pBits, OSSIZE numbits)
- int rtxCheckUnusedBitsZero (const OSOCTET * pBits, OSSIZE numbits)

Macros

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

Function Documentation

OSUINT32 rtxGetBitCount (OSUINT32 value)

This function returns the minimum size of the bit field required to hold the given integer value.

Table 2.134. Parameters

value	Integer value

Returns: . Minimum size of the the field in bits required to hold value.

int rtxSetBit (OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)

This function sets the specified zero-counted bit in the bit string.

Table 2.135. 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

OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OS-BOOL action)

This function sets one or more bits to TRUE or FALSE in a 32-bit unsigned bit flag set.

Table 2.136. 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.

int rtxClearBit (OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)

This function clears the specified zero-counted bit in the bit string.

Table 2.137. 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

OSBOOL rtxTestBit (const OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)

This function tests the specified zero-counted bit in the bit string.

Table 2.138. 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.

OSSIZE rtxLastBitSet (const OSOCTET *pBits, OSSIZE numbits)

This function returns the zero-counted index of the last bit set in a bit string.

Table 2.139. 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.

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.

Table 2.140. 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.

int rtxZeroUnusedBits (OSOCTET *pBits, OSSIZE numbits)

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

Table 2.141. 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.

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.

Table 2.142. 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.

Macro Definition Documentation

#define OSRTBYTEARRAYSIZE

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

Definition at line 48 of file rtxBitString.h

The Documentation for this define was generated from the following file:

- rtxBitString.h

Context Management Functions

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.

Classes

- struct OSRTErrLocn
- struct OSRTErrInfo
- struct OSRTErrInfoList
- struct OSRTBuffer
- struct OSRTBufSave
- struct OSBufferIndex
- struct OSCTXT

Typedefs

- typedef OSUINT32 OSRTFLAGS
- typedef int(* OSFreeCtxtAppInfoPtr
- typedef int(* OSResetCtxtAppInfoPtr
- typedef void(* OSFreeCtxtGlobalPtr
- typedef struct OSCTXT OSCTXT

Functions

- int rtxInitContext (OSCTXT * pctxt)
- int rtxInitContextExt (OSCTXT * pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- int rtxInitThreadContext (OSCTXT * pctxt, const OSCTXT * pSrcCtxt)
- int rtxInitContextUsingKey (OSCTXT * pctxt, const OSOCTET * key, OSSIZE keylen)
- int rtxInitContextBuffer (OSCTXT * pctxt, OSOCTET * bufaddr, OSSIZE bufsiz)

- int rtxCtxtSetBufPtr (OSCTXT * pctxt, OSOCTET * bufaddr, OSSIZE bufsiz)
- OSSIZE rtxCtxtGetBitOffset (OSCTXT * pctxt)
- int rtxCtxtSetBitOffset (OSCTXT * pctxt, OSSIZE offset)
- OSSIZE rtxCtxtGetIOByteCount (OSCTXT * pctxt)
- int rtxCheckContext (OSCTXT * pctxt)
- void rtxFreeContext (OSCTXT * pctxt)
- void rtxCopyContext (OSCTXT * pdest, OSCTXT * psrc)
- void rtxCtxtSetFlag (OSCTXT * pctxt, OSUINT32 mask)
- void rtxCtxtClearFlag (OSCTXT * pctxt, OSUINT32 mask)
- int rtxCtxtPushArrayElemName (OSCTXT * pctxt, const OSUTF8CHAR * elemName, OSSIZE idx)
- int rtxCtxtPushElemName (OSCTXT * pctxt, const OSUTF8CHAR * elemName)
- int rtxCtxtPushElemNameCopy (OSCTXT * pctxt, const OSUTF8CHAR * elemName)
- int rtxCtxtPushTypeName (OSCTXT * pctxt, const OSUTF8CHAR * typeName)
- OSBOOL rtxCtxtPopArrayElemName (OSCTXT * pctxt)
- const OSUTF8CHAR * rtxCtxtPopElemName (OSCTXT * pctxt)
- void rtxCtxtPopElemNameCopy (OSCTXT * pctxt)
- const OSUTF8CHAR * rtxCtxtPopTypeName (OSCTXT * pctxt)
- OSBOOL rtxCtxtContainerHasRemBits (OSCTXT * pctxt)
- OSBOOL rtxCtxtContainerEnd (OSCTXT * pctxt)
- OSSIZE rtxCtxtGetContainerRemBits (OSCTXT * pctxt)
- int rtxCtxtPushContainerBytes (OSCTXT * pctxt, OSSIZE bytes)
- int rtxCtxtPushContainerBits (OSCTXT * pctxt, OSSIZE bits)
- void rtxCtxtPopContainer (OSCTXT * pctxt)
- void rtxCtxtPopAllContainers (OSCTXT * pctxt)
- int rtxPreInitContext (OSCTXT * pctxt)
- void rtxCtxtSetMemHeap (OSCTXT * pctxt, OSCTXT * pSrcCtxt)
- void rtxMemHeapSetFlags (OSCTXT * pctxt, OSUINT32 flags)
- void rtxMemHeapClearFlags (OSCTXT * pctxt, OSUINT32 flags)
- int rtxCtxtMarkBitPos (OSCTXT * pctxt, OSSIZE * ppos)
- int rtxCtxtResetToBitPos (OSCTXT * pctxt, OSSIZE pos)

- int rtxMarkPos (OSCTXT * pctxt, OSSIZE * ppos)
- int rtxResetToPos (OSCTXT * pctxt, OSSIZE pos)
- const char * rtxCtxtGetExpDateStr (OSCTXT * pctxt, char * buf, OSSIZE bufsiz)
- void rtxLicenseClose (void)

Macros

- #define OSRTERRSTKSIZ 8 /* error stack size */
- #define OSRTMAXERRPRM 5 /* maximum error parameters */
- #define OSDIAG 0x80000000 /* diagnostic tracing enabled */
- #define OSTRACE 0x40000000 /* tracing enabled */
- #define OSDISSTRM 0x20000000 /* disable stream encode/decode */
- #define OSNOSTRMBACKOFF 0x08000000 /* stream mark/reset funcs is not used */
- #define OS3GMOBORIG 0x04000000 /* 3G mobile-originated (net to MS) */
- #define OSCONTCLOSED 0x02000000 /* 3G container closed. */
- #define OSRESERVED1 0x01000000 /* reserved */
- #define OSBUFSYSALLOC 0x00800000 /* ctxt buf allocated using sys alloc */
- #define OSLICCHECKIN 0x00400000 /* check in ifloat license on free */
- #define OSNOWHITESPACE 0x00400000 /* Turn off indentation whitesapce */
- #define OSCDECL
- #define pLicInfo pli709
- #define OSRT_GET_FIRST_ERROR_INFO (((pctxt)->errInfo.list.head == 0) ? (OSRTErrInfo*)0 : \ (OSRTErrInfo*)((pctxt)->errInfo.list.head->data))
- #define OSRT_GET_LAST_ERROR_INFO (((pctxt)->errInfo.list.tail == 0) ? (OSRTErrInfo*)0 : \ (OSRTErrInfo*)((pctxt)->errInfo.list.tail->data))
- #define rtxCtxtGetMsgPtr (pctxt)->buffer.data
- #define rtxCtxtGetMsgLen (pctxt)->buffer.byteIndex
- #define rtxCtxtTestFlag (((pctxt)->flags & mask) != 0)
- #define rtxCtxtPeekElemName (((pctxt)->elemNameStack.count > 0) ? \ (const OSUTF8CHAR*)(pctxt)->elemNameStack.tail->data : (const OSUTF8CHAR*)0)
- #define rtxByteAlign if ((pctxt)->buffer.bitOffset != 8) { \ (pctxt)->buffer.byteIndex++; (pctxt)->buffer.bitOffset = 8; }
- #define rtxCtxtSetProtocolVersion (pctxt)->version = value
- #define rtxMarkBitPos (*(pbitoff) = (OSUINT8) (pctxt)->buffer.bitOffset, rtxMarkPos (pctxt, ppos))

- #define rtxResetToBitPos ((pctxt)->buffer.bitOffset = (OSUINT8) bitoff, rtxResetToPos (pctxt, pos))
- #define RTXCTXTPUSHARRAYELEMNAME rtxCtxtPushArrayElemName(pctxt,OSUTF8(name),idx)
- #define RTXCTXTPOPARRAYELEMNAME rtxCtxtPopArrayElemName(pctxt)
- #define RTXCTXTPUSHELEMNAME rtxCtxtPushElemName(pctxt,OSUTF8(name))
- #define RTXCTXTPOPELEMNAME rtxCtxtPopElemName(pctxt)
- #define RTXCTXTPUSHTYPENAME rtxCtxtPushTypeName(pctxt,OSUTF8(name))
- #define RTXCTXTPOPTYPENAME rtxCtxtPopTypeName(pctxt)

Typedef Documentation

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. 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.

Function Documentation

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.

Table 2.143. Parameters

pctxt	Pointer to the context structure variable to be initialized.

Returns: . Completion status of operation:

- 0 = success,

- negative return value is error.

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.

Table 2.144. 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.

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.

Table 2.145. 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.

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.

Table 2.146. 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.

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 rtxInit-Context.

Table 2.147. 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.

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.

Table 2.148. 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).

OSSIZE rtxCtxtGetBitOffset (OSCTXT *pctxt)

This function returns the total bit offset to the current element in the context buffer.

Table 2.149. Parameters

pctxt	Pointer to a context structure.

Returns: . Bit offset.

int rtxCtxtSetBitOffset (OSCTXT *pctxt, OSSIZE offset)

This function sets the bit offset in the context to the given value.

Table 2.150. Parameters

pctxt	Pointer to a context structure.
offset	Bit offset.

Returns: . Completion status of operation:

- 0 = success,
- Negative status code if error

OSSIZE rtxCtxtGetIOByteCount (OSCTXT *pctxt)

This function returns the count of bytes either written to a stream or memory buffer.

Table 2.151. Parameters

pctxt	Pointer to a context structure.

Returns: . I/O byte count.

int rtxCheckContext (OSCTXT *pctxt)

This function verifies that the given context structure is initialized and ready for use.

Table 2.152. Parameters

pctxt	Pointer to a context structure.

Returns: . Completion status of operation:

- 0 = success,
- RTERR_NOTINIT status code if not initialized

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.

Table 2.153. Parameters

pctxt	Pointer to a context structure.

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.

Table 2.154. Parameters

pdest	- Context structure to which data is to be copied.
psrc	- Context structure from which data is to be copied.

void rtxCtxtSetFlag (OSCTXT *pctxt, OSUINT32 mask)

This function is used to set a processing flag within the context structure.

Table 2.155. Parameters

pctxt	- A pointer to a context structure.
mask	- Mask containing bit(s) to be set.

void rtxCtxtClearFlag (OSCTXT *pctxt, OSUINT32 mask)

This function is used to clear a processing flag within the context structure.

Table 2.156. Parameters

pctxt	- A pointer to a context structure.
mask	- Mask containing bit(s) to be cleared.

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.

Table 2.157. 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.

int rtxCtxtPushElemName (OSCTXT *pctxt, const OSUTF8CHAR *elemName)

This function is used to push an element name onto the context element name stack.

Table 2.158. 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.

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.

Table 2.159. 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.

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.

Table 2.160. 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.

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.

Table 2.161. Parameters

pctxt	Pointer to a context structure.

Returns: . True if name popped from stack or false if stack is empty.

const OSUTF8CHAR* rtxCtxtPopElemName (OSCTXT *pctxt)

This function pops the last element name from the context stack.

Table 2.162. Parameters

pctxt	Pointer to a context structure.

Returns: . Element name popped from stack or NULL if stack is empty.

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.

Table 2.163. Parameters

pctxt	Pointer to a context structure.

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.

Table 2.164. Parameters

pctxt	Pointer to a context structure.

Returns: . Type name popped from stack or NULL if stack count not equal to one.

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

Table 2.165. Parameters

pctxt	Pointer to context structure.

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

Table 2.166. Parameters

pctxt	Pointer to context structure.

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

Table 2.167. Parameters

pctxt	Pointer to context structure.

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.

Table 2.168. 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.

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.

Table 2.169. 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.

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

Table 2.170. Parameters

pctxt	Pointer to context structure.

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.

Table 2.171. Parameters

pctxt	Pointer to context structure.

void rtxMemHeapSetFlags (OSCTXT *pctxt, OSUINT32 flags)

This function sets flags to a heap. May be used to control the heap's behavior.

Table 2.172. Parameters

pctxt	Pointer to a memory block structure that contains the list of dynamic memory block maintained by these functions.
flags	The flags.

void rtxMemHeapClearFlags (OSCTXT *pctxt, OSUINT32 flags)

This function clears memory heap flags.

Table 2.173. Parameters

pctxt	Pointer to a memory block structure that contains the list of dynamic memory block maintained by these functions.
flags	The flags

int rtxCtxtMarkBitPos (OSCTXT *pctxt, OSSIZE *ppos)

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

Table 2.174. Parameters

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

Returns: . Completion status of operation:

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

int rtxCtxtResetToBitPos (OSCTXT *pctxt, OSSIZE pos)

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

Table 2.175. Parameters

pctxt	Pointer to a context block.
pos	Context position.

Returns: . Completion status of operation:

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

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 rtx-MarkBitPos).

Table 2.176. Parameters

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

Returns: . Completion status of operation:

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

int rtxResetToPos (OSCTXT *pctxt, OSSIZE pos)

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

Table 2.177. Parameters

pctxt	Pointer to a context block.
pos	Context position.

Returns: . Completion status of operation:

- 0 = success,

- negative return value is error.

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

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

Table 2.178. 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.

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.

Macro Definition Documentation

#define OSNOWHITESPACE

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

The Documentation for this define was generated from the following file:

- rtxContext.h

#define rtxCtxtGetMsgPtr

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.

Table 2.179. Parameters

pctxt	Pointer to a context structure.

Definition at line 458 of file rtxContext.h

The Documentation for this define was generated from the following file:

- rtxContext.h

#define rtxCtxtGetMsgLen

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.

Table 2.180. Parameters

pctxt	Pointer to a context structure.

Definition at line 469 of file rtxContext.h

The Documentation for this define was generated from the following file:

- rtxContext.h

#define rtxCtxtTestFlag

This macro tests if the given bit flag is set in the context.

Table 2.181. Parameters

pctxt	- A pointer to a context structure.
mask	- Bit flag to be tested

Definition at line 554 of file rtxContext.h

The Documentation for this define was generated from the following file:

- rtxContext.h

#define rtxCtxtPeekElemName

This macro returns the last element name from the context stack.

Table 2.182. 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

The Documentation for this define was generated from the following file:

- rtxContext.h

#define rtxByteAlign

This macro will byte-align the context buffer.

Definition at line 783 of file rtxContext.h

The Documentation for this define was generated from the following file:

- rtxContext.h

#define rtxCtxtSetProtocolVersion

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.

Table 2.183. Parameters

pctxt	Pointer to a context structure.
value	The version number value.

Definition at line 861 of file rtxContext.h

The Documentation for this define was generated from the following file:

- rtxContext.h

Memory Allocation Macros and Functions

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.

Functions

- void rtxMemHeapAddRef (void ** ppvMemHeap)
- void * rtxMemHeapAlloc (void ** ppvMemHeap, OSSIZE nbytes)
- void * rtxMemHeapAllocZ (void ** ppvMemHeap, OSSIZE nbytes)
- void * rtxMemHeapSysAlloc (void ** ppvMemHeap, OSSIZE nbytes)
- void * rtxMemHeapSysAllocZ (void ** ppvMemHeap, OSSIZE nbytes)
- int rtxMemHeapCheckPtr (void ** ppvMemHeap, const void * mem_p)

- void rtxMemHeapFreeAll (void ** ppvMemHeap)
- void rtxMemHeapFreePtr (void ** ppvMemHeap, void * mem_p)
- void rtxMemHeapSysFreePtr (void ** ppvMemHeap, void * mem_p)
- void * rtxMemHeapReallocStatic (void ** ppvMemHeap, void * mem_p, OSSIZE oldsize, OSSIZE newsize)
- void * rtxMemHeapRealloc (void ** ppvMemHeap, void * mem_p, OSSIZE nbytes_)
- void * rtxMemHeapSysRealloc (void ** ppvMemHeap, void * mem_p, OSSIZE nbytes_)
- void rtxMemHeapRelease (void ** ppvMemHeap)
- void rtxMemHeapReset (void ** ppvMemHeap)
- void rtxMemHeapSetProperty (void ** ppvMemHeap, OSUINT32 propId, void * pProp)
- void * rtxMemNewArray (OSSIZE nbytes)
- void * rtxMemNewArrayZ (OSSIZE nbytes)
- void rtxMemDeleteArray (void * mem_p)
- void * rtxMemHeapAutoPtrRef (void ** ppvMemHeap, void * ptr)
- int rtxMemHeapAutoPtrUnref (void ** ppvMemHeap, void * ptr)
- int rtxMemHeapAutoPtrGetRefCount (void ** ppvMemHeap, void * mem_p)
- void rtxMemHeapInvalidPtrHook (void ** ppvMemHeap, const void * mem_p)
- void rtxMemHeapCheck (void ** ppvMemHeap, const char * file, int line)
- void rtxMemHeapPrint (void ** ppvMemHeap)
- void rtxMemHeapPrintWithFree (void ** ppvMemHeap)
- int rtxMemHeapCreate (void ** ppvMemHeap)
- int rtxMemHeapCreateExt (void ** ppvMemHeap, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- int rtxMemStaticHeapCreate (void ** ppvMemHeap, void * pmem, OSSIZE memsize)
- int rtxMemHeapConvertStatic (void ** ppvMemHeap, void * pmem, OSSIZE memsize)
- void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- void rtxMemFreeOpenSeqExt (OSCTXT * pctxt, struct OSRTDList * pElemList)
- OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT * pctxt)
- void rtxMemSetDefBlkSize (OSUINT32 blkSize)
- OSUINT32 rtxMemGetDefBlkSize (OSVOIDARG)
- OSBOOL rtxMemHeapIsEmpty (OSCTXT * pctxt)

- OSBOOL rtxMemIsZero (const void * pmem, OSSIZE memsiz)
- void rtxMemFree (OSCTXT * pctxt)
- void rtxMemReset (OSCTXT * pctxt)
- void * rtxMemAllocArray2 (OSCTXT * pctxt, OSSIZE numElements, OSSIZE typeSize, OSUINT32 flags)

Macros

- #define OSRTALLOCTYPE (type*) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTALLOCTYPEZ (type*) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTREALLOCARRAY 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)
- #define OSCRTMALLOC0 malloc(nbytes)
- #define OSCRTFREE0 free(ptr)
- #define OSCRTMALLOC rtxMemAlloc
- #define OSCRTFREE rtxMemFreePtr
- #define OSCDECL
- #define rtxMemAlloc rtxMemHeapAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAlloc rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemAllocZ rtxMemHeapAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAllocZ rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemRealloc rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void*)mem_p, nbytes)
- #define rtxMemSysRealloc rtxMemHeapSysRealloc(&(pctxt)->pMemHeap,(void*)mem_p,nbytes)
- #define rtxMemFreePtr rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemSysFreePtr rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemAllocType (ctype*)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocType (ctype*)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemAllocTypeZ (ctype*)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocTypeZ (ctype*)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemFreeType rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemSysFreeType rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemAllocArray (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), 0)
- #define rtxMemSysAllocArray (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_SYSALLOC)

- #define rtxMemAllocArrayZ (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_ZEROARRAY)
- #define rtxMemFreeArray rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemSysFreeArray rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemReallocArray (type*)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void*)mem_p, sizeof(type)*n)
- #define rtxMemNewAutoPtr rtxMemHeapAlloc(&(pctxt)->pMemHeap, nbytes)
- #define rtxMemAutoPtrRef rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemAutoPtrUnref rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemAutoPtrGetRefCount rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemCheckPtr rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemCheck rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)
- #define rtxMemPrint rtxMemHeapPrint(&(pctxt)->pMemHeap)
- #define rtxMemPrintWithFree rtxMemHeapPrintWithFree(&(pctxt)->pMemHeap)
- #define rtxMemSetProperty rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, pProp)

Function Documentation

void rtxMemHeapPrint (void **ppvMemHeap)

Print details about the memory heap.

void rtxMemHeapPrintWithFree (void **ppvMemHeap)

Print the same details about the memory heap as rtxMemHeapPrint but add deatils about the free memory list.

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.

Table 2.184. 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.

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.

Table 2.185. 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.

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.

Table 2.186. 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.

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.

Table 2.187. 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.

void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSRealloc-Func 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.

Table 2.188. 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).

OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT *pctxt)

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

Table 2.189. Parameters

pctxt	Pointer to a context block.

void rtxMemSetDefBlkSize (OSUINT32 blkSize)

This function sets the minimum size and the granularity of memory blocks for newly created memory heaps.

Table 2.190. Parameters

blkSize	The minimum size and the granularity of memory blocks.

OSUINT32 rtxMemGetDefBlkSize (OSVOIDARG)

This function returns the actual granularity of memory blocks.

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

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).

Table 2.191. Parameters

pctxt	Pointer to a context block.

Returns: . Boolean true value if heap is empty.

OSBOOL rtxMemIsZero (const void *pmem, OSSIZE memsiz)

This helper function determines if an arbitrarily sized block of memory is set to zero.

