The software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement.

Copyright Notice
Copyright ©1997-2008 Objective Systems, Inc. All rights reserved.
This document may be distributed in any form, electronic or otherwise, provided that it is distributed in its entirety and that the copyright and this notice are included.

Author’s Contact Information
Comments, suggestions, and inquiries regarding XBinder may be submitted via electronic mail to info@obj-sys.com.
5.4.2.13 rtxGetCurrDateTime ............................................. 28
5.4.2.14 rtxGetDateTime ................................................. 28
5.4.2.15 rtxGMonthDayToString ......................................... 28
5.4.2.16 rtxGMonthToString ............................................ 29
5.4.2.17 rtxGYearMonthToString ........................................ 29
5.4.2.18 rtxGYearToString .............................................. 29
5.4.2.19 rtxMSecsToDuration ........................................... 30
5.4.2.20 rtxParseDateString ........................................... 30
5.4.2.21 rtxParseDateTimeString ....................................... 30
5.4.2.22 rtxParseGDayString ........................................... 31
5.4.2.23 rtxParseGMonthDayString ..................................... 31
5.4.2.24 rtxParseGMonthString ......................................... 32
5.4.2.25 rtxParseGYearMonthString .................................... 32
5.4.2.26 rtxParseGYearString ........................................... 32
5.4.2.27 rtxParseTimeString ............................................ 33
5.4.2.28 rtxSetDateTime ................................................ 33
5.4.2.29 rtxSetLocalDateTime .......................................... 34
5.4.2.30 rtxSetUtcDateTime ............................................ 34
5.4.2.31 rtxTimeIsValid ................................................. 34
5.4.2.32 rtxTimeToString ............................................... 35

5.5 Decimal number utility functions .................................. 36
5.5.1 Detailed Description .............................................. 36

5.6 Diagnostic trace functions ........................................... 37
5.6.1 Detailed Description .............................................. 38
5.6.2 Function Documentation .......................................... 38
5.6.2.1 rtxDiagEnabled ................................................ 38
5.6.2.2 rtxDiagHexDump ............................................... 38
5.6.2.3 rtxDiagPrint .................................................. 39
5.6.2.4 rtxDiagPrintChars ............................................. 39
5.6.2.5 rtxDiagSetTraceLevel ......................................... 39
5.6.2.6 rtxDiagStream ................................................. 40
5.6.2.7 rtxDiagStreamHexDump ....................................... 40
5.6.2.8 rtxDiagStreamPrintChars .................................... 40
5.6.2.9 rtxDiagToStream ............................................... 40
5.6.2.10 rtxPrintStreamRelease ...................................... 41
5.6.2.11 rtxPrintToStream ............................................. 41
5.6.2.12 rtxSetDiag .................................................... 41
5.6.2.13 rtxSetGlobalDiag .................................................. 41
5.6.2.14 rtxSetGlobalPrintStream ..................................... 42
5.6.2.15 rtxSetPrintStream ................................................ 42

5.7 Doubly-Linked List Utility Functions .................................. 43
5.7.1 Detailed Description .................................................. 44
5.7.2 Function Documentation .............................................. 44
  5.7.2.1 rtxDListAppend ............................................... 44
  5.7.2.2 rtxDListAppendArray ........................................ 45
  5.7.2.3 rtxDListAppendArrayCopy .................................... 45
  5.7.2.4 rtxDListFindByData .......................................... 45
  5.7.2.5 rtxDListFindByIndex ......................................... 46
  5.7.2.6 rtxDListFindIndexByData .................................... 46
  5.7.2.7 rtxDListFreeAll ............................................... 46
  5.7.2.8 rtxDListFreeNode ............................................. 46
  5.7.2.9 rtxDListFreeNodes ............................................ 47
  5.7.2.10 rtxDListInit .................................................. 47
  5.7.2.11 rtxDListInsert ................................................ 47
  5.7.2.12 rtxDListInsertAfter .......................................... 47
  5.7.2.13 rtxDListInsertBefore ........................................ 48
  5.7.2.14 rtxDListRemove ............................................... 48
  5.7.2.15 rtxDListToArray .............................................. 48
  5.7.2.16 rtxDListToUTF8Str .......................................... 49

5.8 Enumeration utility functions ......................................... 50
5.8.1 Detailed Description ................................................ 50
5.8.2 Function Documentation .............................................. 50
  5.8.2.1 rtxLookupEnum ................................................ 50
  5.8.2.2 rtxLookupEnumByValue ....................................... 50
  5.8.2.3 rtxTestNumericEnum ......................................... 51