Table 2.192. Parameters

pmem	Pointer to memory block to check
memsiz	Size of the memory block

Returns: . Boolean result: true if memory is all zero

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.

Table 2.193. Parameters

pctxt	- Pointer to a context block

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.

Table 2.194. Parameters

pctxt	- Pointer to a context block

Macro Definition Documentation

#define OSRTALLOCTYPE

This macro allocates a single element of the given type.

Table 2.195. Parameters

pctxt	- Pointer to a context block

type	- Data type of record to allocate

Definition at line 77 of file rtxMemory.h

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define OSRTALLOCTYPEZ

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

Table 2.196. Parameters

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

Definition at line 86 of file rtxMemory.h

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define OSRTREALLOCARRAY

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.

Table 2.197. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAlloc

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

Table 2.198. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysAlloc

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.

Table 2.199. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAllocZ

Allocate and zero memory. This macro allocates the given number of bytes and then initializes the memory block to zero.

Table 2.200. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysAllocZ

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.

Table 2.201. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemRealloc

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.

Table 2.202. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysRealloc

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.

Table 2.203. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemFreePtr

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.

Table 2.204. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysFreePtr

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.

Table 2.205. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAllocType

Allocate type. This macro allocates memory to hold a variable of the given type.

Table 2.206. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysAllocType

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.

Table 2.207. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAllocTypeZ

Allocate type and zero memory. This macro allocates memory to hold a variable of the given type and initializes the allocated memory to zero.

Table 2.208. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysAllocTypeZ

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.

Table 2.209. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemFreeType

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.

Table 2.210. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysFreeType

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.

Table 2.211. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAllocArray

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.

Table 2.212. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysAllocArray

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.

Table 2.213. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAllocArrayZ

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.

Table 2.214. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemFreeArray

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.

Table 2.215. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSysFreeArray

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.

Table 2.216. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemReallocArray

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.

Table 2.217. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemNewAutoPtr

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.

Table 2.218. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAutoPtrRef

This function increments the auto-pointer reference count.

Table 2.219. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAutoPtrUnref

This function decrements the auto-pointer reference count. If the count goes to zero, the memory is freed.

Table 2.220. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemAutoPtrGetRefCount

This function returns the reference count of the given pointer. goes to zero, the memory is freed.

Table 2.221. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemCheckPtr

Check memory pointer. This macro check pointer on presence in heap.

Table 2.222. 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

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemCheck

Check memory heap.

Table 2.223. Parameters

pctxt	- Pointer to a context block

Definition at line 723 of file rtxMemory.h

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemPrint

Print memory heap structure to stderr.

Table 2.224. Parameters

pctxt	- Pointer to a context block

Definition at line 731 of file rtxMemory.h

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemPrintWithFree

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

Table 2.225. Parameters

pctxt	- Pointer to a context block

Definition at line 740 of file rtxMemory.h

The Documentation for this define was generated from the following file:

- rtxMemory.h

#define rtxMemSetProperty

Set memory heap property.

Table 2.226. Parameters

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

Definition at line 750 of file rtxMemory.h

The Documentation for this define was generated from the following file:

- rtxMemory.h

Memory Buffer Management Functions

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.

Classes

- struct OSRTMEMBUF

Typedefs

- typedef struct OSRTMEMBUF OSRTMEMBUF

Functions

- int rtxMemBufAppend (OSRTMEMBUF * pMemBuf, const OSOCTET * pdata, OSSIZE nbytes)
- int rtxMemBufCut (OSRTMEMBUF * pMemBuf, OSSIZE fromOffset, OSSIZE nbytes)
- void rtxMemBufFree (OSRTMEMBUF * pMemBuf)
- OSOCTET * rtxMemBufGetData (const OSRTMEMBUF * pMemBuf, int * length)

- OSOCTET * rtxMemBufGetDataExt (const OSRTMEMBUF * pMemBuf, OSSIZE * length)
- OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF * pMemBuf)
- void rtxMemBufInit (OSCTXT * pCtxt, OSRTMEMBUF * pMemBuf, OSSIZE segsize)
- void rtxMemBufInitBuffer (OSCTXT * pCtxt, OSRTMEMBUF * pMemBuf, OSOCTET * buf, OSSIZE bufsize, OSSIZE segsize)
- int rtxMemBufPreAllocate (OSRTMEMBUF * pMemBuf, OSSIZE nbytes)
- void rtxMemBufReset (OSRTMEMBUF * pMemBuf)
- int rtxMemBufSet (OSRTMEMBUF * pMemBuf, OSOCTET value, OSSIZE nbytes)
- OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF * pMemBuf, OSBOOL isExpandable)
- OSBOOL rtxMemBufSetUseSysMem (OSRTMEMBUF * pMemBuf, OSBOOL value)
- OSSIZE rtxMemBufTrimW (OSRTMEMBUF * pMemBuf)

Macros

- #define OSMBDFLTSEGSIZE 1024
- #define OSMEMBUFPTR ((pmb)->buffer + (pmb)->startidx)
- #define OSMEMBUFENDPTR ((pmb)->buffer + (pmb)->startidx + (pmb)->usedcnt)
- #define OSMEMBUFUSEDSIZE ((OSSIZE)(pmb)->usedcnt)
- #define OSMBAPPENDSTR if (0 != str) \ rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
- #define OSMBAPPENDSTRL rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
- #define OSMBAPPENDUTF8 if (0 != str) \ rtxMemBufAppend(pmb,(OSOCTET*)str,rtxUTF8LenBytes(str))
- #define OSMBAPPENDSTRZ rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str)+1)

Function Documentation

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.

Table 2.227. 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.

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.

Table 2.228. 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.

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.

Table 2.229. Parameters

pMemBuf	A pointer to a memory buffer structure.

OSOCTET* rtxMemBufGetData (const OSRTMEMBUF *pMemBuf, int *length)

This function returns the pointer to the used part of a memory buffer.

Table 2.230. 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.

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.

Table 2.231. 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.

OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF *pMemBuf)

This function returns the length of the used part of a memory buffer.

Table 2.232. Parameters

pMemBuf	A pointer to a memory buffer structure.

Returns: . The length of the used part of the buffer.

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.

Table 2.233. 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.

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.

Table 2.234. 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.

int rtxMemBufPreAllocate (OSRTMEMBUF *pMemBuf, OSSIZE nbytes)

This function allocates a buffer with a predetermined amount of space.

Table 2.235. 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.

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.

Table 2.236. Parameters

pMemBuf	A pointer to a memory buffer structure.

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.

Table 2.237. 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.

OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF *pMemBuf, OS-BOOL 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.

Table 2.238. Parameters

pMemBuf	A pointer to a memory buffer structure.
isExpandable	TRUE, if buffer should be expandable.

Returns: . Previous state of "isExpandable" flag.

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.

Table 2.239. Parameters

pMemBuf	A pointer to a memory buffer structure.
value	Boolean indicating system memory management to be used.

Returns: . Previous state of "useSysMem" flag.

OSSIZE rtxMemBufTrimW (OSRTMEMBUF *pMemBuf)

This function trims white space of the memory buffer.

Table 2.240. Parameters

pMemBuf	A pointer to a memory buffer structure.

Returns: . Length of trimmed buffer.

Print Functions

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.

Functions

- void rtxPrintBoolean (const char * name, OSBOOL value)
- void rtxPrintDate (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintTime (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintDateTime (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGYear (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGYearMonth (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGMonth (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGMonthDay (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGDay (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintInteger (const char * name, OSINT32 value)
- void rtxPrintInt64 (const char * name, OSINT64 value)
- void rtxPrintIpv4Addr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintIpv6Addr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintTBCDStr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintText (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintUnsigned (const char * name, OSUINT32 value)
- void rtxPrintUInt64 (const char * name, OSUINT64 value)
- void rtxPrintHexStr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintHexStrPlain (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintHexStrNoAscii (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintHexBinary (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintCharStr (const char * name, const char * cstring)

- void rtxPrintUTF8CharStr (const char * name, const OSUTF8CHAR * cstring)
- void rtxPrintUnicodeCharStr (const char * name, const OSUNICHAR * str, int nchars)
- void rtxPrintUnicodeCharStr64 (const char * name, const OSUNICHAR * str, OSSIZE nchars)
- void rtxPrintReal (const char * name, OSREAL value)
- void rtxPrintNull (const char * name)
- void rtxPrintNVP (const char * name, const OSUTF8NVP * value)
- void rtxPrintArrayNVP (const char * name, OSSIZE subscript, const OSUTF8NVP * value)
- int rtxPrintFile (const char * filename)
- void rtxPrintIndent (OSVOIDARG)
- void rtxPrintIncrIndent (OSVOIDARG)
- void rtxPrintDecrIndent (OSVOIDARG)
- void rtxPrintCloseBrace (OSVOIDARG)
- void rtxPrintOpenBrace (const char *)
- const char * rtxGetArrayElemName (char * buffer, OSSIZE bufsize, const char * name, OSSIZE subscript)

Function Documentation

void rtxPrintBoolean (const char *name, OSBOOL value)

Prints a boolean value to stdout.

Table 2.241. Parameters

name	The name of the variable to print.
value	Boolean value to print.

void rtxPrintDate (const char *name, const OSNumDateTime *pval-ue)

Prints a date value to stdout.

Table 2.242. Parameters

name	Name of the variable to print.
pvalue	Pointer to a structure that holds numeric DateTime value to print.

void rtxPrintTime (const char *name, const OSNumDateTime *pvalue)

Prints a time value to stdout.

Table 2.243. Parameters

name	Name of the variable to print.
pvalue	Pointer to a structure that holds numeric DateTime value to print.

void rtxPrintDateTime (const char *name, const OSNumDateTime *pvalue)

Prints a dateTime value to stdout.

Table 2.244. Parameters

name	Name of the variable to print.
pvalue	Pointer to a structure that holds numeric DateTime value to print.

void rtxPrintInteger (const char *name, OSINT32 value)

Prints an integer value to stdout.

Table 2.245. Parameters

name	The name of the variable to print.
value	Integer value to print.

void rtxPrintInt64 (const char *name, OSINT64 value)

Prints a 64-bit integer value to stdout.

Table 2.246. Parameters

name	The name of the variable to print.
value	64-bit integer value to print.

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.

Table 2.247. 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.

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.

Table 2.248. 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.

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.

Table 2.249. 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.

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.

Table 2.250. 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.

void rtxPrintUnsigned (const char *name, OSUINT32 value)

Prints an unsigned integer value to stdout.

Table 2.251. Parameters

name	The name of the variable to print.
value	Unsigned integer value to print.

void rtxPrintUInt64 (const char *name, OSUINT64 value)

Prints an unsigned 64-bit integer value to stdout.

Table 2.252. Parameters

name	The name of the variable to print.
value	Unsigned 64-bit integer value to print.

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.

Table 2.253. 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.

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.

Table 2.254. 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.

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.

Table 2.255. 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.

void rtxPrintHexBinary (const char *name, OSSIZE numocts, const OSOCTET *data)

Prints an octet string value in hex binary format to stdout.

Table 2.256. 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.

void rtxPrintCharStr (const char *name, const char *cstring)

Prints an ASCII character string value to stdout.

Table 2.257. Parameters

name	The name of the variable to print.
cstring	A pointer to the character string to be printed.

void rtxPrintUTF8CharStr (const char *name, const OSUTF8CHAR *cstring)

Prints a UTF-8 encoded character string value to stdout.

Table 2.258. Parameters

name	The name of the variable to print.
cstring	A pointer to the character string to be printed.

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).

Table 2.259. 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).

void rtxPrintReal (const char *name, OSREAL value)

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

Table 2.260. Parameters

name	The name of the variable to print.
value	REAL value to print.

void rtxPrintNull (const char *name)

Prints a NULL value to stdout.

Table 2.261. Parameters

name	The name of the variable to print.

void rtxPrintNVP (const char *name, const OSUTF8NVP *value)

Prints a name-value pair to stdout.

Table 2.262. Parameters

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

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]".

Table 2.263. Parameters

name	The name of the variable to print.

subscript	Array subscript value.
value	A pointer to name-value pair structure to print.

int rtxPrintFile (const char *filename)

This function prints the contents of a text file to stdout.

Table 2.264. Parameters

filename	The name of the text file to print.

Returns: . Status of operation, 0 if success.

void rtxPrintIndent (OSVOIDARG)

This function prints indentation spaces to stdout.

void rtxPrintIncrIndent (OSVOIDARG)

This function increments the current indentation level.

void rtxPrintDecrIndent (OSVOIDARG)

This function decrements the current indentation level.

void rtxPrintCloseBrace (OSVOIDARG)

This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.

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.

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.

Table 2.265. 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

Print-To-Stream Functions

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.

Functions

- void rtxPrintToStreamBoolean (OSCTXT * pctxt, const char * name, OSBOOL value)
- void rtxPrintToStreamDate (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamTime (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamDateTime (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGYear (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGYearMonth (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGMonth (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGMonthDay (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGDay (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamInteger (OSCTXT * pctxt, const char * name, OSINT32 value)
- void rtxPrintToStreamInt64 (OSCTXT * pctxt, const char * name, OSINT64 value)
- void rtxPrintToStreamUnsigned (OSCTXT * pctxt, const char * name, OSUINT32 value)
- void rtxPrintToStreamUInt64 (OSCTXT * pctxt, const char * name, OSUINT64 value)
- void rtxPrintToStreamHexStr (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamHexStrPlain (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamHexStrNoAscii (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamHexBinary (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamCharStr (OSCTXT * pctxt, const char * name, const char * cstring)
- void rtxPrintToStreamUTF8CharStr (OSCTXT * pctxt, const char * name, const OSUTF8CHAR * cstring)
- void rtxPrintToStreamUnicodeCharStr (OSCTXT * pctxt, const char * name, const OSUNICHAR * str, int nchars)
- void rtxPrintToStreamReal (OSCTXT * pctxt, const char * name, OSREAL value)

- void rtxPrintToStreamNull (OSCTXT * pctxt, const char * name)
- void rtxPrintToStreamNVP (OSCTXT * pctxt, const char * name, const OSUTF8NVP * value)
- int rtxPrintToStreamFile (OSCTXT * pctxt, const char * filename)
- void rtxPrintToStreamIndent (OSCTXT * pctxt)
- void rtxPrintToStreamResetIndent (OSCTXT * pctxt)
- void rtxPrintToStreamIncrIndent (OSCTXT * pctxt)
- void rtxPrintToStreamDecrIndent (OSCTXT * pctxt)
- void rtxPrintToStreamCloseBrace (OSCTXT * pctxt)
- void rtxPrintToStreamOpenBrace (OSCTXT * pctxt, const char *)
- void rtxHexDumpToStream (OSCTXT * pctxt, const OSOCTET * data, OSSIZE numocts)
- void rtxHexDumpToStreamEx (OSCTXT * pctxt, const OSOCTET * data, OSSIZE numocts, OSSIZE bytesPe-rUnit)
- void rtxHexDumpToStreamExNoAscii (OSCTXT * pctxt, const OSOCTET * data, OSSIZE numocts, OSSIZE bytesPerUnit)

Function Documentation

void rtxPrintToStreamBoolean (OSCTXT *pctxt, const char *name, OSBOOL value)

Prints a boolean value to a print stream.

Table 2.266. Parameters

pctxt	A pointer to a context structure.
name	The name of the variable to print.
value	Boolean value to print.

void rtxPrintToStreamDate (OSCTXT *pctxt, const char *name, con-st OSNumDateTime *pvalue)

Prints a date value to a print stream.

Table 2.267. 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.

void rtxPrintToStreamTime (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)

Prints a time value to a print stream.

Table 2.268. 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.

void rtxPrintToStreamDateTime (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)

Prints a dateTime value to a print stream.

Table 2.269. 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.

void rtxPrintToStreamInteger (OSCTXT *pctxt, const char *name, OSINT32 value)

Prints an integer value to a print stream.

Table 2.270. Parameters

pctxt	A pointer to a context structure.
name	The name of the variable to print.
value	Integer value to print.

void rtxPrintToStreamInt64 (OSCTXT *pctxt, const char *name, OSINT64 value)

Prints a 64-bit integer value to a print stream.

Table 2.271. Parameters

pctxt	A pointer to a context structure.

name	The name of the variable to print.
value	64-bit integer value to print.

void rtxPrintToStreamUnsigned (OSCTXT *pctxt, const char *name, OSUINT32 value)

Prints an unsigned integer value to a print stream.

Table 2.272. Parameters

pctxt	A pointer to a context structure.
name	The name of the variable to print.
value	Unsigned integer value to print.

void rtxPrintToStreamUInt64 (OSCTXT *pctxt, const char *name, OSUINT64 value)

Prints an unsigned 64-bit integer value to a print stream.

Table 2.273. Parameters

pctxt	A pointer to a context structure.
name	The name of the variable to print.
value	Unsigned 64-bit integer value to print.

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.

Table 2.274. 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.

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.

Table 2.275. 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.

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.

Table 2.276. 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.

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.

Table 2.277. 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.

void rtxPrintToStreamCharStr (OSCTXT *pctxt, const char *name, const char *cstring)

Prints an ASCII character string value to a print stream.

Table 2.278. 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.

void rtxPrintToStreamUTF8CharStr (OSCTXT *pctxt, const char *name, const OSUTF8CHAR *cstring)

Prints a UTF-8 encoded character string value to a print stream.

Table 2.279. 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.

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).

Table 2.280. 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).

void rtxPrintToStreamReal (OSCTXT *pctxt, const char *name, OSREAL value)

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

Table 2.281. Parameters

pctxt	A pointer to a context structure.
name	The name of the variable to print.
value	REAL value to print.

void rtxPrintToStreamNull (OSCTXT *pctxt, const char *name)

Prints a NULL value to a print stream.

Table 2.282. Parameters

pctxt	A pointer to a context structure.
name	The name of the variable to print.

void rtxPrintToStreamNVP (OSCTXT *pctxt, const char *name, const OSUTF8NVP *value)

Prints a name-value pair to a print stream.

Table 2.283. 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.

int rtxPrintToStreamFile (OSCTXT *pctxt, const char *filename)

This function prints the contents of a text file to a print stream.

Table 2.284. Parameters

pctxt	A pointer to a context structure.
filename	The name of the text file to print.

Returns: . Status of operation, 0 if success.

void rtxPrintToStreamIndent (OSCTXT *pctxt)

This function prints indentation spaces to a print stream.

void rtxPrintToStreamResetIndent (OSCTXT *pctxt)

This function resets the current indentation level to zero.

Table 2.285. Parameters

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

void rtxPrintToStreamIncrIndent (OSCTXT *pctxt)

This function increments the current indentation level.

Table 2.286. Parameters

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

void rtxPrintToStreamDecrIndent (OSCTXT *pctxt)

This function decrements the current indentation level.

Table 2.287. Parameters

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

void rtxPrintToStreamCloseBrace (OSCTXT *pctxt)

This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.

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.

void rtxHexDumpToStream (OSCTXT *pctxt, const OSOCTET *data, OSSIZE numocts)

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

Table 2.288. 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

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.

Table 2.289. 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).

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.

Table 2.290. 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).

TCP/IP or UDP socket utility functions

Detailed Description

Typedefs

- typedef unsigned long OSIPADDR

Functions

- int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET * pNewSocket, OSIPADDR * destAddr, int * destPort)
- int rtxSocketAddrToStr (OSIPADDR ipAddr, char * pbuf, size_t bufsize)
- int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)
- int rtxSocketClose (OSRTSOCKET socket)
- int rtxSocketConnect (OSRTSOCKET socket, const char * host, int port)
- int rtxSocketConnectTimed (OSRTSOCKET socket, const char * host, int port, int nsecs)
- int rtxSocketCreate (OSRTSOCKET * psocket)
- int rtxSocketCreateUDP (OSRTSOCKET * psocket)
- int rtxSocketGetHost (const char * host, struct in_addr * inaddr)
- int rtxSocketsInit (OSVOIDARG)
- int rtxSocketListen (OSRTSOCKET socket, int maxConnection)
- int rtxSocketParseURL (char * url, char ** protocol, char ** address, int * port)

- int rtxSocketRecv (OSRTSOCKET socket, OSOCTET * pbuf, size_t bufsize)
- int rtxSocketRecvTimed (OSRTSOCKET socket, OSOCTET * pbuf, size_t bufsize, OSUINT32 secs)
- int rtxSocketSelect (int nfds, fd_set * readfds, fd_set * writefds, fd_set * exceptfds, struct timeval * timeout)
- int rtxSocketSend (OSRTSOCKET socket, const OSOCTET * pdata, size_t size)
- int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)
- int rtxSocketStrToAddr (const char * pIPAddrStr, OSIPADDR * pIPAddr)

Macros

- #define OSIPADDR_ANY ((OSIPADDR)0)
- #define OSIPADDR_LOCAL ((OSIPADDR)0x7f000001UL) /* 127.0.0.1 */

Typedef Documentation

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.

Function Documentation

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.

Table 2.291. 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.

int rtxSocketAddrToStr (OSIPADDR ipAddr, char *pbuf, size_t bufsize)

This function converts an IP address to its string representation.

Table 2.292. 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.

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.

Table 2.293. 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.

int rtxSocketClose (OSRTSOCKET socket)

This function closes an existing socket.

Table 2.294. 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.

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.

Table 2.295. 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.

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.

Table 2.296. 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.

int rtxSocketCreate (OSRTSOCKET *psocket)

This function creates a TCP socket.

Table 2.297. 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.

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.

Table 2.298. 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.

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.

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.

Table 2.299. 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.

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.

Table 2.300. 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.

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.

Table 2.301. 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.

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.

Table 2.302. 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.

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.

Table 2.303. 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.

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.

Table 2.304. 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.

int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)

This function turns blocking mode for a socket on or off.

Table 2.305. 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.

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.

Table 2.306. 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.

Input/Output Data Stream Utility Functions

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`

Classes

- struct OSRTSTREAM

Typedefs

- typedef long(* OSRTStreamReadProc
- typedef OSRTStreamReadProc OSRTStreamBlockingReadProc
- typedef long(* OSRTStreamWriteProc
- typedef int(* OSRTStreamFlushProc
- typedef int(* OSRTStreamCloseProc
- typedef int(* OSRTStreamSkipProc
- typedef int(* OSRTStreamMarkProc
- typedef int(* OSRTStreamResetProc
- typedef int(* OSRTStreamGetPosProc
- typedef int(* OSRTStreamSetPosProc
- typedef struct OSRTSTREAM OSRTSTREAM

Functions

- int rtxStreamClose (OSCTXT * pctxt)
- int rtxStreamFlush (OSCTXT * pctxt)
- int rtxStreamLoadInputBuffer (OSCTXT * pctxt, OSSIZE nbytes)
- int rtxStreamInit (OSCTXT * pctxt)
- int rtxStreamInitCtxtBuf (OSCTXT * pctxt)
- int rtxStreamRemoveCtxtBuf (OSCTXT * pctxt)
- long rtxStreamRead (OSCTXT * pctxt, OSOCTET * pbuffer, size_t nocts)
- long rtxStreamRead2 (OSCTXT * pctxt, OSOCTET * pbuffer, size_t nocts, size_t bufSize)
- long rtxStreamReadDirect (OSCTXT * pctxt, OSOCTET * pbuffer, OSSIZE bufSize)
- int rtxStreamSkip (OSCTXT * pctxt, size_t skipBytes)
- long rtxStreamWrite (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)
- int rtxStreamGetIOBytes (OSCTXT * pctxt, size_t * pPos)

- int rtxStreamMark (OSCTXT * pctxt, size_t readAheadLimit)
- int rtxStreamReset (OSCTXT * pctxt)
- OSBOOL rtxStreamMarkSupported (OSCTXT * pctxt)
- OSBOOL rtxStreamIsOpened (OSCTXT * pctxt)
- OSBOOL rtxStreamIsReadable (OSCTXT * pctxt)
- OSBOOL rtxStreamIsWritable (OSCTXT * pctxt)
- int rtxStreamRelease (OSCTXT * pctxt)
- void rtxStreamSetCapture (OSCTXT * pctxt, OSRTMEMBUF * pmembuf)
- OSRTMEMBUF * rtxStreamGetCapture (OSCTXT * pctxt)
- int rtxStreamGetPos (OSCTXT * pctxt, size_t * ppos)
- int rtxStreamSetPos (OSCTXT * pctxt, size_t pos)

Macros

- #define OSRTSTRMF_INPUT 0x0001
- #define OSRTSTRMF_OUTPUT 0x0002
- #define OSRTSTRMF_BUFFERED 0x8000 /* direct-buffer stream */
- #define OSRTSTRMF_UNBUFFERED 0x4000 /* force unbuffered stream */
- #define OSRTSTRMF_POSMARKED 0x2000 /* stream has marked position */
- #define OSRTSTRMF_FIXINMEM 0x1000 /* disable flushing */
- #define OSRTSTRMF_HEXTEXT 0x0800 /* do hex text / binary conversion */
- #define OSRTSTRMF_BUF_INPUT (OSRTSTRMF_INPUT|OSRTSTRMF_BUFFERED)
- #define OSRTSTRMF_BUF_OUTPUT (OSRTSTRMF_OUTPUT|OSRTSTRMF_BUFFERED)
- #define OSRTSTRMID_FILE 1
- #define OSRTSTRMID_SOCKET 2
- #define OSRTSTRMID_MEMORY 3
- #define OSRTSTRMID_BUFFERED 4
- #define OSRTSTRMID_DIRECTBUF 5
- #define OSRTSTRMID_CTXTBUF 6
- #define OSRTSTRMID_ZLIB 7
- #define OSRTSTRMID_USER 1000

- #define OSRTSTRM_K_BUFSIZE 1024
- #define OSRTSTRM_K_INVALIDMARK ((size_t)-1)
- #define OSRTSTREAM_BYTEINDEX (((pctxt)->pStream->flags & OSRTSTRMF_BUFFERED) ? \ ((pctxt)->pStream->bytesProcessed + (pctxt)->buffer.byteIndex) : \ ((pctxt)->pStream->bytesProcessed /* was ioBytes */))
- #define OSRTSTREAM_ID ((pctxt)->pStream->id)
- #define OSRTSTREAM_FLAGS ((pctxt)->pStream->flags)
- #define rtxStreamBlockingRead rtxStreamRead

Typedef Documentation

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 OSRTSTREAM 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.

Table 2.307. 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.

typedef OSRTStreamReadProc OSRTStreamBlockingReadProc

OSRTStreamBlockingReadProc is deprecated. Use OSRTStreamReadProc.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Function Documentation

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.

Table 2.308. Parameters

pctxt	Pointer to a context structure variable which has been initialized for stream operations via a call to `rtxStreamInit`.

int rtxStreamFlush (OSCTXT *pctxt)

This function flushes the output stream and forces any buffered output octets to be written out.

Table 2.309. 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.

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.

Table 2.310. 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.

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.

Table 2.311. 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.

int rtxStreamInitCtxtBuf (OSCTXT *pctxt)

This function initializes a stream to use the context memory buffer for stream buffering.

Table 2.312. 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.

int rtxStreamRemoveCtxtBuf (OSCTXT *pctxt)

This function removes the use of a context memory buffer from a stream.

Table 2.313. 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.

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.

Table 2.314. 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.

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.

Table 2.315. 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.

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.)

Table 2.316. 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.

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.

Table 2.317. 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.

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.

Table 2.318. 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).

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.

Table 2.319. 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.

int rtxStreamReset (OSCTXT *pctxt)

Repositions this stream to the position recorded by the last call to the rtxStreamMark function.

Table 2.320. 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.

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.

Table 2.321. 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.

OSBOOL rtxStreamIsOpened (OSCTXT *pctxt)

Tests if this stream opened (for reading or writing).

Table 2.322. 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.

OSBOOL rtxStreamIsReadable (OSCTXT *pctxt)

Tests if this stream opened for reading.

Table 2.323. 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.

OSBOOL rtxStreamIsWritable (OSCTXT *pctxt)

Tests if this stream opened for writing.

Table 2.324. 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.

int rtxStreamRelease (OSCTXT *pctxt)

This function releases the stream's resources. If it is opened for reading or writing it will be closed.

Table 2.325. 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.

void rtxStreamSetCapture (OSCTXT *pctxt, OSRTMEMBUF *pmem-buf)

This function sets a capture buffer for the stream. This is used to record all data read from the stream.

Table 2.326. 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 dis-able capture if previously set.

OSRTMEMBUF* rtxStreamGetCapture (OSCTXT *pctxt)

This function returns the capture buffer currently assigned to the stream.

Table 2.327. 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.

int rtxStreamGetPos (OSCTXT *pctxt, size_t *ppos)

Get the current position in input stream.

Table 2.328. 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.

int rtxStreamSetPos (OSCTXT *pctxt, size_t pos)

Set the current position in input stream.

Table 2.329. 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.

Macro Definition Documentation

#define rtxStreamBlockingRead

rtxStreamBlockingRead is deprecated. Use rtxStreamRead.

Definition at line 344 of file rtxStream.h

The Documentation for this define was generated from the following file:

- rtxStream.h

File stream functions.

Detailed Description

File stream functions are used for stream operations with files.

Functions

- int rtxStreamFileAttach (OSCTXT * pctxt, FILE * pFile, OSUINT16 flags)

- int rtxStreamFileOpen (OSCTXT * pctxt, const char * pFilename, OSUINT16 flags)
- int rtxStreamFileCreateReader (OSCTXT * pctxt, const char * pFilename)
- int rtxStreamFileCreateWriter (OSCTXT * pctxt, const char * pFilename)

Function Documentation

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.

Table 2.330. 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.

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.

Table 2.331. 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.

int rtxStreamFileCreateReader (OSCTXT *pctxt, const char *pFilename)

This function creates an input file stream using the specified file name.

Table 2.332. 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.

int rtxStreamFileCreateWriter (OSCTXT *pctxt, const char *pFile-name)

This function creates an output file stream using the file name.

Table 2.333. 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.

Memory stream functions.

Detailed Description

Memory stream functions are used for memory stream operations.

Functions

- int rtxStreamMemoryCreate (OSCTXT * pctxt, OSUINT16 flags)
- int rtxStreamMemoryAttach (OSCTXT * pctxt, OSOCTET * pMemBuf, size_t bufSize, OSUINT16 flags)
- OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT * pctxt, size_t * pSize)
- int rtxStreamMemoryCreateReader (OSCTXT * pctxt, OSOCTET * pMemBuf, size_t bufSize)
- int rtxStreamMemoryCreateWriter (OSCTXT * pctxt, OSOCTET * pMemBuf, size_t bufSize)
- int rtxStreamMemoryResetWriter (OSCTXT * pctxt)

Function Documentation

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.

Table 2.334. 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.

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.

Table 2.335. 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.

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.

Table 2.336. 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.

int rtxStreamMemoryCreateReader (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)

This function creates an input memory stream using the specified buffer.

Table 2.337. 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.

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.

Table 2.338. 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.

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.

Table 2.339. 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.

Socket stream functions.

Detailed Description

Socket stream functions are used for socket stream operations.

Functions

- int rtxStreamSocketAttach (OSCTXT * pctxt, OSRTSOCKET socket, OSUINT16 flags)
- int rtxStreamSocketClose (OSCTXT * pctxt)
- int rtxStreamSocketCreateWriter (OSCTXT * pctxt, const char * host, int port)

- int rtxStreamSocketSetOwnership (OSCTXT * pctxt, OSBOOL ownSocket)
- int rtxStreamSocketSetReadTimeout (OSCTXT * pctxt, OSUINT32 nsecs)

Function Documentation

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.

Table 2.340. 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.

int rtxStreamSocketClose (OSCTXT *pctxt)

This function closes a socket stream.

Table 2.341. 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.

int rtxStreamSocketCreateWriter (OSCTXT *pctxt, const char *host, int port)

This function opens a socket stream for writing.

Table 2.342. 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.

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.

Table 2.343. 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.

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.

Table 2.344. 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.

Doubly-Linked List Utility Functions

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.

Classes

- struct OSRTDListNode
- struct OSRTDList
- struct OSRTDListBuf
- struct OSRTDListUTF8StrNode

Typedefs

- typedef struct OSRTDListNode OSRTDListNode

- typedef struct OSRTDList OSRTDList
- typedef struct OSRTDListBuf OSRTDListBuf
- typedef struct OSRTDListUTF8StrNode OSRTDListUTF8StrNode
- typedef int(* PEqualsFunc

Functions

- void rtxDListInit (OSRTDList * pList)
- OSRTDListNode * rtxDListAppend (struct OSCTXT * pctxt, OSRTDList * pList, void * pData)
- OSRTDListNode * rtxDListAppendCharArray (struct OSCTXT * pctxt, OSRTDList * pList, size_t length, char * pData)
- OSRTDListNode * rtxDListAppendNode (OSRTDList * pList, OSRTDListNode * pListNode)
- OSRTDListNode * rtxDListInsert (struct OSCTXT * pctxt, OSRTDList * pList, OSSIZE idx, void * pData)
- OSRTDListNode * rtxDListInsertNode (OSRTDList * pList, OSSIZE idx, OSRTDListNode * pListNode)
- OSRTDListNode * rtxDListInsertBefore (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node, void * pData)
- OSRTDListNode * rtxDListInsertAfter (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node, void * pData)
- OSRTDListNode * rtxDListFindByIndex (const OSRTDList * pList, OSSIZE idx)
- OSRTDListNode * rtxDListFindByData (const OSRTDList * pList, void * data)
- OSRTDListNode * rtxDListFindFirstData (const OSRTDList * pList)
- int rtxDListFindIndexByData (const OSRTDList * pList, void * data)
- void rtxDListFreeNode (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node)
- void rtxDListRemove (OSRTDList * pList, OSRTDListNode * node)
- void rtxDListFreeNodes (struct OSCTXT * pctxt, OSRTDList * pList)
- void rtxDListFreeAll (struct OSCTXT * pctxt, OSRTDList * pList)
- int rtxDListToArray (struct OSCTXT * pctxt, OSRTDList * pList, void ** ppArray, OSSIZE * pElemCount, OSSIZE elemSize)
- int rtxDListToPointerArray (struct OSCTXT * pctxt, OSRTDList * pList, void *** ppArray, OSSIZE * pElemCount)
- int rtxDListAppendArray (struct OSCTXT * pctxt, OSRTDList * pList, void * pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListAppendArrayCopy (struct OSCTXT * pctxt, OSRTDList * pList, const void * pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListToUTF8Str (struct OSCTXT * pctxt, OSRTDList * pList, OSUTF8CHAR ** ppstr, char sep)

- OSRTDListNode * rtxDListInsertSorted (struct OSCTXT * pctxt, OSRTDList * pList, void * pData, PEqualsFunc equalsFunc, void * sortCtxt)
- OSRTDListNode * rtxDListInsertNodeSorted (OSRTDList * pList, OSRTDListNode * pListNode, PEqualsFunc equalsFunc, void * sortCtxt)

Macros

- #define DLISTBUF_SEG 16

Function Documentation

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).

Table 2.345. Parameters

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

OSRTDListNode* rtxDListAppend (struct OSCTXT *pctxt, OSRT-DList *pList, void *pData)

This function appends an item to the linked list structure. The data item is passed into the function as a void pointer that can point to 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.

Table 2.346. 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.

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.

Table 2.347. 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.

OSRTDListNode* rtxDListAppendNode (OSRTDList *pList, OSRT-DListNode *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.

Table 2.348. 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.

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.

Table 2.349. 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.

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.

Table 2.350. 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.

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.

Table 2.351. 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.

OSRTDListNode* rtxDListFindByIndex (const OSRTDList *pList, OSSIZE idx)

This function will return the node pointer of the indexed entry in the list.

Table 2.352. 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.

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.

Table 2.353. 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.

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.

Table 2.354. Parameters

pList	A pointer to a linked list structure.

Returns: . A pointer to an allocated linked list node structure.

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.

Table 2.355. 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.

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.

Table 2.356. 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.

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.

Table 2.357. Parameters

pList	A pointer to a linked list structure.
node	Pointer to the list node to be removed.

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.

Table 2.358. Parameters

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

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.

Table 2.359. Parameters

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

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.

Table 2.360. 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.

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.

Table 2.361. 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.

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.

Table 2.362. 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.

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.

Table 2.363. 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.

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.

Table 2.364. 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.

Linked List Utility Functions

Detailed Description

Singly linked list structures have only a single link pointer and can therefore only be traversed in a single direction (forward). The node structures consume less memory than those of a doubly linked list.

Another difference between the singly linked list implementation and doubly linked lists is that the singly linked list uses conventional memory allocation functions (C malloc and free) for less storage instead of the rtxMem functions. Therefore, it is not a requirement to have an initialized context structure to work with these lists. However, performance may suffer if the lists become large due to the use of non-optimized memory management.

Classes

- struct _OSRTSListNode
- struct _OSRTSList

Typedefs

- typedef struct _OSRTSListNode OSRTSListNode
- typedef struct _OSRTSList OSRTSList

Functions

- void rtxSListInit (OSRTSList * pList)
- void rtxSListInitEx (OSCTXT * pctxt, OSRTSList * pList)
- void rtxSListFree (OSRTSList * pList)
- void rtxSListFreeAll (OSRTSList * pList)
- OSRTSList * rtxSListCreate (OSVOIDARG)
- OSRTSList * rtxSListCreateEx (OSCTXT * pctxt)
- OSRTSListNode * rtxSListAppend (OSRTSList * pList, void * pData)
- OSBOOL rtxSListFind (OSRTSList * pList, void * pData)
- void rtxSListRemove (OSRTSList * pList, void * pData)

Function Documentation

void rtxSListInit (OSRTSList *pList)

This function initializes a singly linked list structure. It sets the number of elements to zero and sets all internal pointer values to NULL.

Table 2.365. Parameters

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

void rtxSListInitEx (OSCTXT *pctxt, OSRTSList *pList)

This function is similar to rtxSListInit but it also sets the pctxt (pointer to a context structure member of OSRTSList structure). This context will be used for further memory allocations; otherwise the standard malloc is used.

Table 2.366. Parameters

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

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

void rtxSListFree (OSRTSList *pList)

This function removes all nodes from the linked list structure and releases memory that was allocated for storing node structures (OSRTSListNode). The data will not be freed.

Table 2.367. Parameters

pList	A pointer to a linked list onto which the data item is to be appended.

void rtxSListFreeAll (OSRTSList *pList)

This function removes all nodes from the linked list structure and releases memory that was allocated for storing node structures (OSRTSListNode). It also frees the data structures which are assumed to have been allocated using the rtxMemAlloc function.

Table 2.368. Parameters

pList	A pointer to a linked list onto which the data item is to be appended.

OSRTSList* rtxSListCreate (OSVOIDARG)

This function creates a new linked list structure. It allocates memory for the structure and calls rtxSListInit to initialize the structure.

Returns: . A pointer to the allocated linked list structure.

OSRTSList* rtxSListCreateEx (OSCTXT *pctxt)

The rtxSListAppend function appends an item to linked list structure. The data is passed into the function as a void that can point to an object of any type.

Table 2.369. Parameters

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

Returns: . A pointer to the allocated linked list structure.

OSRTSListNode* rtxSListAppend (OSRTSList *pList, void *pData)

This function appends an item to a linked list structure. The data item is passed into the function as a void parameter that can point to an object of any type.

Table 2.370. Parameters

pList	A pointer to a linked list onto which the data item is to be appended.
pData	A pointer to a data item to be appended to the list.

Returns: . A pointer to the allocated linked list structure.

OSBOOL rtxSListFind (OSRTSList *pList, void *pData)

This function finds an item in the linked list structure. The data item is passed into the function as a void pointer that can point to an object of any type. If the appropriate node is found in the list this function returns TRUE, otherwise FALSE.

Table 2.371. Parameters

pList	A pointer to a linked list onto which the data item is to be appended.
pData	A pointer to a data item to be appended to the list.

Returns: . TRUE, if the node is found, otherwise FALSE.

void rtxSListRemove (OSRTSList *pList, void *pData)

This function finds an item in the linked list structure and removes it from the list. The data item is passed into the function as a void pointer that can point to an object of any type.

Table 2.372. Parameters

pList	A pointer to a linked list onto which the data item is to be appended.
pData	A pointer to a data item to be appended to the list.

Stack Utility Functions

Detailed Description

This is a simple stack structure with supporting push and pop functions. Different initialization routines are provided to permit different memory management schemes.

Classes

- struct _OSRTStack

Typedefs

- typedef struct _OSRTStack OSRTStack

Functions

- OSRTStack * rtxStackCreate (struct OSCTXT * pctxt)

- void rtxStackInit (struct OSCTXT * pctxt, OSRTStack * pStack)
- void * rtxStackPop (OSRTStack * pStack)
- int rtxStackPush (OSRTStack * pStack, void * pData)
- void * rtxStackPeek (OSRTStack * pStack)

Macros

- #define rtxStackIsEmpty (OSBOOL)((stack).dlist.count == 0)

Function Documentation

OSRTStack* rtxStackCreate (struct OSCTXT *pctxt)

This function creates a new stack structure. It allocates memory for the structure and calls rtxStackInit to initialize the structure.

Table 2.373. Parameters

pctxt	A pointer to the context with which the stack is associated.

Returns: . A pointer to an allocated stack structure.

void rtxStackInit (struct OSCTXT *pctxt, OSRTStack *pStack)

This function initializes a stack structure. It sets the number of elements to zero and sets all internal pointer values to NULL.

Table 2.374. Parameters

pctxt	A pointer to the context with which the stack is associated.
pStack	A pointer to a stack structure to be initialized.

void* rtxStackPop (OSRTStack *pStack)

This function pops an item off the stack.

Table 2.375. Parameters

pStack	The pointer to the stack structure from which the value is to be popped. Pointer to the up-dated stack structure.

Returns: . The pointer to the item popped from the stack

int rtxStackPush (OSRTStack *pStack, void *pData)

This function pushes an item onto the stack.

Table 2.376. Parameters

pStack	A pointer to the structure onto which the data item is to be pushed. The pointer updated to the stack structure
pData	A pointer to the data item to be pushed on the stack.

Returns: . Completion status of operation:

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

void* rtxStackPeek (OSRTStack *pStack)

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

Table 2.377. Parameters

pStack	A pointer to the structure onto which the data item is to be pushed. The pointer updated to the stack structure

Returns: . Pointer to data item at top of stack or NULL if stack empty.

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

Macro Definition Documentation

#define rtxStackIsEmpty

This macro tests if the stack is empty.

Table 2.378. Parameters

stack	Stack structure variable to be tested.

Definition at line 82 of file rtxStack.h

The Documentation for this define was generated from the following file:

- rtxStack.h

Pattern matching functions

Detailed Description

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

Functions

- OSBOOL rtxMatchPattern (OSCTXT * pctxt, const OSUTF8CHAR * text, const OSUTF8CHAR * pattern)
- OSBOOL rtxMatchPattern2 (OSCTXT * pctxt, const OSUTF8CHAR * pattern)
- void rtxFreeRegexpCache (OSCTXT * pctxt)

Function Documentation

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.

Table 2.379. Parameters

pctxt	Pointer to context structure.
text	Text to be matched.
pattern	Regular expression.

Returns: . Boolean result.

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.)

Diagnostic trace functions

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.

Classes

- struct OSRTPrintStream
- struct OSRTStrBuf

Enumerations

- enum OSRTDiagTraceLevel {
  OSRTDiagError,

OSRTDiagWarning,
OSRTDiagInfo,
OSRTDiagDebug
}

Typedefs

- typedef void(* rtxPrintCallback
- typedef struct OSRTPrintStream OSRTPrintStream

Variables

- OSRTPrintStream g_PrintStream

Functions

- int rtxSetPrintStream (OSCTXT * pctxt, rtxPrintCallback myCallback, void * pStrmInfo)
- int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void * pStrmInfo)
- int rtxPrintToStream (OSCTXT * pctxt, const char * fmtspec, ...)
- int rtxDiagToStream (OSCTXT * pctxt, const char * fmtspec, va_list arglist)
- int rtxPrintStreamRelease (OSCTXT * pctxt)
- void rtxPrintStreamToStdoutCB (void * pPrntStrmInfo, const char * fmtspec, va_list arglist)
- void rtxPrintStreamToFileCB (void * pPrntStrmInfo, const char * fmtspec, va_list arglist)
- void rtxPrintStreamToStringCB (void * pPrntStrmInfo, const char * fmtspec, va_list arglist)
- OSBOOL rtxDiagEnabled (OSCTXT * pctxt)
- OSBOOL rtxSetDiag (OSCTXT * pctxt, OSBOOL value)
- OSBOOL rtxSetGlobalDiag (OSBOOL value)
- void rtxDiagPrint (OSCTXT * pctxt, const char * fmtspec, ...)
- void rtxDiagStream (OSCTXT * pctxt, const char * fmtspec, ...)
- void rtxDiagHexDump (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)
- void rtxDiagStreamHexDump (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)
- void rtxDiagPrintChars (OSCTXT * pctxt, const char * data, size_t nchars)
- void rtxDiagStreamPrintChars (OSCTXT * pctxt, const char * data, size_t nchars)
- void rtxDiagStreamPrintBits (OSCTXT * pctxt, const char * descr, const OSOCTET * data, size_t bitIndex, size_t nbits)
- void rtxDiagSetTraceLevel (OSCTXT * pctxt, OSRTDiagTraceLevel level)
- OSBOOL rtxDiagTraceLevelEnabled (OSCTXT * pctxt, OSRTDiagTraceLevel level)

Macros

- #define RTDIAG1
- #define RTDIAG2
- #define RTDIAG3
- #define RTDIAG4
- #define RTDIAG5
- #define RTDIAGU
- #define RTHEXDUMP
- #define RTDIAGCHARS
- #define RTDIAGSTRM2
- #define RTDIAGSTRM3
- #define RTDIAGSTRM4
- #define RTDIAGSTRM5
- #define RTHEXDUMPSTRM
- #define RTDIAGSCHARS

Typedef Documentation

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.

Variable Documentation

OSRTPrintStream g_PrintStream

Global PrintStream

Function Documentation

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.

Table 2.380. 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

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.

Table 2.381. 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

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.

Table 2.382. 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

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.

Table 2.383. 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

int rtxPrintStreamRelease (OSCTXT *pctxt)

This function releases the memory held by PrintStream in the context

Table 2.384. Parameters

pctxt	Pointer to a context for which the memory has to be released.

Returns: . Completion status, 0 on success, negative value on failure

void rtxPrintStreamToStdoutCB (void *pPrntStrmInfo, const char *fmtspec, va_list arglist)

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

Table 2.385. 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.

void rtxPrintStreamToFileCB (void *pPrntStrmInfo, const char *fmt-spec, va_list arglist)

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

Table 2.386. 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.

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

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

Table 2.387. 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 OSRTStr-Buf 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.

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.

Table 2.388. Parameters

pctxt	Pointer to context structure.

Returns: . Boolean result.

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.

Table 2.389. 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.

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.

Table 2.390. Parameters

value	Boolean switch: TRUE turns tracing on, FALSE off.

Returns: . Prior setting of the diagnostic trace switch in the context.

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.

Table 2.391. 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.

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

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.

Table 2.392. Parameters

pctxt	Pointer to context structure.
data	Start address of memory to dump.
numocts	Number of bytes to dump.

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.

Table 2.393. Parameters

pctxt	Pointer to context structure.
data	Start address of memory to dump.
numocts	Number of bytes to dump.

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.

Table 2.394. Parameters

pctxt	Pointer to context structure.
data	Start address of character string.
nchars	Number of characters to dump (this assumes 1-byte chars).

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.

Table 2.395. Parameters

pctxt	Pointer to context structure.
data	Start address of character string.
nchars	Number of characters to dump (this assumes 1-byte chars).

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.

Table 2.396. 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

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.

Table 2.397. Parameters

pctxt	Pointer to context structure.
level	Trace level to be set.

OSBOOL rtxDiagTraceLevelEnabled (OSCTXT *pctxt, OSRTDiag-TraceLevel level)

This function tests if a given trace level is enabled.

Table 2.398. Parameters

pctxt	Pointer to context structure.
level	Trace level to check.

Returns: . True if enabled.

Error Formatting and Print Functions

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.

Typedefs

- typedef int(* OSErrCbFunc

Functions

- OSBOOL rtxErrAddCtxtBufParm (OSCTXT * pctxt)
- OSBOOL rtxErrAddDoubleParm (OSCTXT * pctxt, double errParm)
- OSBOOL rtxErrAddErrorTableEntry (const char *const * ppStatusText, OSINT32 minErrCode, OSINT32 max-ErrCode)
- OSBOOL rtxErrAddElemNameParm (OSCTXT * pctxt)
- OSBOOL rtxErrAddIntParm (OSCTXT * pctxt, int errParm)

- OSBOOL rtxErrAddInt64Parm (OSCTXT * pctxt, OSINT64 errParm)
- OSBOOL rtxErrAddSizeParm (OSCTXT * pctxt, OSSIZE errParm)
- OSBOOL rtxErrAddStrParm (OSCTXT * pctxt, const char * pErrParm)
- OSBOOL rtxErrAddStrnParm (OSCTXT * pctxt, const char * pErrParm, OSSIZE nchars)
- OSBOOL rtxErrAddUIntParm (OSCTXT * pctxt, unsigned int errParm)
- OSBOOL rtxErrAddUInt64Parm (OSCTXT * pctxt, OSUINT64 errParm)
- void rtxErrAssertionFailed (const char * conditionText, int lineNo, const char * fileName)
- const char * rtxErrFmtMsg (OSRTErrInfo * pErrInfo, char * bufp, OSSIZE bufsiz)
- void rtxErrFreeParms (OSCTXT * pctxt)
- char * rtxErrGetText (OSCTXT * pctxt, char * pBuf, OSSIZE * pBufSize)
- char * rtxErrGetTextBuf (OSCTXT * pctxt, char * pbuf, OSSIZE bufsiz)
- char * rtxErrGetMsgText (OSCTXT * pctxt)
- char * rtxErrGetMsgTextBuf (OSCTXT * pctxt, char * pbuf, OSSIZE bufsiz)
- OSRTErrInfo * rtxErrNewNode (OSCTXT * pctxt)
- void rtxErrInit (OSVOIDARG)
- int rtxErrReset (OSCTXT * pctxt)
- int rtxErrResetErrInfo (OSCTXT * pctxt)
- void rtxErrLogUsingCB (OSCTXT * pctxt, OSErrCbFunc cb, void * cbArg_p)
- void rtxErrPrint (OSCTXT * pctxt)
- void rtxErrPrintElement (OSRTErrInfo * pErrInfo)
- int rtxErrSetData (OSCTXT * pctxt, int status, const char * module, int lineno)
- int rtxErrSetNewData (OSCTXT * pctxt, int status, const char * module, int lineno)
- void rtxErrSetNonFatal (OSCTXT * pctxt)
- int rtxErrCheckNonFatal (OSCTXT * pctxt)
- int rtxErrGetFirstError (const OSCTXT * pctxt)
- int rtxErrGetLastError (const OSCTXT * pctxt)
- OSSIZE rtxErrGetErrorCnt (const OSCTXT * pctxt)
- int rtxErrGetStatus (const OSCTXT * pctxt)
- int rtxErrResetLastErrors (OSCTXT * pctxt, OSSIZE errorsToReset)

- int rtxErrCopy (OSCTXT * pDestCtxt, const OSCTXT * pSrcCtxt)
- int rtxErrAppend (OSCTXT * pDestCtxt, const OSCTXT * pSrcCtxt)
- int rtxErrInvUIntOpt (OSCTXT * pctxt, OSUINT32 ident)

Macros

- #define LOG_RTERR rtxErrSetData(pctxt,stat,__FILE__,__LINE__)
- #define LOG_RTERRNEW rtxErrSetNewData(pctxt,stat,__FILE__,__LINE__)
- #define RETURN_RTERR return LOG_RTERR(pctxt, stat)
- #define OSRTASSERT if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }
- #define OSRTCHECKPARAM if (condition) { /* do nothing */ }
- #define LOG_RTERR1 (a,LOG_RTERR (pctxt, stat),stat)
- #define LOG_RTERRNEW1 (a,LOG_RTERRNEW (pctxt, stat),stat)
- #define LOG_RTERR2 (a,b,LOG_RTERR (pctxt, stat),stat)
- #define LOG_RTERRNEW2 (a,b,LOG_RTERRNEW (pctxt, stat),stat)
- #define LOG_RTERR_AND_FREE_MEM rtxMemFreePtr ((ctxt_p),(mem_p)), LOG_RTERR(ctxt_p, stat)

Function Documentation

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.

Table 2.399. Parameters

pctxt	A pointer to a context structure.

Returns: . The status of the operation (TRUE if the parameter was sucessfully added).

OSBOOL rtxErrAddDoubleParm (OSCTXT *pctxt, double errParm)

This function adds a double parameter to an error information structure.

Table 2.400. 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).

OSBOOL rtxErrAddErrorTableEntry (const char *const *ppStatus-Text, 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.

Table 2.401. 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).

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.

Table 2.402. Parameters

pctxt	A pointer to a context structure.

Returns: . The status of the operation (TRUE if the parameter was sucessfully added).

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.

Table 2.403. 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).

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.

Table 2.404. 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).

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.

Table 2.405. 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).

OSBOOL rtxErrAddStrParm (OSCTXT *pctxt, const char *pErrParm)

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

Table 2.406. 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).

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.

Table 2.407. 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).

OSBOOL rtxErrAddUIntParm (OSCTXT *pctxt, unsigned int errParm)

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

Table 2.408. 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).

OSBOOL rtxErrAddUInt64Parm (OSCTXT *pctxt, OSUINT64 errParm)

This function adds an unsigned 64-bit integer parameter to an error information structure.

Table 2.409. 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).

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.

Table 2.410. 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.

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.

Table 2.411. 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).

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.

Table 2.412. Parameters

pctxt	A pointer to a context structure.

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.

Table 2.413. 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.

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.

Table 2.414. 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).

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.

Table 2.415. 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.

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.

Table 2.416. 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).

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.

Table 2.417. Parameters

pctxt	A pointer to a context structure.

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

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.

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.

Table 2.418. Parameters

pctxt	A pointer to a context structure.

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.

Table 2.419. Parameters

pctxt	A pointer to a context structure.

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.

Table 2.420. 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.

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.

Table 2.421. Parameters

pctxt	A pointer to a context structure.

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.

Table 2.422. Parameters

pErrInfo	A pointer to an error information element.

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.

Table 2.423. 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.

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.

Table 2.424. 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.

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.

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.

Table 2.425. Parameters

pctxt	A pointer to a context structure.

Returns: . An error status or zero.

int rtxErrGetFirstError (const OSCTXT *pctxt)

This function returns the error code, stored in the first error record.

Table 2.426. Parameters

pctxt	A pointer to a context structure.

Returns: . The first status code; zero if no error records exist.

int rtxErrGetLastError (const OSCTXT *pctxt)

This function returns the error code, stored in the last error record.

Table 2.427. Parameters

pctxt	A pointer to a context structure.

Returns: . The last status code; zero if no error records exist.

OSSIZE rtxErrGetErrorCnt (const OSCTXT *pctxt)

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

Table 2.428. Parameters

pctxt	A pointer to a context structure.

Returns: . The total number of error records in the context.

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.

Table 2.429. Parameters

pctxt	A pointer to a context structure.

Returns: . Status code corresponding to errors in the context.

int rtxErrResetLastErrors (OSCTXT *pctxt, OSSIZE errorsToReset)

This function resets last 'errorsToReset` errors in the context.

Table 2.430. 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

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.

Table 2.431. 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

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.

Table 2.432. 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

int rtxErrInvUIntOpt (OSCTXT *pctxt, OSUINT32 ident)

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

Table 2.433. Parameters

pctxt	Pointer to context structure.
ident	Identifier which was found to be invalid.

Macro Definition Documentation

#define LOG_RTERR

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.

Table 2.434. Parameters

pctxt	A pointer to a context structure.
stat	Error status value from `rtxErrCodes.h`

Definition at line 79 of file rtxError.h

The Documentation for this define was generated from the following file:

- rtxError.h

#define OSRTASSERT

This macro is used to check an assertion. This is a condition that is expected to be true. The `rtxErrAssertion-Failed` 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.

Table 2.435. Parameters

condition	Condition to check (for example, "(ptr != NULL)")

Definition at line 104 of file rtxError.h

The Documentation for this define was generated from the following file:

- rtxError.h

#define OSRTCHECKPARAM

This macro check a condition but takes no action. Its main use is to supress VC++ level 4 "argument not used" warnings.

Table 2.436. Parameters

condition	Condition to check (for example, "(ptr != NULL)")

Definition at line 113 of file rtxError.h

The Documentation for this define was generated from the following file:

- rtxError.h

#define LOG_RTERR_AND_FREE_MEM

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

Table 2.437. 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

The Documentation for this define was generated from the following file:

- rtxError.h

Run-time error status codes.

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.

Macros

- #define RT_OK 0
- #define RT_OK_FRAG 2
- #define RTERR_BUFOVFLW -1
- #define RTERR_ENDOFBUF -2
- #define RTERR_IDNOTFOU -3
- #define RTERR_INVENUM -4
- #define RTERR_SETDUPL -5

- #define RTERR_SETMISRQ -6
- #define RTERR_NOTINSET -7
- #define RTERR_SEQOVFLW -8
- #define RTERR_INVOPT -9
- #define RTERR_NOMEM -10
- #define RTERR_INVHEXS -11
- #define RTERR_INVREAL -12
- #define RTERR_STROVFLW -13
- #define RTERR_BADVALUE -14
- #define RTERR_TOODEEP -15
- #define RTERR_CONSVIO -16
- #define RTERR_ENDOFFILE -17
- #define RTERR_INVUTF8 -18
- #define RTERR_OUTOFBND -19
- #define RTERR_INVPARAM -20
- #define RTERR_INVFORMAT -21
- #define RTERR_NOTINIT -22
- #define RTERR_TOOBIG -23
- #define RTERR_INVCHAR -24
- #define RTERR_XMLSTATE -25
- #define RTERR_XMLPARSE -26
- #define RTERR_SEQORDER -27
- #define RTERR_FILNOTFOU -28
- #define RTERR_READERR -29
- #define RTERR_WRITEERR -30
- #define RTERR_INVBASE64 -31
- #define RTERR_INVSOCKET -32
- #define RTERR_INVATTR -33
- #define RTERR_REGEXP -34
- #define RTERR_PATMATCH -35

- #define RTERR_ATTRMISRQ -36
- #define RTERR_HOSTNOTFOU -37
- #define RTERR_HTTPERR -38
- #define RTERR_SOAPERR -39
- #define RTERR_EXPIRED -40
- #define RTERR_UNEXPELEM -41
- #define RTERR_INVOCCUR -42
- #define RTERR_INVMSGBUF -43
- #define RTERR_DECELEMFAIL -44
- #define RTERR_DECATTRFAIL -45
- #define RTERR_STRMINUSE -46
- #define RTERR_NULLPTR -47
- #define RTERR_FAILED -48
- #define RTERR_ATTRFIXEDVAL -49
- #define RTERR_MULTIPLE -50
- #define RTERR_NOTYPEINFO -51
- #define RTERR_ADDRINUSE -52
- #define RTERR_CONNRESET -53
- #define RTERR_UNREACHABLE -54
- #define RTERR_NOCONN -55
- #define RTERR_CONNREFUSED -56
- #define RTERR_INVSOCKOPT -57
- #define RTERR_SOAPFAULT -58
- #define RTERR_MARKNOTSUP -59
- #define RTERR_NOTSUPP -60 /* feature is not supported */
- #define RTERR_UNBAL -61
- #define RTERR_EXPNAME -62
- #define RTERR_UNICODE -63
- #define RTERR_INVBOOL -64
- #define RTERR_INVNULL -65

- #define RTERR_INVLEN -66
- #define RTERR_UNKNOWNIE -67
- #define RTERR_NOTALIGNED -68
- #define RTERR_EXTRDATA -69
- #define RTERR_INVMAC -70
- #define RTERR_NOSECPARAMS -71
- #define RTERR_COPYFAIL -72
- #define RTERR_PARSEFAIL -73
- #define RTERR_VALCMPERR -74
- #define RTERR_BUFCMPERR -75
- #define RTERR_INVBITS -76
- #define RTERR_RLM -77
- #define RTERR_NOCODEC -78
- #define RTERR_WOULDBLOCK -79
- #define RTERR_NODATA -80
- #define RTERR_MBEDTLS -81
- #define RTERR_INTOVFLOW -82
- #define RTERR_ABORT_REQUESTED -83

Macro Definition Documentation

#define RT_OK

Normal completion status.

Definition at line 46 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RT_OK_FRAG

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_BUFOVFLW

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_ENDOFBUF

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_IDNOTFOU

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVENUM

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_SETDUPL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_SETMISRQ

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOTINSET

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_SEQOVFLW

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVOPT

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOMEM

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVHEXS

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVREAL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_STROVFLW

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_BADVALUE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_TOODEEP

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_CONSVIO

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_ENDOFFILE

Unexpected end-of-file error. This status code is returned when an unexpected end-of-file condition is detected on decode. It is similar to the ENDOFBUF error code described above except that in this case, decoding is being done from a file stream instead of from a memory buffer.

Definition at line 193 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVUTF8

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_OUTOFBND

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVPARAM

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVFORMAT

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOTINIT

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_TOOBIG

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVCHAR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_XMLSTATE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_XMLPARSE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_SEQORDER

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_FILNOTFOU

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_READERR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_WRITEERR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVBASE64

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVSOCKET

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVATTR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_REGEXP

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_PATMATCH

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_ATTRMISRQ

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_HOSTNOTFOU

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_HTTPERR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_SOAPERR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_EXPIRED

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_UNEXPELEM

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVOCCUR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVMSGBUF

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_DECELEMFAIL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_DECATTRFAIL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_STRMINUSE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NULLPTR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_FAILED

General failure. Low level call returned error.

Definition at line 433 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_ATTRFIXEDVAL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_MULTIPLE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOTYPEINFO

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_ADDRINUSE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_CONNRESET

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_UNREACHABLE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOCONN

Not connected. This status code is returned when an operation is issued on an unconnected socket.

Definition at line 489 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_CONNREFUSED

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVSOCKOPT

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

Definition at line 501 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_SOAPFAULT

This error is returned when decoded SOAP envelope is fault message

Definition at line 509 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_MARKNOTSUP

This error is returned when an attempt is made to mark a stream position on a stream type that does not support it.

Definition at line 518 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOTSUPP

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_UNBAL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_EXPNAME

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_UNICODE

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

Definition at line 549 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVBOOL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVNULL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVLEN

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_UNKNOWNIE

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOTALIGNED

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_EXTRDATA

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVMAC

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOSECPARAMS

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_COPYFAIL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_PARSEFAIL

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_VALCMPERR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_BUFCMPERR

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INVBITS

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_RLM

RLM error encounterd.

Definition at line 641 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NOCODEC

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

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_WOULDBLOCK

A non-blocking socket could not return any data without blocking.

Definition at line 652 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_NODATA

CDR or message file contains no data records.

Definition at line 657 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_MBEDTLS

Error returned from MBEDTLS function.

Definition at line 662 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_INTOVFLOW

Integer value overflow

Definition at line 667 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

#define RTERR_ABORT_REQUESTED

User event handler calls for abort.

Definition at line 672 of file rtxErrCodes.h

The Documentation for this define was generated from the following file:

- rtxErrCodes.h

Chapter 3. Class Documentation

_OSRTBufLocDescr struct Reference

```
#include <rtxBuffer.h>
```

Public Attributes

- OSSIZE numocts
- OSSIZE offset

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

_OSRTSList struct Reference

```
#include <rtxSList.h>
```

Public Attributes

- OSSIZE count
- OSRTSListNode * head
- OSRTSListNode * tail
- struct OSCTXT * pctxt

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 68 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

Member Data Documentation

OSSIZE _OSRTSList::count

Count of items in the list.

Definition at line 69 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

OSRTSListNode* _OSRTSList::head

Pointer to first entry in list.

Definition at line 70 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

OSRTSListNode* _OSRTSList::tail

Pointer to last entry in list.

Definition at line 71 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

_OSRTSListNode struct Reference

```
#include <rtxSList.h>
```

Public Attributes

- void * data
- struct _OSRTSListNode * next

Detailed Description

This structure is used to hold a single data item within the list. It contains a void pointer to point at any type of data item and forward pointer to the next entry in the list.

Definition at line 55 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

Member Data Documentation

void* _OSRTSListNode::data

Pointer to list data item.

Definition at line 56 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

struct _OSRTSListNode* _OSRTSListNode::next

Pointer to next node in list.

Definition at line 57 of file rtxSList.h

The Documentation for this struct was generated from the following file:

- rtxSList.h

_OSRTStack struct Reference

```
#include <rtxStack.h>
```

Public Attributes

- struct OSCTXT * pctxt
- OSRTDList dlist

Detailed Description

This is the main stack structure. It uses a linked list structure.

Definition at line 48 of file rtxStack.h

The Documentation for this struct was generated from the following file:

- rtxStack.h

DirBufDesc struct Reference

Public Attributes

- OSCTXT * pctxt
- OSRTSTREAM * pUnderStream
- size_t savedIndex

OSBigInt struct Reference

Public Attributes

- OSSIZE numocts
- OSOCTET * mag

- int sign
- OSSIZE allocated
- OSBOOL dynamic

OSBufferIndex struct Reference

`#include <rtxContext.h>`

Public Attributes

- OSSIZE byteIndex
- OSINT16 bitOffset

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

OSCTXT struct Reference

`#include <rtxContext.h>`

Public Attributes

- void * pMemHeap
- OSRTBuffer buffer
- OSRTBufSave savedInfo
- OSRTErrInfoList errInfo
- OSUINT32 initCode
- OSRTFLAGS flags
- OSOCTET level
- OSOCTET state
- OSOCTET diagLevel
- OSOCTET lastChar
- struct OSRTSTREAM * pStream
- struct OSRTPrintStream * pPrintStrm

- OSRTDList elemNameStack
- OSRTDList regExpCache
- OSRTStack containerEndIndexStack
- const OSOCTET * key
- OSSIZE keylen
- OSVoidPtr pXMLInfo
- OSVoidPtr pEXIInfo
- OSVoidPtr pASN1Info
- OSVoidPtr pLicInfo
- OSVoidPtr pUserData
- OSVoidPtr pGlobalData
- struct OS3GPPSecParams * p3gppSec
- OSFreeCtxtGlobalPtr gblFreeFunc
- OSVoidPtr ssl
- struct OSRTDiagBitFieldList * pBitFldList
- OSUINT16 indent
- OSUINT16 version

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

The Documentation for this struct was generated from the following file:

- rtxContext.h

Member Data Documentation

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

OSRTBuffer struct Reference

```
#include <rtxContext.h>
```

Public Attributes

- OSOCTET * data
- OSSIZE byteIndex
- OSSIZE size
- OSINT16 bitOffset
- OSBOOL dynamic
- OSBOOL aligned

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

OSRTBufSave struct Reference

```
#include <rtxContext.h>
```

Public Attributes

- OSSIZE byteIndex
- OSINT16 bitOffset
- OSRTFLAGS flags

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

OSRTDList struct Reference

```
#include <rtxDList.h>
```

Public Attributes

- OSSIZE count
- OSRTDListNode * head
- OSRTDListNode * tail

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

The Documentation for this struct was generated from the following file:

- rtxDList.h

Member Data Documentation

OSSIZE OSRTDList::count

Count of items in the list.

Definition at line 65 of file rtxDList.h

The Documentation for this struct was generated from the following file:

- rtxDList.h

OSRTDListNode* OSRTDList::head

Pointer to first entry in list.

Definition at line 66 of file rtxDList.h

The Documentation for this struct was generated from the following file:

- rtxDList.h

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

OSRTDListBuf struct Reference

Public Attributes

- OSSIZE n
- OSSIZE nMax
- OSSIZE nAll
- OSSIZE firstSegSz
- OSSIZE elemSize
- OSRTDList tmplist
- void ** dataArray

OSRTDListNode struct Reference

```
#include <rtxDList.h>
```

Public Attributes

- void * data
- struct OSRTDListNode * next
- struct OSRTDListNode * prev

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

The Documentation for this struct was generated from the following file:

- rtxDList.h

Member Data Documentation

void* OSRTDListNode::data

Pointer to list data item.

Definition at line 53 of file rtxDList.h

The Documentation for this struct was generated from the following file:

- rtxDList.h

struct OSRTDListNode* OSRTDListNode::next

Pointer to next node in list.

Definition at line 54 of file rtxDList.h

The Documentation for this struct was generated from the following file:

- rtxDList.h

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

OSRTDListUTF8StrNode struct Reference

Public Attributes

- OSRTDListNode node
- OSUTF8CHAR utf8chars[1]

OSRTErrInfo struct Reference

```
#include <rtxContext.h>
```

Public Attributes

- OSRTErrLocn stack[OSRTERRSTKSIZ]
- OSINT16 status
- OSUINT8 stkx
- OSUINT8 parmcnt
- OSUTF8CHAR * parms[OSRTMAXERRPRM]
- OSUTF8CHAR * elemName
- OSBOOL fatal

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

OSRTErrInfoList struct Reference

Public Attributes

- OSRTDList list
- OSRTErrInfo reserved
- OSRTDListNode reservedNode

OSRTErrLocn struct Reference

```
#include <rtxContext.h>
```

Public Attributes

- const OSUTF8CHAR * module
- OSINT32 lineno

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

OSRTMEMBUF struct Reference

Public Attributes

- OSCTXT * pctxt
- OSSIZE segsize
- OSSIZE startidx

- OSSIZE usedcnt
- OSSIZE bufsize
- OSSIZE bitOffset
- OSUINT32 userState
- OSOCTET * buffer
- OSBOOL isDynamic
- OSBOOL isExpandable
- OSBOOL useSysMem

OSRTPrintStream struct Reference

```
#include <rtxPrintStream.h>
```

Public Attributes

- rtxPrintCallback pfPrintFunc
- void * pPrntStrmInfo

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

OSRTStrBuf struct Reference

Public Attributes

- char * strbuf
- OSSIZE bufsize
- OSSIZE endx

OSRTSTREAM struct Reference

```
#include <rtxStream.h>
```

Public Attributes

- OSRTStreamReadProc read

- OSRTStreamWriteProc write
- OSRTStreamFlushProc flush
- OSRTStreamCloseProc close
- OSRTStreamSkipProc skip
- OSRTStreamMarkProc mark
- OSRTStreamResetProc reset
- OSRTStreamGetPosProc getPos
- OSRTStreamSetPosProc setPos
- void * extra
- size_t bufsize
- size_t readAheadLimit
- size_t bytesProcessed
- size_t markedBytesProcessed
- size_t ioBytes
- size_t nextMarkOffset
- size_t segsize
- OSUINT32 id
- OSRTMEMBUF * pCaptureBuf
- OSUINT16 flags

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

The Documentation for this struct was generated from the following file:

- rtxStream.h

Member Data Documentation

OSRTStreamReadProc OSRTSTREAM::read

pointer to read function

Definition at line 185 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamWriteProc OSRTSTREAM::write

pointer to write function

Definition at line 186 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamFlushProc OSRTSTREAM::flush

pointer to flush function

Definition at line 187 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamCloseProc OSRTSTREAM::close

pointer to close function

Definition at line 188 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamSkipProc OSRTSTREAM::skip

pointer to skip function

Definition at line 189 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamMarkProc OSRTSTREAM::mark

pointer to mark function

Definition at line 190 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamResetProc OSRTSTREAM::reset

pointer to reset function

Definition at line 191 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamGetPosProc OSRTSTREAM::getPos

pointer to getPos function

Definition at line 192 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTStreamSetPosProc OSRTSTREAM::setPos

pointer to setPos function

Definition at line 193 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

void* OSRTSTREAM::extra

pointer to stream-specific data

Definition at line 195 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

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

size_t OSRTSTREAM::readAheadLimit

read ahead limit (used by rtxStreamMark/rtxStreamReset

Definition at line 200 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

size_t OSRTSTREAM::bytesProcessed

the number of bytes processed by the application program

Definition at line 202 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

size_t OSRTSTREAM::markedBytesProcessed

the marked number of bytes already processed

Definition at line 204 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

size_t OSRTSTREAM::ioBytes

the actual number of bytes read from or written to the stream

Definition at line 206 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

size_t OSRTSTREAM::nextMarkOffset

offset of next appropriate mark position

Definition at line 208 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

size_t OSRTSTREAM::segsize

size of decoded segment

Definition at line 210 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSUINT32 OSRTSTREAM::id

id of stream (see OSRTSTRMID_* macros)

Definition at line 211 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSRTMEMBUF* OSRTSTREAM::pCaptureBuf

Buffer into which data read from stream can be captured for debugging purposes.

Definition at line 217 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

OSUINT16 OSRTSTREAM::flags

flags (see OSRTSTRMF_* macros

Definition at line 219 of file rtxStream.h

The Documentation for this struct was generated from the following file:

- rtxStream.h

Chapter 4. File Documentation

rtxBase64.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- long rtxBase64EncodeData (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET ** ppDstData)
- long rtxBase64EncodeURLParam (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET ** ppDstData)
- long rtxBase64DecodeData (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET ** ppDstData)
- long rtxBase64DecodeDataToFSB (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET * buf, size_t bufsiz)
- long rtxBase64GetBinDataLen (const char * pSrcData, size_t srcDataSize)
- long rtxBase64UrlEncodeData (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET ** ppDstData)
- long rtxBase64UrlDecodeData (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET ** ppDstData)
- long rtxBase64UrlDecodeDataToFSB (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET * buf, size_t bufsiz)
- int rtxBase64CharToIdx (char c, OSBOOL url)
- char rtxBase64IdxToChar (int idx, OSBOOL url)

Detailed Description

base64 and base64url are defined in RFC 4648.

Definition in file rtxBase64.h

rtxBigInt.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Classes

- struct OSBigInt

Macros

- #define rtxBigIntSetBytes rtxBigIntSetBytesSigned

Typedefs

- typedef struct OSBigInt OSBigInt

Functions

- void rtxBigIntInit (OSBigInt * pInt)
- int rtxBigIntEnsureCapacity (OSCTXT * pctxt, OSBigInt * pInt, OSSIZE capacity)
- int rtxBigIntSetStr (OSCTXT * pctxt, OSBigInt * pInt, const char * value, int radix)
- int rtxBigIntSetStrn (OSCTXT * pctxt, OSBigInt * pInt, const char * value, OSSIZE len, int radix)
- int rtxBigIntSetInt64 (OSCTXT * pctxt, OSBigInt * pInt, OSINT64 value)
- int rtxBigIntSetUInt64 (OSCTXT * pctxt, OSBigInt * pInt, OSUINT64 value)
- int rtxBigIntSetBytesSigned (OSCTXT * pctxt, OSBigInt * pInt, OSOCTET * value, OSSIZE vallen)
- int rtxBigIntSetBytesUnsigned (OSCTXT * pctxt, OSBigInt * pInt, OSOCTET * value, OSSIZE vallen)
- OSSIZE rtxBigIntGetDataLen (const OSBigInt * pInt)
- int rtxBigIntGetData (OSCTXT * pctxt, const OSBigInt * pInt, OSOCTET * buffer, OSSIZE bufSize)
- OSSIZE rtxBigIntDigitsNum (const OSBigInt * pInt, int radix)
- int rtxBigIntCopy (OSCTXT * pctxt, const OSBigInt * pSrc, OSBigInt * pDst)
- int rtxBigIntFastCopy (OSCTXT * pctxt, const OSBigInt * pSrc, OSBigInt * pDst)
- OSBOOL rtxBigIntToReal (const OSBigInt * pSrc, OSREAL * pvalue)
- int rtxBigIntToString (OSCTXT * pctxt, const OSBigInt * pInt, int radix, char * str, OSSIZE strSize)
- char * rtxBigIntToDynStr (OSCTXT * pctxt, const OSBigInt * pInt, int radix)
- int rtxBigIntPrint (const OSUTF8CHAR * name, const OSBigInt * bigint, int radix)
- int rtxBigIntCompare (const OSBigInt * arg1, const OSBigInt * arg2)
- int rtxBigIntStrCompare (OSCTXT * pctxt, const char * arg1, const char * arg2)
- void rtxBigIntFree (OSCTXT * pctxt, OSBigInt * pInt)
- int rtxBigIntAdd (OSCTXT * pctxt, OSBigInt * result, const OSBigInt * arg1, const OSBigInt * arg2)
- int rtxBigIntSubtract (OSCTXT * pctxt, OSBigInt * result, const OSBigInt * arg1, const OSBigInt * arg2)
- int rtxBigIntMultiply (OSCTXT * pctxt, OSBigInt * result, const OSBigInt * arg1, const OSBigInt * arg2)
- EXTERN OSSIZE rtxBase2DigitsToRadix (OSUINT8 radix, OSSIZE n)
- EXTERN OSSIZE rtxRadixDigitsToBase2 (OSUINT8 radix, OSSIZE n)
- unsigned short rtxBigIntBitsPerDigit (int halfRadix)
- short rtxBigIntByteRadix (int halfRadix)

Detailed Description

Definition in file rtxBigInt.h

rtxBitString.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Macros

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

Functions

- OSUINT32 rtxGetBitCount (OSUINT32 value)
- int rtxSetBit (OSOCTET * pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)
- int rtxClearBit (OSOCTET * pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSBOOL rtxTestBit (const OSOCTET * pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSSIZE rtxLastBitSet (const OSOCTET * pBits, OSSIZE numbits)
- int rtxCheckBitBounds (OSCTXT * pctxt, OSOCTET ** ppBits, OSSIZE * pNumocts, OSSIZE minRequiredBits, OSSIZE preferredLimitBits)
- int rtxZeroUnusedBits (OSOCTET * pBits, OSSIZE numbits)
- int rtxCheckUnusedBitsZero (const OSOCTET * pBits, OSSIZE numbits)

Detailed Description

Contains utility functions for setting, clearing, and testing bits at any position in an arbitrarily sized array of bytes.

Definition in file rtxBitString.h

rtxBuffer.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

```
#include "rtxsrc/rtxSList.h"
```

Classes

- struct _OSRTBufLocDescr

Macros

- #define MIN_STREAM_BACKOFF 0

- #define OSRTPUTCHAR rtxWriteBytes (pctxt, (OSOCTET*)&ch, 1)
- #define OSRTPUTCHARREV (pctxt)->buffer.data[--(pctxt)->buffer.byteIndex]=(OSOCTET)ch;
- #define OSRTZTERM (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]=(OSOCTET)0;
- #define OSRTSAFEZTERM do { \ if (rtxCheckOutputBuffer (pctxt, 1) == 0) \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]=(OSOCTET)0; \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEPUTCHAR do { \ if (rtxCheckOutputBuffer (pctxt, 1) == 0) \ (pctxt)->lastChar= \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex++]=(OSOCTET)ch; \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEPUTCHAR_ZTERM do { \ if (rtxCheckOutputBuffer (pctxt, 2) == 0) { \ (pctxt)->lastChar= \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex++]=(OSOCTET)ch; \ (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]=(OSOCTET)0; } \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEPUTCHAR1 do { \ OSOCTET b = (OSOCTET)ch; \ rtxWriteBytes (pctxt, &b, 1); \ } while (0)
- #define OSRTMEMCPY do { \ OSCRTLSAFEMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], \ (pctxt)->buffer.size-(pctxt)->buffer.byteIndex, bdata, len); \ (pctxt)->buffer.byteIndex += len; \ (pctxt)->lastChar = (pctxt)->buffer.data[(pctxt)->buffer.byteIndex-1]; \ } while (0)
- #define OSRTMEMCPYREV do { \ (pctxt)->buffer.byteIndex -= len; \ OSCRTLSAFEMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], \ (pctxt)->buffer.size-(pctxt)->buffer.byteIndex, bdata, len); \ } while (0)
- #define OSRTSAFEMEMCPY do { \ if (rtxCheckOutputBuffer (pctxt, len) == 0) { \ OSCRTLMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], bdata, len); \ (pctxt)->buffer.byteIndex += len; \ (pctxt)->lastChar = (pctxt)->buffer.data[(pctxt)->buffer.byteIndex-1]; } \ else return LOG_RTERRNEW (pctxt, RTERR_BUFOVFLW); \ } while (0)
- #define OSRTSAFEMEMCPY1 do { \ if (rtxCheckOutputBuffer (pctxt, len) == 0) { \ OSCRTLMEMCPY (&(pctxt)->buffer.data[(pctxt)->buffer.byteIndex], bdata, len); \ (pctxt)->buffer.byteIndex += len; \ (pctxt)->lastChar = (pctxt)->buffer.data[(pctxt)->buffer.byteIndex-1]; \ stat = 0; } \ else stat = RTERR_BUFOVFLW; \ } while (0)
- #define OSRTGETBUFUTF8LEN rtxCalcUTF8Len (OSRTBUFPTR (pctxt), OSRTBUFSIZE (pctxt))
- #define OSRTCHKBUFUTF8LEN do { size_t nchars = OSRTGETBUFUTF8LEN (pctxt); \ stat = (nchars >= lower && nchars <= upper) ? 0 : RTERR_CONSVIO; } while(0)
- #define OSRTENDOFBUF ((pctxt)->buffer.byteIndex >= (pctxt)->buffer.size)
- #define OSRTByteAlign if ((pctxt)->buffer.bitOffset != 8) { \ (pctxt)->buffer.byteIndex++; \ (pctxt)->buffer.bitOffset = 8; } \

Typedefs

- typedef struct _OSRTBufLocDescr OSRTBufLocDescr

Functions

- int rtxCheckBuffer (OSCTXT * pctxt, size_t nbytes)
- int rtxCheckOutputBuffer (OSCTXT * pctxt, OSSIZE nbytes)

- int rtxDecrBufferByteIndex (OSCTXT * pctxt, OSSIZE nbytes)
- OSBOOL rtxIsOutputBufferFlushable (OSCTXT * pctxt)
- int rtxFlushOutputBuffer (OSCTXT * pctxt)
- int rtxExpandOutputBuffer (OSCTXT * pctxt, size_t nbytes)
- int rtxCheckInputBuffer (OSCTXT * pctxt, size_t nbytes)
- int rtxCopyAsciiText (OSCTXT * pctxt, const char * text)
- int rtxCopyUTF8Text (OSCTXT * pctxt, const OSUTF8CHAR * text)
- int rtxCopyUnicodeText (OSCTXT * pctxt, const OSUNICHAR * text)
- int rtxLoadInputBuffer (OSCTXT * pctxt, OSSIZE nbytes)
- int rtxPeekByte (OSCTXT * pctxt, OSOCTET * pbyte)
- int rtxPeekBytes (OSCTXT * pctxt, OSOCTET * pdata, OSSIZE bufsize, OSSIZE nocts, OSSIZE * pactual)
- int rtxReadBytesSafe (OSCTXT * pctxt, OSOCTET * buffer, size_t bufsize, size_t nocts)
- int rtxReadBytes (OSCTXT * pctxt, OSOCTET * pdata, size_t nocts)
- int rtxReadBytesDynamic (OSCTXT * pctxt, OSOCTET ** ppdata, size_t nocts, OSBOOL * pMemAlloc)
- int rtxWriteBytes (OSCTXT * pctxt, const OSOCTET * pdata, size_t nocts)
- int rtxWriteIndent (OSCTXT * pctxt)
- void rtxIndentDecr (OSCTXT * pctxt)
- void rtxIndentIncr (OSCTXT * pctxt)
- void rtxIndentReset (OSCTXT * pctxt)
- size_t rtxGetIndentLevels (OSCTXT * pctxt)
- OSBOOL rtxCanonicalSort (OSOCTET * refPoint, OSRTSList * pList, OSBOOL normal)
- int rtxEncCanonicalSort (OSCTXT * pctxt, OSCTXT * pMemCtxt, OSRTSList * pList)
- void rtxGetBufLocDescr (OSCTXT * pctxt, OSRTBufLocDescr * pDescr)
- void rtxAddBufLocDescr (OSCTXT * pctxt, OSRTSList * pElemList, OSRTBufLocDescr * pDescr)

Detailed Description

Common runtime functions for reading from or writing to the message buffer defined within the context structure.

Definition in file rtxBuffer.h

rtxCharStr.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- int rtxStricmp (const char * str1, const char * str2)
- int rtxStrnicmp (const char * str1, const char * str2, size_t count)
- char * rtxStrcat (char * dest, size_t bufsiz, const char * src)
- char * rtxStrncat (char * dest, size_t bufsiz, const char * src, size_t nchars)
- char * rtxStrcpy (char * dest, size_t bufsiz, const char * src)
- char * rtxStrncpy (char * dest, size_t bufsiz, const char * src, size_t nchars)
- char * rtxStrdup (OSCTXT * pctxt, const char * src)
- char * rtxStrndup (OSCTXT * pctxt, const char * src, OSSIZE nchars)
- const char * rtxStrJoin (char * dest, size_t bufsiz, const char * str1, const char * str2, const char * str3, const char * str4, const char * str5)
- char * rtxStrDynJoin (OSCTXT * pctxt, const char * str1, const char * str2, const char * str3, const char * str4, const char * str5)
- char * rtxStrTrimEnd (char * s)
- int rtxValidateConstrainedStr (OSCTXT * pctxt, const char * pvalue, const char * pCharSet)
- int rtxIntToCharStr (OSINT32 value, char * dest, size_t bufsiz, char padchar)
- int rtxUIntToCharStr (OSUINT32 value, char * dest, size_t bufsiz, char padchar)
- int rtxInt64ToCharStr (OSINT64 value, char * dest, size_t bufsiz, char padchar)
- int rtxUInt64ToCharStr (OSUINT64 value, char * dest, size_t bufsiz, char padchar)
- int rtxSizeToCharStr (size_t value, char * dest, size_t bufsiz, char padchar)
- int rtxHexCharsToBinCount (const char * hexstr, size_t nchars)
- int rtxHexCharsToBin (const char * hexstr, size_t nchars, OSOCTET * binbuf, size_t bufsize)
- int rtxCharStrToInt (const char * cstr, OSINT32 * pvalue)
- int rtxCharStrnToInt (const char * cstr, OSSIZE ndigits, OSINT32 * pvalue)
- int rtxCharStrToInt8 (const char * cstr, OSINT8 * pvalue)
- int rtxCharStrToInt16 (const char * cstr, OSINT16 * pvalue)
- int rtxCharStrToInt64 (const char * cstr, OSINT64 * pvalue)
- int rtxCharStrToUInt (const char * cstr, OSUINT32 * pvalue)
- int rtxCharStrToUInt8 (const char * cstr, OSUINT8 * pvalue)
- int rtxCharStrToUInt16 (const char * cstr, OSUINT16 * pvalue)

- int rtxCharStrToUInt64 (const char * cstr, OSUINT64 * pvalue)

Detailed Description

Definition in file rtxCharStr.h

rtxCommon.h File Reference

```
#include "rtxsrc/osSysTypes.h"

#include "rtxsrc/osMacros.h"

#include "rtxsrc/rtxExternDefs.h"

#include "rtxsrc/rtxBigInt.h"

#include "rtxsrc/rtxBitString.h"

#include "rtxsrc/rtxBuffer.h"

#include "rtxsrc/rtxCharStr.h"

#include "rtxsrc/rtxCommonDefs.h"

#include "rtxsrc/rtxDateTime.h"

#include "rtxsrc/rtxDiag.h"

#include "rtxsrc/rtxEnum.h"

#include "rtxsrc/rtxError.h"

#include "rtxsrc/rtxFile.h"

#include "rtxsrc/rtxMemory.h"

#include "rtxsrc/rtxPattern.h"

#include "rtxsrc/rtxReal.h"

#include "rtxsrc/rtxUTF8.h"

#include "rtxsrc/rtxUtil.h"
```

Detailed Description

Common runtime constants, data structure definitions, and run-time functions to support various data encoding standards.

Definition in file rtxCommon.h

rtxContext.h File Reference

```
#include "rtxsrc/rtxDList.h"
```

```
#include "rtxsrc/rtxStack.h"
```

Classes

- struct OSRTErrLocn
- struct OSRTErrInfo
- struct OSRTErrInfoList
- struct OSRTBuffer
- struct OSRTBufSave
- struct OSBufferIndex
- struct OSCTXT

Macros

- #define OSRTENCBUFSIZ 1024 /* dynamic encode buffer extent size */
- #define OSRTERRSTKSIZ 8 /* error stack size */
- #define OSRTMAXERRPRM 5 /* maximum error parameters */
- #define OSDIAG 0x80000000 /* diagnostic tracing enabled */
- #define OSTRACE 0x40000000 /* tracing enabled */
- #define OSDISSTRM 0x20000000 /* disable stream encode/decode */
- #define OSNOSTRMBACKOFF 0x08000000 /* stream mark/reset funcs is not used */
- #define OS3GMOBORIG 0x04000000 /* 3G mobile-originated (net to MS) */
- #define OSCONTCLOSED 0x02000000 /* 3G container closed. */
- #define OSRESERVED1 0x01000000 /* reserved */
- #define OSBUFSYSALLOC 0x00800000 /* ctxt buf allocated using sys alloc */
- #define OSLICCHECKIN 0x00400000 /* check in ifloat license on free */
- #define OSNOWHITESPACE 0x00400000 /* Turn off indentation whitesapce */
- #define OSCDECL
- #define pLicInfo pli709
- #define OSRT_GET_FIRST_ERROR_INFO (((pctxt)->errInfo.list.head == 0) ? (OSRTErrInfo*)0 : \ (OSRTErrInfo*)((pctxt)->errInfo.list.head->data))
- #define OSRT_GET_LAST_ERROR_INFO (((pctxt)->errInfo.list.tail == 0) ? (OSRTErrInfo*)0 : \ (OSRTErrInfo*)((pctxt)->errInfo.list.tail->data))
- #define OSRTISSTREAM ((pctxt)->pStream != 0 && !((pctxt)->flags & OSDISSTRM))

- #define OSRTISBUFSTREAM (OSRTISSTREAM(pctxt) && (0 != ((pctxt)->pStream->flags & OSRTSTRMF_BUFFERED)))
- #define OSRTBUFCUR (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]
- #define OSRTBUFPTR &(pctxt)->buffer.data[(pctxt)->buffer.byteIndex]
- #define OSRTBUFFER (pctxt)->buffer.data
- #define OSRTBUFSIZE (pctxt)->buffer.size
- #define OSRTBUFSAVE { \ (pctxt)->savedInfo.byteIndex = (pctxt)->buffer.byteIndex; \ (pctxt)->savedInfo.flags = (pctxt)->flags; }
- #define OSRTBUFSAVE2 { \ (pSavedBuf)->byteIndex = (pctxt)->buffer.byteIndex; \ (pSavedBuf)->bitOffset = (pctxt)->buffer.bitOffset; \ (pSavedBuf)->flags = (pctxt)->flags; }
- #define OSRTBUFRESTORE { \ (pctxt)->buffer.byteIndex = (pctxt)->savedInfo.byteIndex; \ (pctxt)->flags = (pctxt)->savedInfo.flags; }
- #define OSRTBUFRESTORE2 { \ (pctxt)->buffer.byteIndex = (pSavedBuf)->byteIndex; \ (pctxt)->buffer.bitOffset = (pSavedBuf)->bitOffset; \ (pctxt)->flags = (pSavedBuf)->flags; }
- #define OSRTBYTEALIGNED ((pctxt)->buffer.bitOffset == 8 || (pctxt)->buffer.bitOffset == 0)
- #define rtxCtxtGetMsgPtr (pctxt)->buffer.data
- #define rtxCtxtGetMsgLen (pctxt)->buffer.byteIndex
- #define rtxCtxtTestFlag (((pctxt)->flags & mask) != 0)
- #define rtxCtxtPeekElemName (((pctxt)->elemNameStack.count > 0) ? \ (const OSUTF8CHAR*)(pctxt)->elemNameStack.tail->data : (const OSUTF8CHAR*)0)
- #define rtxByteAlign if ((pctxt)->buffer.bitOffset != 8) { \ (pctxt)->buffer.byteIndex++; (pctxt)->buffer.bitOffset = 8; }
- #define rtxCtxtSetProtocolVersion (pctxt)->version = value
- #define rtxMarkBitPos (*(pbitoff) = (OSUINT8) (pctxt)->buffer.bitOffset, rtxMarkPos (pctxt, ppos))
- #define rtxResetToBitPos ((pctxt)->buffer.bitOffset = (OSUINT8) bitoff, rtxResetToPos (pctxt, pos))
- #define RTXCTXTPUSHARRAYELEMNAME rtxCtxtPushArrayElemName(pctxt,OSUTF8(name),idx)
- #define RTXCTXTPOPARRAYELEMNAME rtxCtxtPopArrayElemName(pctxt)
- #define RTXCTXTPUSHELEMNAME rtxCtxtPushElemName(pctxt,OSUTF8(name))
- #define RTXCTXTPOPELEMNAME rtxCtxtPopElemName(pctxt)
- #define RTXCTXTPUSHTYPENAME rtxCtxtPushTypeName(pctxt,OSUTF8(name))
- #define RTXCTXTPOPTYPENAME rtxCtxtPopTypeName(pctxt)

Typedefs

- typedef OSUINT32 OSRTFLAGS

- typedef int(* OSFreeCtxtAppInfoPtr
- typedef int(* OSResetCtxtAppInfoPtr
- typedef void(* OSFreeCtxtGlobalPtr
- typedef struct OSCTXT OSCTXT
- typedef void *OSCDECL * OSMallocFunc
- typedef void *(OSCDECL * OSReallocFunc

Functions

- typedef void (OSCDECL * OSFreeFunc)
- int rtxInitContext (OSCTXT * pctxt)
- int rtxInitContextExt (OSCTXT * pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- int rtxInitThreadContext (OSCTXT * pctxt, const OSCTXT * pSrcCtxt)
- int rtxInitContextUsingKey (OSCTXT * pctxt, const OSOCTET * key, OSSIZE keylen)
- int rtxInitContextBuffer (OSCTXT * pctxt, OSOCTET * bufaddr, OSSIZE bufsiz)
- int rtxCtxtSetBufPtr (OSCTXT * pctxt, OSOCTET * bufaddr, OSSIZE bufsiz)
- OSSIZE rtxCtxtGetBitOffset (OSCTXT * pctxt)
- int rtxCtxtSetBitOffset (OSCTXT * pctxt, OSSIZE offset)
- OSSIZE rtxCtxtGetIOByteCount (OSCTXT * pctxt)
- int rtxCheckContext (OSCTXT * pctxt)
- void rtxFreeContext (OSCTXT * pctxt)
- void rtxCopyContext (OSCTXT * pdest, OSCTXT * psrc)
- void rtxCtxtSetFlag (OSCTXT * pctxt, OSUINT32 mask)
- void rtxCtxtClearFlag (OSCTXT * pctxt, OSUINT32 mask)
- int rtxCtxtPushArrayElemName (OSCTXT * pctxt, const OSUTF8CHAR * elemName, OSSIZE idx)
- int rtxCtxtPushElemName (OSCTXT * pctxt, const OSUTF8CHAR * elemName)
- int rtxCtxtPushElemNameCopy (OSCTXT * pctxt, const OSUTF8CHAR * elemName)
- int rtxCtxtPushTypeName (OSCTXT * pctxt, const OSUTF8CHAR * typeName)
- OSBOOL rtxCtxtPopArrayElemName (OSCTXT * pctxt)
- const OSUTF8CHAR * rtxCtxtPopElemName (OSCTXT * pctxt)
- void rtxCtxtPopElemNameCopy (OSCTXT * pctxt)

- const OSUTF8CHAR * rtxCtxtPopTypeName (OSCTXT * pctxt)
- OSBOOL rtxCtxtContainerHasRemBits (OSCTXT * pctxt)
- OSBOOL rtxCtxtContainerEnd (OSCTXT * pctxt)
- OSSIZE rtxCtxtGetContainerRemBits (OSCTXT * pctxt)
- int rtxCtxtPushContainerBytes (OSCTXT * pctxt, OSSIZE bytes)
- int rtxCtxtPushContainerBits (OSCTXT * pctxt, OSSIZE bits)
- void rtxCtxtPopContainer (OSCTXT * pctxt)
- void rtxCtxtPopAllContainers (OSCTXT * pctxt)
- int rtxPreInitContext (OSCTXT * pctxt)
- void rtxCtxtSetMemHeap (OSCTXT * pctxt, OSCTXT * pSrcCtxt)
- void rtxMemHeapSetFlags (OSCTXT * pctxt, OSUINT32 flags)
- void rtxMemHeapClearFlags (OSCTXT * pctxt, OSUINT32 flags)
- int rtxCtxtMarkBitPos (OSCTXT * pctxt, OSSIZE * ppos)
- int rtxCtxtResetToBitPos (OSCTXT * pctxt, OSSIZE pos)
- int rtxMarkPos (OSCTXT * pctxt, OSSIZE * ppos)
- int rtxResetToPos (OSCTXT * pctxt, OSSIZE pos)
- const char * rtxCtxtGetExpDateStr (OSCTXT * pctxt, char * buf, OSSIZE bufsiz)
- void rtxLicenseClose (void)

Detailed Description

Common run-time context definitions.

Definition in file rtxContext.h

rtxCtype.h File Reference

Macros

- #define OS_ISASCII ((unsigned)(c) < 0x80)
- #define OS_ISUPPER (c >= 'A' && c <= 'Z')
- #define OS_ISLOWER (c >= 'a' && c <= 'z')
- #define OS_ISDIGIT (c >= '0' && c <= '9')
- #define OS_ISALPHA (OS_ISUPPER(c) || OS_ISLOWER(c))
- #define OS_ISSPACE ((c >= 0x09 && c <= 0x0d) || (c == ' '))

- #define OS_ISPUNCT (c >= 0 && c <= 0x20)
- #define OS_ISALNUM (OS_ISALPHA(c) || OS_ISDIGIT(c))
- #define OS_ISPRINT (c >= ' ' && c <= '~')
- #define OS_ISGRAPH (c >= '!' && c <= '~')
- #define OS_ISCNTRL ((c >= 0 && c <= 0x1F) || c == 0x7F)
- #define OS_ISXDIGIT (OS_ISDIGIT(c) || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f'))
- #define OS_ISBASE64 (OS_ISALNUM(c) || c == '+' || c == '/' || c == '=')
- #define OS_TOLOWER (OS_ISUPPER(c) ? (c) - 'A' + 'a' : (c))
- #define OS_TOUPPER (OS_ISLOWER(c) ? (c) - 'a' + 'A' : (c))

Detailed Description

Definition in file rtxCtype.h

rtxDateTime.h File Reference

```
#include <time.h>
```

```
#include "rtxsrc/rtxContext.h"
```

Functions

- int rtxDateToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxTimeToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxDateTimeToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGYearToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGYearMonthToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGMonthToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGMonthDayToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGDayToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
- int rtxGetCurrDateTime (OSNumDateTime * pvalue)
- int rtxGetCurrDateTimeString (char * buffer, OSSIZE bufsize, OSBOOL local)
- int rtxCmpDate (const OSNumDateTime * pvalue1, const OSNumDateTime * pvalue2)
- int rtxCmpDate2 (const OSNumDateTime * pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpTime (const OSNumDateTime * pvalue1, const OSNumDateTime * pvalue2)

- int rtxCmpTime2 (const OSNumDateTime * pvalue, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpDateTime (const OSNumDateTime * pvalue1, const OSNumDateTime * pvalue2)
- int rtxCmpDateTime2 (const OSNumDateTime * pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxParseDateString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseTimeString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseDateTimeString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGYearString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGYearMonthString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGMonthString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGMonthDayString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxParseGDayString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)
- int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR * buf, OSUINT32 bufsize)
- int rtxDurationToMSecs (OSUTF8CHAR * buf, OSUINT32 bufsize, OSINT32 * msecs)
- int rtxSetDateTime (OSNumDateTime * pvalue, struct tm * timeStruct)
- int rtxGetGMTime (struct tm * pvalue, time_t timeMs)
- int rtxGetLocalTime (struct tm * pvalue, time_t timeMs)
- int rtxSetLocalDateTime (OSNumDateTime * pvalue, time_t timeMs)
- int rtxSetUtcDateTime (OSNumDateTime * pvalue, time_t timeMs)
- int rtxGetDateTime (const OSNumDateTime * pvalue, time_t * timeMs)
- OSBOOL rtxDateIsValid (const OSNumDateTime * pvalue)
- OSBOOL rtxTimeIsValid (const OSNumDateTime * pvalue)
- OSBOOL rtxDateTimeIsValid (const OSNumDateTime * pvalue)
- int rtxAscTime (char * buffer, OSSIZE bufsize, struct tm * pvalue)

Detailed Description

Common runtime functions for converting to and from various standard date/time formats.

Definition in file rtxDateTime.h

rtxDecimal.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Functions

- const char * rtxNR3toDecimal (OSCTXT * pctxt, const char * object_p)

Detailed Description

Common runtime functions for working with xsd:decimal numbers.

Definition in file rtxDecimal.h

rtxDiag.h File Reference

```
#include <stdarg.h>
```

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define RTDIAG1
- #define RTDIAG2
- #define RTDIAG3
- #define RTDIAG4
- #define RTDIAG5
- #define RTDIAGU
- #define RTHEXDUMP
- #define RTDIAGCHARS
- #define RTDIAGSTRM2
- #define RTDIAGSTRM3
- #define RTDIAGSTRM4
- #define RTDIAGSTRM5
- #define RTHEXDUMPSTRM
- #define RTDIAGSCHARS

Enumerations

- enum OSRTDiagTraceLevel {
  OSRTDiagError,
  OSRTDiagWarning,
  OSRTDiagInfo,
  OSRTDiagDebug
  }

Functions

- OSBOOL rtxDiagEnabled (OSCTXT * pctxt)
- OSBOOL rtxSetDiag (OSCTXT * pctxt, OSBOOL value)
- OSBOOL rtxSetGlobalDiag (OSBOOL value)
- void rtxDiagPrint (OSCTXT * pctxt, const char * fmtspec, ...)
- void rtxDiagStream (OSCTXT * pctxt, const char * fmtspec, ...)
- void rtxDiagHexDump (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)
- void rtxDiagStreamHexDump (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)
- void rtxDiagPrintChars (OSCTXT * pctxt, const char * data, size_t nchars)
- void rtxDiagStreamPrintChars (OSCTXT * pctxt, const char * data, size_t nchars)
- void rtxDiagStreamPrintBits (OSCTXT * pctxt, const char * descr, const OSOCTET * data, size_t bitIndex, size_t nbits)
- void rtxDiagSetTraceLevel (OSCTXT * pctxt, OSRTDiagTraceLevel level)
- OSBOOL rtxDiagTraceLevelEnabled (OSCTXT * pctxt, OSRTDiagTraceLevel level)

Detailed Description

Common runtime functions for diagnostic tracing and debugging.

Definition in file rtxDiag.h

rtxDList.h File Reference

```
#include "rtxsrc/osSysTypes.h"

#include "rtxsrc/rtxExternDefs.h"

#include "rtxsrc/rtxCommonDefs.h"
```

Classes

- struct OSRTDListNode
- struct OSRTDList
- struct OSRTDListBuf
- struct OSRTDListUTF8StrNode

Macros

- #define DLISTBUF_SEG 16

- #define OSRTDLISTNODESIZE ((sizeof(OSRTDListNode)+7)&(~7))
- #define rtxDListAllocNodeAndData do { \ *ppnode = (OSRTDListNode*) \ rtxMemAlloc (pctxt, sizeof(type)+OSRTDLISTNODESIZE); \ if (0 != *ppnode) { \ (*ppnode)->data = (void*)((char*)(*ppnode)+OSRTDLISTNODESIZE); \ *ppdata = (type*)((*ppnode)->data); \ } else { *ppdata = 0; } \ } while (0)
- #define rtxDListAppendData do { \ rtxDListAppend(pctxt,pList,pData); \ } while(0);
- #define rtxDListFastInit do { \ if ((pList) != 0) { \ (pList)->head = (pList)->tail = (OSRTDListNode*) 0; \ (pList)->count = 0; } \ } while (0)
- #define rtxDListFreeTailNode rtxDListFreeNode(pctxt,pList,(pList)->tail)
- #define rtxDListFreeHeadNode rtxDListFreeNode(pctxt,pList,(pList)->head)

Typedefs

- typedef struct OSRTDListNode OSRTDListNode
- typedef struct OSRTDList OSRTDList
- typedef struct OSRTDListBuf OSRTDListBuf
- typedef struct OSRTDListUTF8StrNode OSRTDListUTF8StrNode
- typedef int(* PEqualsFunc

Functions

- void rtxDListInit (OSRTDList * pList)
- OSRTDListNode * rtxDListAppend (struct OSCTXT * pctxt, OSRTDList * pList, void * pData)
- OSRTDListNode * rtxDListAppendCharArray (struct OSCTXT * pctxt, OSRTDList * pList, size_t length, char * pData)
- OSRTDListNode * rtxDListAppendNode (OSRTDList * pList, OSRTDListNode * pListNode)
- OSRTDListNode * rtxDListInsert (struct OSCTXT * pctxt, OSRTDList * pList, OSSIZE idx, void * pData)
- OSRTDListNode * rtxDListInsertNode (OSRTDList * pList, OSSIZE idx, OSRTDListNode * pListNode)
- OSRTDListNode * rtxDListInsertBefore (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node, void * pData)
- OSRTDListNode * rtxDListInsertAfter (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node, void * pData)
- OSRTDListNode * rtxDListFindByIndex (const OSRTDList * pList, OSSIZE idx)
- OSRTDListNode * rtxDListFindByData (const OSRTDList * pList, void * data)
- OSRTDListNode * rtxDListFindFirstData (const OSRTDList * pList)
- int rtxDListFindIndexByData (const OSRTDList * pList, void * data)
- void rtxDListFreeNode (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node)

- void rtxDListRemove (OSRTDList * pList, OSRTDListNode * node)
- void rtxDListFreeNodes (struct OSCTXT * pctxt, OSRTDList * pList)
- void rtxDListFreeAll (struct OSCTXT * pctxt, OSRTDList * pList)
- int rtxDListToArray (struct OSCTXT * pctxt, OSRTDList * pList, void ** ppArray, OSSIZE * pElemCount, OSSIZE elemSize)
- int rtxDListToPointerArray (struct OSCTXT * pctxt, OSRTDList * pList, void *** ppArray, OSSIZE * pElemCount)
- int rtxDListAppendArray (struct OSCTXT * pctxt, OSRTDList * pList, void * pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListAppendArrayCopy (struct OSCTXT * pctxt, OSRTDList * pList, const void * pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListToUTF8Str (struct OSCTXT * pctxt, OSRTDList * pList, OSUTF8CHAR ** ppstr, char sep)
- OSRTDListNode * rtxDListInsertSorted (struct OSCTXT * pctxt, OSRTDList * pList, void * pData, PEqualsFunc equalsFunc, void * sortCtxt)
- OSRTDListNode * rtxDListInsertNodeSorted (OSRTDList * pList, OSRTDListNode * pListNode, PEqualsFunc equalsFunc, void * sortCtxt)
- void rtxDListBufInit (OSRTDListBuf * pBuf, OSSIZE segSz, void ** ppdata, OSSIZE elemSz)
- int rtxDListBufExpand (struct OSCTXT * pctxt, OSRTDListBuf * pBuf)
- int rtxDListBufToArray (struct OSCTXT * pctxt, OSRTDListBuf * pBuf)

Detailed Description

Doubly-Linked List Utility Functions.

Definition in file rtxDList.h

rtxErrCodes.h File Reference

Macros

- #define RT_OK 0
- #define RT_OK_FRAG 2
- #define RTERR_BUFOVFLW -1
- #define RTERR_ENDOFBUF -2
- #define RTERR_IDNOTFOU -3
- #define RTERR_INVENUM -4
- #define RTERR_SETDUPL -5

- #define RTERR_SETMISRQ -6
- #define RTERR_NOTINSET -7
- #define RTERR_SEQOVFLW -8
- #define RTERR_INVOPT -9
- #define RTERR_NOMEM -10
- #define RTERR_INVHEXS -11
- #define RTERR_INVREAL -12
- #define RTERR_STROVFLW -13
- #define RTERR_BADVALUE -14
- #define RTERR_TOODEEP -15
- #define RTERR_CONSVIO -16
- #define RTERR_ENDOFFILE -17
- #define RTERR_INVUTF8 -18
- #define RTERR_OUTOFBND -19
- #define RTERR_INVPARAM -20
- #define RTERR_INVFORMAT -21
- #define RTERR_NOTINIT -22
- #define RTERR_TOOBIG -23
- #define RTERR_INVCHAR -24
- #define RTERR_XMLSTATE -25
- #define RTERR_XMLPARSE -26
- #define RTERR_SEQORDER -27
- #define RTERR_FILNOTFOU -28
- #define RTERR_READERR -29
- #define RTERR_WRITEERR -30
- #define RTERR_INVBASE64 -31
- #define RTERR_INVSOCKET -32
- #define RTERR_INVATTR -33
- #define RTERR_REGEXP -34
- #define RTERR_PATMATCH -35

- #define RTERR_ATTRMISRQ -36
- #define RTERR_HOSTNOTFOU -37
- #define RTERR_HTTPERR -38
- #define RTERR_SOAPERR -39
- #define RTERR_EXPIRED -40
- #define RTERR_UNEXPELEM -41
- #define RTERR_INVOCCUR -42
- #define RTERR_INVMSGBUF -43
- #define RTERR_DECELEMFAIL -44
- #define RTERR_DECATTRFAIL -45
- #define RTERR_STRMINUSE -46
- #define RTERR_NULLPTR -47
- #define RTERR_FAILED -48
- #define RTERR_ATTRFIXEDVAL -49
- #define RTERR_MULTIPLE -50
- #define RTERR_NOTYPEINFO -51
- #define RTERR_ADDRINUSE -52
- #define RTERR_CONNRESET -53
- #define RTERR_UNREACHABLE -54
- #define RTERR_NOCONN -55
- #define RTERR_CONNREFUSED -56
- #define RTERR_INVSOCKOPT -57
- #define RTERR_SOAPFAULT -58
- #define RTERR_MARKNOTSUP -59
- #define RTERR_NOTSUPP -60 /* feature is not supported */
- #define RTERR_UNBAL -61
- #define RTERR_EXPNAME -62
- #define RTERR_UNICODE -63
- #define RTERR_INVBOOL -64
- #define RTERR_INVNULL -65

- #define RTERR_INVLEN -66
- #define RTERR_UNKNOWNIE -67
- #define RTERR_NOTALIGNED -68
- #define RTERR_EXTRDATA -69
- #define RTERR_INVMAC -70
- #define RTERR_NOSECPARAMS -71
- #define RTERR_COPYFAIL -72
- #define RTERR_PARSEFAIL -73
- #define RTERR_VALCMPERR -74
- #define RTERR_BUFCMPERR -75
- #define RTERR_INVBITS -76
- #define RTERR_RLM -77
- #define RTERR_NOCODEC -78
- #define RTERR_WOULDBLOCK -79
- #define RTERR_NODATA -80
- #define RTERR_MBEDTLS -81
- #define RTERR_INTOVFLOW -82
- #define RTERR_ABORT_REQUESTED -83

Detailed Description

List of numeric status codes that can be returned by common run-time functions and generated code.

Definition in file rtxErrCodes.h

rtxError.h File Reference

```
#include "rtxsrc/rtxContext.h"

#include "rtxsrc/rtxErrCodes.h"

#include <errno.h>
```

Macros

- #define ERRNO errno
- #define LOG_RTERR rtxErrSetData(pctxt,stat,__FILE__,__LINE__)
- #define LOG_RTERRNEW rtxErrSetNewData(pctxt,stat,__FILE__,__LINE__)

- #define RETURN_RTERR return LOG_RTERR(pctxt, stat)
- #define OSRTASSERT if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }
- #define OSRTCHECKPARAM if (condition) { /* do nothing */ }
- #define LOG_RTERR1 (a,LOG_RTERR (pctxt, stat),stat)
- #define LOG_RTERRNEW1 (a,LOG_RTERRNEW (pctxt, stat),stat)
- #define LOG_RTERR2 (a,b,LOG_RTERR (pctxt, stat),stat)
- #define LOG_RTERRNEW2 (a,b,LOG_RTERRNEW (pctxt, stat),stat)
- #define LOG_RTERR_AND_FREE_MEM rtxMemFreePtr ((ctxt_p),(mem_p)), LOG_RTERR(ctxt_p, stat)

Typedefs

- typedef int(* OSErrCbFunc

Functions

- OSBOOL rtxErrAddCtxtBufParm (OSCTXT * pctxt)
- OSBOOL rtxErrAddDoubleParm (OSCTXT * pctxt, double errParm)
- OSBOOL rtxErrAddErrorTableEntry (const char *const * ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)
- OSBOOL rtxErrAddElemNameParm (OSCTXT * pctxt)
- OSBOOL rtxErrAddIntParm (OSCTXT * pctxt, int errParm)
- OSBOOL rtxErrAddInt64Parm (OSCTXT * pctxt, OSINT64 errParm)
- OSBOOL rtxErrAddSizeParm (OSCTXT * pctxt, OSSIZE errParm)
- OSBOOL rtxErrAddStrParm (OSCTXT * pctxt, const char * pErrParm)
- OSBOOL rtxErrAddStrnParm (OSCTXT * pctxt, const char * pErrParm, OSSIZE nchars)
- OSBOOL rtxErrAddUIntParm (OSCTXT * pctxt, unsigned int errParm)
- OSBOOL rtxErrAddUInt64Parm (OSCTXT * pctxt, OSUINT64 errParm)
- void rtxErrAssertionFailed (const char * conditionText, int lineNo, const char * fileName)
- const char * rtxErrFmtMsg (OSRTErrInfo * pErrInfo, char * bufp, OSSIZE bufsiz)
- void rtxErrFreeParms (OSCTXT * pctxt)
- char * rtxErrGetText (OSCTXT * pctxt, char * pBuf, OSSIZE * pBufSize)
- char * rtxErrGetTextBuf (OSCTXT * pctxt, char * pbuf, OSSIZE bufsiz)
- char * rtxErrGetMsgText (OSCTXT * pctxt)
- char * rtxErrGetMsgTextBuf (OSCTXT * pctxt, char * pbuf, OSSIZE bufsiz)

- OSRTErrInfo * rtxErrNewNode (OSCTXT * pctxt)
- void rtxErrInit (OSVOIDARG)
- int rtxErrReset (OSCTXT * pctxt)
- int rtxErrResetErrInfo (OSCTXT * pctxt)
- void rtxErrLogUsingCB (OSCTXT * pctxt, OSErrCbFunc cb, void * cbArg_p)
- void rtxErrPrint (OSCTXT * pctxt)
- void rtxErrPrintElement (OSRTErrInfo * pErrInfo)
- int rtxErrSetData (OSCTXT * pctxt, int status, const char * module, int lineno)
- int rtxErrSetNewData (OSCTXT * pctxt, int status, const char * module, int lineno)
- void rtxErrSetNonFatal (OSCTXT * pctxt)
- int rtxErrCheckNonFatal (OSCTXT * pctxt)
- int rtxErrGetFirstError (const OSCTXT * pctxt)
- int rtxErrGetLastError (const OSCTXT * pctxt)
- OSSIZE rtxErrGetErrorCnt (const OSCTXT * pctxt)
- int rtxErrGetStatus (const OSCTXT * pctxt)
- int rtxErrResetLastErrors (OSCTXT * pctxt, OSSIZE errorsToReset)
- int rtxErrCopy (OSCTXT * pDestCtxt, const OSCTXT * pSrcCtxt)
- int rtxErrAppend (OSCTXT * pDestCtxt, const OSCTXT * pSrcCtxt)
- int rtxErrInvUIntOpt (OSCTXT * pctxt, OSUINT32 ident)

Detailed Description

Error handling function and macro definitions.

Definition in file rtxError.h

rtxMemBuf.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Classes

- struct OSRTMEMBUF

Macros

- #define OSMBDFLTSEGSIZE 1024

- #define OSMEMBUFPTR ((pmb)->buffer + (pmb)->startidx)
- #define OSMEMBUFENDPTR ((pmb)->buffer + (pmb)->startidx + (pmb)->usedcnt)
- #define OSMEMBUFUSEDSIZE ((OSSIZE)(pmb)->usedcnt)
- #define OSMBAPPENDSTR if (0 != str) \ rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
- #define OSMBAPPENDSTRL rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
- #define OSMBAPPENDUTF8 if (0 != str) \ rtxMemBufAppend(pmb,(OSOCTET*)str,rtxUTF8LenBytes(str))
- #define OSMBAPPENDSTRZ rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str)+1)

Typedefs

- typedef struct OSRTMEMBUF OSRTMEMBUF

Functions

- int rtxMemBufAppend (OSRTMEMBUF * pMemBuf, const OSOCTET * pdata, OSSIZE nbytes)
- int rtxMemBufCut (OSRTMEMBUF * pMemBuf, OSSIZE fromOffset, OSSIZE nbytes)
- void rtxMemBufFree (OSRTMEMBUF * pMemBuf)
- OSOCTET * rtxMemBufGetData (const OSRTMEMBUF * pMemBuf, int * length)
- OSOCTET * rtxMemBufGetDataExt (const OSRTMEMBUF * pMemBuf, OSSIZE * length)
- OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF * pMemBuf)
- void rtxMemBufInit (OSCTXT * pCtxt, OSRTMEMBUF * pMemBuf, OSSIZE segsize)
- void rtxMemBufInitBuffer (OSCTXT * pCtxt, OSRTMEMBUF * pMemBuf, OSOCTET * buf, OSSIZE bufsize, OSSIZE segsize)
- int rtxMemBufPreAllocate (OSRTMEMBUF * pMemBuf, OSSIZE nbytes)
- void rtxMemBufReset (OSRTMEMBUF * pMemBuf)
- int rtxMemBufSet (OSRTMEMBUF * pMemBuf, OSOCTET value, OSSIZE nbytes)
- OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF * pMemBuf, OSBOOL isExpandable)
- OSBOOL rtxMemBufSetUseSysMem (OSRTMEMBUF * pMemBuf, OSBOOL value)
- OSSIZE rtxMemBufTrimW (OSRTMEMBUF * pMemBuf)

Detailed Description

Definition in file rtxMemBuf.h

rtxMemory.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define RT_MH_DONTKEEPFREE 0x1
- #define RT_MH_VALIDATEPTR 0x2
- #define RT_MH_CHECKHEAP 0x4
- #define RT_MH_TRACE 0x8
- #define RT_MH_DIAG 0x10
- #define RT_MH_DIAG_DEBUG 0x20
- #define RT_MH_ZEROONFREE 0x40
- #define RT_MH_ZEROARRAY 0x80
- #define RT_MH_SYSALLOC 0x100
- #define RT_MH_TRACE_FREELIST 0x200 /* enable printing of the free element list at certain junctures in the memory management code. Requires building with _MEMTRACE */
- #define OSRTMH_PROPID_DEFBLKSIZE 1
- #define OSRTMH_PROPID_SETFLAGS 2
- #define OSRTMH_PROPID_CLEARFLAGS 3
- #define OSRTMH_PROPID_KEEPFREEUNITS 4
- #define OSRTMH_PROPID_USER 10
- #define OSRTXM_K_MEMBLKSIZ (4*1024)
- #define OSRTALLOCTYPE (type*) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTALLOCTYPEZ (type*) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTREALLOCARRAY 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)
- #define OSCRTMALLOC0 malloc(nbytes)
- #define OSCRTFREE0 free(ptr)
- #define OSCRTMALLOC rtxMemAlloc
- #define OSCRTFREE rtxMemFreePtr
- #define OSCDECL
- #define rtxMemAlloc rtxMemHeapAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAlloc rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemAllocZ rtxMemHeapAllocZ(&(pctxt)->pMemHeap,nbytes)

- #define rtxMemSysAllocZ rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemRealloc rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void*)mem_p, nbytes)
- #define rtxMemSysRealloc rtxMemHeapSysRealloc(&(pctxt)->pMemHeap,(void*)mem_p,nbytes)
- #define rtxMemFreePtr rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemSysFreePtr rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemAllocType (ctype*)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocType (ctype*)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemAllocTypeZ (ctype*)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocTypeZ (ctype*)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemFreeType rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemSysFreeType rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemAllocArray (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), 0)
- #define rtxMemSysAllocArray (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_SYSALLOC)
- #define rtxMemAllocArrayZ (type*)rtxMemAllocArray2 (pctxt, n, sizeof(type), RT_MH_ZEROARRAY)
- #define rtxMemFreeArray rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemSysFreeArray rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemReallocArray (type*)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void*)mem_p, sizeof(type)*n)
- #define rtxMemNewAutoPtr rtxMemHeapAlloc(&(pctxt)->pMemHeap, nbytes)
- #define rtxMemAutoPtrRef rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemAutoPtrUnref rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemAutoPtrGetRefCount rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemCheckPtr rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemCheck rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)
- #define rtxMemPrint rtxMemHeapPrint(&(pctxt)->pMemHeap)
- #define rtxMemPrintWithFree rtxMemHeapPrintWithFree(&(pctxt)->pMemHeap)
- #define rtxMemSetProperty rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, pProp)

Functions

- void rtxMemHeapAddRef (void ** ppvMemHeap)
- void * rtxMemHeapAlloc (void ** ppvMemHeap, OSSIZE nbytes)
- void * rtxMemHeapAllocZ (void ** ppvMemHeap, OSSIZE nbytes)

- void * rtxMemHeapSysAlloc (void ** ppvMemHeap, OSSIZE nbytes)
- void * rtxMemHeapSysAllocZ (void ** ppvMemHeap, OSSIZE nbytes)
- int rtxMemHeapCheckPtr (void ** ppvMemHeap, const void * mem_p)
- void rtxMemHeapFreeAll (void ** ppvMemHeap)
- void rtxMemHeapFreePtr (void ** ppvMemHeap, void * mem_p)
- void rtxMemHeapSysFreePtr (void ** ppvMemHeap, void * mem_p)
- void * rtxMemHeapReallocStatic (void ** ppvMemHeap, void * mem_p, OSSIZE oldsize, OSSIZE newsize)
- void * rtxMemHeapRealloc (void ** ppvMemHeap, void * mem_p, OSSIZE nbytes_)
- void * rtxMemHeapSysRealloc (void ** ppvMemHeap, void * mem_p, OSSIZE nbytes_)
- void rtxMemHeapRelease (void ** ppvMemHeap)
- void rtxMemHeapReset (void ** ppvMemHeap)
- void rtxMemHeapSetProperty (void ** ppvMemHeap, OSUINT32 propId, void * pProp)
- void * rtxMemNewArray (OSSIZE nbytes)
- void * rtxMemNewArrayZ (OSSIZE nbytes)
- void rtxMemDeleteArray (void * mem_p)
- void * rtxMemHeapAutoPtrRef (void ** ppvMemHeap, void * ptr)
- int rtxMemHeapAutoPtrUnref (void ** ppvMemHeap, void * ptr)
- int rtxMemHeapAutoPtrGetRefCount (void ** ppvMemHeap, void * mem_p)
- void rtxMemHeapInvalidPtrHook (void ** ppvMemHeap, const void * mem_p)
- void rtxMemHeapCheck (void ** ppvMemHeap, const char * file, int line)
- void rtxMemHeapPrint (void ** ppvMemHeap)
- void rtxMemHeapPrintWithFree (void ** ppvMemHeap)
- int rtxMemHeapCreate (void ** ppvMemHeap)
- int rtxMemHeapCreateExt (void ** ppvMemHeap, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- int rtxMemStaticHeapCreate (void ** ppvMemHeap, void * pmem, OSSIZE memsize)
- int rtxMemHeapConvertStatic (void ** ppvMemHeap, void * pmem, OSSIZE memsize)
- void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- void rtxMemFreeOpenSeqExt (OSCTXT * pctxt, struct OSRTDList * pElemList)
- OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT * pctxt)

- void rtxMemSetDefBlkSize (OSUINT32 blkSize)
- OSUINT32 rtxMemGetDefBlkSize (OSVOIDARG)
- OSBOOL rtxMemHeapIsEmpty (OSCTXT * pctxt)
- OSBOOL rtxMemIsZero (const void * pmem, OSSIZE memsiz)
- void rtxMemFree (OSCTXT * pctxt)
- void rtxMemReset (OSCTXT * pctxt)
- void * rtxMemAllocArray2 (OSCTXT * pctxt, OSSIZE numElements, OSSIZE typeSize, OSUINT32 flags)

Detailed Description

Memory management function and macro definitions.

Definition in file rtxMemory.h

rtxPattern.h File Reference

```
#include "rtxsrc/osSysTypes.h"
```

```
#include "rtxsrc/rtxExternDefs.h"
```

```
#include "rtxsrc/rtxContext.h"
```

Functions

- OSBOOL rtxMatchPattern (OSCTXT * pctxt, const OSUTF8CHAR * text, const OSUTF8CHAR * pattern)
- OSBOOL rtxMatchPattern2 (OSCTXT * pctxt, const OSUTF8CHAR * pattern)
- void rtxFreeRegexpCache (OSCTXT * pctxt)

Detailed Description

Pattern matching functions.

Definition in file rtxPattern.h

rtxPrint.h File Reference

```
#include <stdio.h>
```

```
#include "rtxsrc/rtxContext.h"
```

```
#include "rtxsrc/rtxHexDump.h"
```

Macros

- #define OSRTINDENTSPACES 3 /* number of spaces for indent */

Functions

- void rtxPrintBoolean (const char * name, OSBOOL value)
- void rtxPrintDate (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintTime (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintDateTime (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGYear (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGYearMonth (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGMonth (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGMonthDay (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintGDay (const char * name, const OSNumDateTime * pvalue)
- void rtxPrintInteger (const char * name, OSINT32 value)
- void rtxPrintInt64 (const char * name, OSINT64 value)
- void rtxPrintIpv4Addr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintIpv6Addr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintTBCDStr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintText (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintUnsigned (const char * name, OSUINT32 value)
- void rtxPrintUInt64 (const char * name, OSUINT64 value)
- void rtxPrintHexStr (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintHexStrPlain (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintHexStrNoAscii (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintHexBinary (const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintCharStr (const char * name, const char * cstring)
- void rtxPrintUTF8CharStr (const char * name, const OSUTF8CHAR * cstring)
- void rtxPrintUnicodeCharStr (const char * name, const OSUNICHAR * str, int nchars)
- void rtxPrintUnicodeCharStr64 (const char * name, const OSUNICHAR * str, OSSIZE nchars)
- void rtxPrintReal (const char * name, OSREAL value)
- void rtxPrintNull (const char * name)
- void rtxPrintNVP (const char * name, const OSUTF8NVP * value)

- void rtxPrintArrayNVP (const char * name, OSSIZE subscript, const OSUTF8NVP * value)
- int rtxPrintFile (const char * filename)
- void rtxPrintIndent (OSVOIDARG)
- void rtxPrintIncrIndent (OSVOIDARG)
- void rtxPrintDecrIndent (OSVOIDARG)
- void rtxPrintCloseBrace (OSVOIDARG)
- void rtxPrintOpenBrace (const char *)
- const char * rtxGetArrayElemName (char * buffer, OSSIZE bufsize, const char * name, OSSIZE subscript)

Detailed Description

Definition in file rtxPrint.h

rtxPrintStream.h File Reference

```
#include <stdarg.h>
```

```
#include "rtxsrc/rtxContext.h"
```

Classes

- struct OSRTPrintStream
- struct OSRTStrBuf

Typedefs

- typedef void(* rtxPrintCallback
- typedef struct OSRTPrintStream OSRTPrintStream

Variables

- OSRTPrintStream g_PrintStream

Functions

- int rtxSetPrintStream (OSCTXT * pctxt, rtxPrintCallback myCallback, void * pStrmInfo)
- int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void * pStrmInfo)
- int rtxPrintToStream (OSCTXT * pctxt, const char * fmtspec, ...)
- int rtxDiagToStream (OSCTXT * pctxt, const char * fmtspec, va_list arglist)
- int rtxPrintStreamRelease (OSCTXT * pctxt)

- void rtxPrintStreamToStdoutCB (void * pPrntStrmInfo, const char * fmtspec, va_list arglist)
- void rtxPrintStreamToFileCB (void * pPrntStrmInfo, const char * fmtspec, va_list arglist)
- void rtxPrintStreamToStringCB (void * pPrntStrmInfo, const char * fmtspec, va_list arglist)

Detailed Description

Functions that allow printing diagnostic message to a stream using a callback function.

Definition in file rtxPrintStream.h

rtxPrintToStream.h File Reference

```
#include <stdio.h>
```

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define OSRTINDENTSPACES 3 /* number of spaces for indent */

Functions

- void rtxPrintToStreamBoolean (OSCTXT * pctxt, const char * name, OSBOOL value)
- void rtxPrintToStreamDate (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamTime (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamDateTime (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGYear (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGYearMonth (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGMonth (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGMonthDay (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamGDay (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)
- void rtxPrintToStreamInteger (OSCTXT * pctxt, const char * name, OSINT32 value)
- void rtxPrintToStreamInt64 (OSCTXT * pctxt, const char * name, OSINT64 value)
- void rtxPrintToStreamUnsigned (OSCTXT * pctxt, const char * name, OSUINT32 value)
- void rtxPrintToStreamUInt64 (OSCTXT * pctxt, const char * name, OSUINT64 value)
- void rtxPrintToStreamHexStr (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamHexStrPlain (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)

- void rtxPrintToStreamHexStrNoAscii (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamHexBinary (OSCTXT * pctxt, const char * name, OSSIZE numocts, const OSOCTET * data)
- void rtxPrintToStreamCharStr (OSCTXT * pctxt, const char * name, const char * cstring)
- void rtxPrintToStreamUTF8CharStr (OSCTXT * pctxt, const char * name, const OSUTF8CHAR * cstring)
- void rtxPrintToStreamUnicodeCharStr (OSCTXT * pctxt, const char * name, const OSUNICHAR * str, int nchars)
- void rtxPrintToStreamReal (OSCTXT * pctxt, const char * name, OSREAL value)
- void rtxPrintToStreamNull (OSCTXT * pctxt, const char * name)
- void rtxPrintToStreamNVP (OSCTXT * pctxt, const char * name, const OSUTF8NVP * value)
- int rtxPrintToStreamFile (OSCTXT * pctxt, const char * filename)
- void rtxPrintToStreamIndent (OSCTXT * pctxt)
- void rtxPrintToStreamResetIndent (OSCTXT * pctxt)
- void rtxPrintToStreamIncrIndent (OSCTXT * pctxt)
- void rtxPrintToStreamDecrIndent (OSCTXT * pctxt)
- void rtxPrintToStreamCloseBrace (OSCTXT * pctxt)
- void rtxPrintToStreamOpenBrace (OSCTXT * pctxt, const char *)
- void rtxHexDumpToStream (OSCTXT * pctxt, const OSOCTET * data, OSSIZE numocts)
- void rtxHexDumpToStreamEx (OSCTXT * pctxt, const OSOCTET * data, OSSIZE numocts, OSSIZE bytesPerUnit)
- void rtxHexDumpToStreamExNoAscii (OSCTXT * pctxt, const OSOCTET * data, OSSIZE numocts, OSSIZE bytesPerUnit)

Detailed Description

Definition in file rtxPrintToStream.h

rtxReal.h File Reference

```
#include "rtxsrc/osSysTypes.h"
```

```
#include "rtxsrc/rtxExternDefs.h"
```

Functions

- OSREAL rtxGetMinusInfinity (OSVOIDARG)
- OSREAL rtxGetMinusZero (OSVOIDARG)
- OSREAL rtxGetNaN (OSVOIDARG)

- OSREAL rtxGetPlusInfinity (OSVOIDARG)
- OSBOOL rtxIsMinusInfinity (OSREAL value)
- OSBOOL rtxIsMinusZero (OSREAL value)
- OSBOOL rtxIsNaN (OSREAL value)
- OSBOOL rtxIsPlusInfinity (OSREAL value)
- OSBOOL rtxIsApproximate (OSREAL a, OSREAL b, OSREAL delta)
- OSBOOL rtxIsApproximateAbs (OSREAL a, OSREAL b, OSREAL delta)

Detailed Description

Common runtime functions for working with floating-point numbers.

Definition in file rtxReal.h

rtxSList.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Classes

- struct _OSRTSListNode
- struct _OSRTSList

Macros

- #define OSALLOCELEMSNODE (type*) (((char*)rtxMemAllocZ (pctxt, sizeof(type) + \ sizeof(OSRTSListNode))) + sizeof(OSRTSListNode))

Typedefs

- typedef struct _OSRTSListNode OSRTSListNode
- typedef struct _OSRTSList OSRTSList

Functions

- void rtxSListInit (OSRTSList * pList)
- void rtxSListInitEx (OSCTXT * pctxt, OSRTSList * pList)
- void rtxSListFree (OSRTSList * pList)
- void rtxSListFreeAll (OSRTSList * pList)
- OSRTSList * rtxSListCreate (OSVOIDARG)
- OSRTSList * rtxSListCreateEx (OSCTXT * pctxt)

- OSRTSListNode * rtxSListAppend (OSRTSList * pList, void * pData)
- OSBOOL rtxSListFind (OSRTSList * pList, void * pData)
- void rtxSListRemove (OSRTSList * pList, void * pData)

Detailed Description

Definition in file rtxSList.h

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"
```

Macros

- #define OSRTSOCKET_INVALID ((OSRTSOCKET)-1)
- #define OSIPADDR_ANY ((OSIPADDR)0)
- #define OSIPADDR_LOCAL ((OSIPADDR)0x7f000001UL) /* 127.0.0.1 */

Typedefs

- typedef int OSRTSOCKET
- typedef unsigned long OSIPADDR

Functions

- int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET * pNewSocket, OSIPADDR * destAddr, int * destPort)
- int rtxSocketAddrToStr (OSIPADDR ipAddr, char * pbuf, size_t bufsize)
- int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)
- int rtxSocketClose (OSRTSOCKET socket)

- int rtxSocketConnect (OSRTSOCKET socket, const char * host, int port)
- int rtxSocketConnectTimed (OSRTSOCKET socket, const char * host, int port, int nsecs)
- int rtxSocketCreate (OSRTSOCKET * psocket)
- int rtxSocketCreateUDP (OSRTSOCKET * psocket)
- int rtxSocketGetHost (const char * host, struct in_addr * inaddr)
- int rtxSocketsInit (OSVOIDARG)
- int rtxSocketListen (OSRTSOCKET socket, int maxConnection)
- int rtxSocketParseURL (char * url, char ** protocol, char ** address, int * port)
- int rtxSocketRecv (OSRTSOCKET socket, OSOCTET * pbuf, size_t bufsize)
- int rtxSocketRecvTimed (OSRTSOCKET socket, OSOCTET * pbuf, size_t bufsize, OSUINT32 secs)
- int rtxSocketSelect (int nfds, fd_set * readfds, fd_set * writefds, fd_set * exceptfds, struct timeval * timeout)
- int rtxSocketSend (OSRTSOCKET socket, const OSOCTET * pdata, size_t size)
- int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)
- int rtxSocketStrToAddr (const char * pIPAddrStr, OSIPADDR * pIPAddr)

Detailed Description

Definition in file rtxSocket.h

rtxStack.h File Reference

`#include "rtxsrc/rtxDList.h"`

Classes

- struct _OSRTStack

Macros

- #define rtxStackIsEmpty (OSBOOL)((stack).dlist.count == 0)

Typedefs

- typedef struct _OSRTStack OSRTStack

Functions

- OSRTStack * rtxStackCreate (struct OSCTXT * pctxt)
- void rtxStackInit (struct OSCTXT * pctxt, OSRTStack * pStack)

- void * rtxStackPop (OSRTStack * pStack)
- int rtxStackPush (OSRTStack * pStack, void * pData)
- void * rtxStackPeek (OSRTStack * pStack)

Detailed Description

Simple FIFO stack for storing void pointers to any type of data.

Definition in file rtxStack.h

rtxStream.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

```
#include "rtxsrc/rtxMemBuf.h"
```

Classes

- struct OSRTSTREAM

Macros

- #define OSRTSTRMF_INPUT 0x0001
- #define OSRTSTRMF_OUTPUT 0x0002
- #define OSRTSTRMF_BUFFERED 0x8000 /* direct-buffer stream */
- #define OSRTSTRMF_UNBUFFERED 0x4000 /* force unbuffered stream */
- #define OSRTSTRMF_POSMARKED 0x2000 /* stream has marked position */
- #define OSRTSTRMF_FIXINMEM 0x1000 /* disable flushing */
- #define OSRTSTRMF_HEXTEXT 0x0800 /* do hex text / binary conversion */
- #define OSRTSTRMF_BUF_INPUT (OSRTSTRMF_INPUT|OSRTSTRMF_BUFFERED)
- #define OSRTSTRMF_BUF_OUTPUT (OSRTSTRMF_OUTPUT|OSRTSTRMF_BUFFERED)
- #define OSRTSTRMID_FILE 1
- #define OSRTSTRMID_SOCKET 2
- #define OSRTSTRMID_MEMORY 3
- #define OSRTSTRMID_BUFFERED 4
- #define OSRTSTRMID_DIRECTBUF 5
- #define OSRTSTRMID_CTXTBUF 6
- #define OSRTSTRMID_ZLIB 7

- #define OSRTSTRMID_USER 1000
- #define OSRTSTRM_K_BUFSIZE 1024
- #define OSRTSTRM_K_INVALIDMARK ((size_t)-1)
- #define OSRTSTREAM_BYTEINDEX (((pctxt)->pStream->flags & OSRTSTRMF_BUFFERED) ? \ ((pctxt)->pStream->bytesProcessed + (pctxt)->buffer.byteIndex) : \ ((pctxt)->pStream->bytesProcessed /* was ioBytes */))
- #define OSRTSTREAM_ID ((pctxt)->pStream->id)
- #define OSRTSTREAM_FLAGS ((pctxt)->pStream->flags)
- #define rtxStreamBlockingRead rtxStreamRead

Typedefs

- typedef long(* OSRTStreamReadProc
- typedef OSRTStreamReadProc OSRTStreamBlockingReadProc
- typedef long(* OSRTStreamWriteProc
- typedef int(* OSRTStreamFlushProc
- typedef int(* OSRTStreamCloseProc
- typedef int(* OSRTStreamSkipProc
- typedef int(* OSRTStreamMarkProc
- typedef int(* OSRTStreamResetProc
- typedef int(* OSRTStreamGetPosProc
- typedef int(* OSRTStreamSetPosProc
- typedef struct OSRTSTREAM OSRTSTREAM

Functions

- int rtxStreamClose (OSCTXT * pctxt)
- int rtxStreamFlush (OSCTXT * pctxt)
- int rtxStreamLoadInputBuffer (OSCTXT * pctxt, OSSIZE nbytes)
- int rtxStreamInit (OSCTXT * pctxt)
- int rtxStreamInitCtxtBuf (OSCTXT * pctxt)
- int rtxStreamRemoveCtxtBuf (OSCTXT * pctxt)
- long rtxStreamRead (OSCTXT * pctxt, OSOCTET * pbuffer, size_t nocts)
- long rtxStreamRead2 (OSCTXT * pctxt, OSOCTET * pbuffer, size_t nocts, size_t bufSize)

- long rtxStreamReadDirect (OSCTXT * pctxt, OSOCTET * pbuffer, OSSIZE bufSize)
- int rtxStreamSkip (OSCTXT * pctxt, size_t skipBytes)
- long rtxStreamWrite (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)
- int rtxStreamGetIOBytes (OSCTXT * pctxt, size_t * pPos)
- int rtxStreamMark (OSCTXT * pctxt, size_t readAheadLimit)
- int rtxStreamReset (OSCTXT * pctxt)
- OSBOOL rtxStreamMarkSupported (OSCTXT * pctxt)
- OSBOOL rtxStreamIsOpened (OSCTXT * pctxt)
- OSBOOL rtxStreamIsReadable (OSCTXT * pctxt)
- OSBOOL rtxStreamIsWritable (OSCTXT * pctxt)
- int rtxStreamRelease (OSCTXT * pctxt)
- void rtxStreamSetCapture (OSCTXT * pctxt, OSRTMEMBUF * pmembuf)
- OSRTMEMBUF * rtxStreamGetCapture (OSCTXT * pctxt)
- int rtxStreamGetPos (OSCTXT * pctxt, size_t * ppos)
- int rtxStreamSetPos (OSCTXT * pctxt, size_t pos)

Detailed Description

Input/output data stream type definitions and function prototypes.

Definition in file rtxStream.h

rtxStreamFile.h File Reference

```
#include <stdio.h>
```

```
#include "rtxsrc/rtxStream.h"
```

Functions

- int rtxStreamFileAttach (OSCTXT * pctxt, FILE * pFile, OSUINT16 flags)
- int rtxStreamFileOpen (OSCTXT * pctxt, const char * pFilename, OSUINT16 flags)
- int rtxStreamFileCreateReader (OSCTXT * pctxt, const char * pFilename)
- int rtxStreamFileCreateWriter (OSCTXT * pctxt, const char * pFilename)

Detailed Description

Definition in file rtxStreamFile.h

rtxStreamHexText.h File Reference

```
#include "rtxsrc/osSysTypes.h"
```

```
#include "rtxsrc/rtxExternDefs.h"
```

```
#include "rtxsrc/rtxStream.h"
```

Functions

- int rtxStreamHexTextAttach (OSCTXT * pctxt, OSUINT16 flags)

Detailed Description

Definition in file rtxStreamHexText.h

rtxStreamMemory.h File Reference

```
#include "rtxsrc/rtxStream.h"
```

Classes

- struct DirBufDesc

Typedefs

- typedef struct DirBufDesc DirBufDesc

Functions

- int rtxStreamMemoryCreate (OSCTXT * pctxt, OSUINT16 flags)
- int rtxStreamMemoryAttach (OSCTXT * pctxt, OSOCTET * pMemBuf, size_t bufSize, OSUINT16 flags)
- OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT * pctxt, size_t * pSize)
- int rtxStreamMemoryCreateReader (OSCTXT * pctxt, OSOCTET * pMemBuf, size_t bufSize)
- int rtxStreamMemoryCreateWriter (OSCTXT * pctxt, OSOCTET * pMemBuf, size_t bufSize)
- int rtxStreamMemoryResetWriter (OSCTXT * pctxt)

Detailed Description

Definition in file rtxStreamMemory.h

rtxStreamSocket.h File Reference

```
#include "rtxsrc/rtxStream.h"
```

```
#include "rtxsrc/rtxSocket.h"
```

Functions

- int rtxStreamSocketAttach (OSCTXT * pctxt, OSRTSOCKET socket, OSUINT16 flags)
- int rtxStreamSocketClose (OSCTXT * pctxt)
- int rtxStreamSocketCreateWriter (OSCTXT * pctxt, const char * host, int port)
- int rtxStreamSocketSetOwnership (OSCTXT * pctxt, OSBOOL ownSocket)
- int rtxStreamSocketSetReadTimeout (OSCTXT * pctxt, OSUINT32 nsecs)

Detailed Description

Definition in file rtxStreamSocket.h

rtxUTF8.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

Macros

- #define rtxUTF8StrToInt32 rtxUTF8StrToInt
- #define rtxUTF8StrToUInt32 rtxUTF8StrToUInt
- #define RTUTF8STRCMPL rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)
- #define OSRTCHKUTF8LEN do { size_t nchars = rtxUTF8Len (str); \ stat = (nchars >= lower && nchars <= upper) ? 0 : RTERR_CONSVIO; } while(0)

Functions

- long rtxUTF8ToUnicode (OSCTXT * pctxt, const OSUTF8CHAR * inbuf, OSUNICHAR * outbuf, size_t outbufsiz)
- long rtxUTF8ToUnicode32 (OSCTXT * pctxt, const OSUTF8CHAR * inbuf, OS32BITCHAR * outbuf, size_t outbufsiz)
- int rtxValidateUTF8 (OSCTXT * pctxt, const OSUTF8CHAR * inbuf)
- size_t rtxUTF8Len (const OSUTF8CHAR * inbuf)
- size_t rtxCalcUTF8Len (const OSUTF8CHAR * inbuf, size_t inbufBytes)
- size_t rtxUTF8LenBytes (const OSUTF8CHAR * inbuf)
- int rtxUTF8CharSize (OS32BITCHAR wc)
- int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET * buf, size_t bufsiz)
- int rtxUTF8DecodeChar (OSCTXT * pctxt, const OSUTF8CHAR * pinbuf, int * pInsize)
- OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR * buf, OSUINT32 * len)

- OSUTF8CHAR * rtxUTF8StrChr (OSUTF8CHAR * utf8str, OS32BITCHAR utf8char)
- OSUTF8CHAR * rtxUTF8Strdup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str)
- OSUTF8CHAR * rtxUTF8Strndup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, size_t nbytes)
- OSUTF8CHAR * rtxUTF8StrRefOrDup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str)
- OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2)
- OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2, size_t count)
- int rtxUTF8Strcmp (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2)
- int rtxUTF8Strncmp (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2, size_t count)
- OSUTF8CHAR * rtxUTF8Strcpy (OSUTF8CHAR * dest, size_t bufsiz, const OSUTF8CHAR * src)
- OSUTF8CHAR * rtxUTF8Strncpy (OSUTF8CHAR * dest, size_t bufsiz, const OSUTF8CHAR * src, size_t nchars)
- OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR * str)
- const OSUTF8CHAR * rtxUTF8StrJoin (OSCTXT * pctxt, const OSUTF8CHAR * str1, const OSUTF8CHAR * str2, const OSUTF8CHAR * str3, const OSUTF8CHAR * str4, const OSUTF8CHAR * str5)
- int rtxUTF8StrToBool (const OSUTF8CHAR * utf8str, OSBOOL * pvalue)
- int rtxUTF8StrnToBool (const OSUTF8CHAR * utf8str, size_t nbytes, OSBOOL * pvalue)
- int rtxUTF8StrToDouble (const OSUTF8CHAR * utf8str, OSREAL * pvalue)
- int rtxUTF8StrnToDouble (const OSUTF8CHAR * utf8str, size_t nbytes, OSREAL * pvalue)
- int rtxUTF8StrToInt (const OSUTF8CHAR * utf8str, OSINT32 * pvalue)
- int rtxUTF8StrnToInt (const OSUTF8CHAR * utf8str, size_t nbytes, OSINT32 * pvalue)
- int rtxUTF8StrToUInt (const OSUTF8CHAR * utf8str, OSUINT32 * pvalue)
- int rtxUTF8StrnToUInt (const OSUTF8CHAR * utf8str, size_t nbytes, OSUINT32 * pvalue)
- int rtxUTF8StrToSize (const OSUTF8CHAR * utf8str, size_t * pvalue)
- int rtxUTF8StrnToSize (const OSUTF8CHAR * utf8str, size_t nbytes, size_t * pvalue)
- int rtxUTF8StrToInt64 (const OSUTF8CHAR * utf8str, OSINT64 * pvalue)
- int rtxUTF8StrnToInt64 (const OSUTF8CHAR * utf8str, size_t nbytes, OSINT64 * pvalue)
- int rtxUTF8StrToUInt64 (const OSUTF8CHAR * utf8str, OSUINT64 * pvalue)
- int rtxUTF8StrnToUInt64 (const OSUTF8CHAR * utf8str, size_t nbytes, OSUINT64 * pvalue)
- int rtxUTF8ToDynUniStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OSUNICHAR ** ppdata, OSUINT32 * pnchars)
- int rtxUTF8ToDynUniStr32 (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OS32BITCHAR ** ppdata, OSUINT32 * pnchars)

- int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR * utf8instr, size_t nbytes, const OSUTF8CHAR ** putf8outstr)
- int rtxUTF8StrToDynHexStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, OSDynOctStr * pvalue)
- int rtxUTF8StrnToDynHexStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, size_t nbytes, OSDynOctStr * pvalue)
- int rtxUTF8StrToNamedBits (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OSBitMapItem * pBitMap, OSOCTET * pvalue, OSUINT32 * pnbits, OSUINT32 bufsize)
- const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR * utf8str, OSUTF8CHAR ** ppNext)

Detailed Description

Utility functions for handling UTF-8 strings.

Definition in file rtxUTF8.h