5.9 Run-time error status codes ........................................... 52
5.9.1 Detailed Description ................................................ 55
5.9.2 Define Documentation ............................................... 56
  5.9.2.1 RT_OK_FRAG .................................................. 56
  5.9.2.2 RTERR_ADDRINUSE .......................................... 56
  5.9.2.3 RTERR_ATTRFIXEDVAL ....................................... 56
  5.9.2.4 RTERR_ATTRMISRQ ........................................... 56
  5.9.2.5 RTERR_BADVALUE ............................................ 56
  5.9.2.6 RTERR_BUFOVFLW ............................................ 56
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9.2.7</td>
<td>RTERR_CONNREFUSED</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.8</td>
<td>RTERR_CONNRESET</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.9</td>
<td>RTERR_CONSVIO</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.10</td>
<td>RTERR_DECATTRFAIL</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.11</td>
<td>RTERR_DECELEMFAIL</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.12</td>
<td>RTERR_EOFBUF</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.13</td>
<td>RTERR_EOFFILE</td>
<td>57</td>
</tr>
<tr>
<td>5.9.2.14</td>
<td>RTERR_EXPIRED</td>
<td>58</td>
</tr>
<tr>
<td>5.9.2.15</td>
<td>RTERR_FAILED</td>
<td>58</td>
</tr>
<tr>
<td>5.9.2.16</td>
<td>RTERR_FILNOTFOU</td>
<td>58</td>
</tr>
<tr>
<td>5.9.2.17</td>
<td>RTERR_HOSTNOTFOU</td>
<td>58</td>
</tr>
<tr>
<td>5.9.2.18</td>
<td>RTERR_HTTPERR</td>
<td>58</td>
</tr>
<tr>
<td>5.9.2.19</td>
<td>RTERR_IDNOTFOU</td>
<td>58</td>
</tr>
<tr>
<td>5.9.2.20</td>
<td>RTERR_INVATTR</td>
<td>59</td>
</tr>
<tr>
<td>5.9.2.21</td>
<td>RTERR_INVBASE64</td>
<td>59</td>
</tr>
<tr>
<td>5.9.2.22</td>
<td>RTERR_INVCHAR</td>
<td>59</td>
</tr>
<tr>
<td>5.9.2.23</td>
<td>RTERR_INVENUM</td>
<td>59</td>
</tr>
<tr>
<td>5.9.2.24</td>
<td>RTERR_INVFORMAT</td>
<td>59</td>
</tr>
<tr>
<td>5.9.2.25</td>
<td>RTERR_INVHEXS</td>
<td>59</td>
</tr>
<tr>
<td>5.9.2.26</td>
<td>RTERR_INVMSGBUF</td>
<td>60</td>
</tr>
<tr>
<td>5.9.2.27</td>
<td>RTERR_INVOCCUR</td>
<td>60</td>
</tr>
<tr>
<td>5.9.2.28</td>
<td>RTERR_INVOPT</td>
<td>60</td>
</tr>
<tr>
<td>5.9.2.29</td>
<td>RTERR_INVPARAM</td>
<td>60</td>
</tr>
<tr>
<td>5.9.2.30</td>
<td>RTERR_INVREAL</td>
<td>60</td>
</tr>
<tr>
<td>5.9.2.31</td>
<td>RTERR_INVSOCKET</td>
<td>60</td>
</tr>
<tr>
<td>5.9.2.32</td>
<td>RTERR_INVSOCKOPT</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.33</td>
<td>RTERR_INVUTF8</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.34</td>
<td>RTERR_MULTIPLE</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.35</td>
<td>RTERR_NOCONN</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.36</td>
<td>RTERR_NOMEM</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.37</td>
<td>RTERR_NOTINIT</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.38</td>
<td>RTERR_NOTINSET</td>
<td>61</td>
</tr>
<tr>
<td>5.9.2.39</td>
<td>RTERR_NOTSUPP</td>
<td>62</td>
</tr>
<tr>
<td>5.9.2.40</td>
<td>RTERR_NOTYPEINFO</td>
<td>62</td>
</tr>
<tr>
<td>5.9.2.41</td>
<td>RTERR_NULLPTR</td>
<td>62</td>
</tr>
<tr>
<td>5.9.2.42</td>
<td>RTERR_OUTOFBND</td>
<td>62</td>
</tr>
<tr>
<td>5.9.2.43</td>
<td>RTERR_PATMATCH</td>
<td>62</td>
</tr>
</tbody>
</table>
5.9.2.44 RTERR_READERR .................................................. 62
5.9.2.45 RTERR_REGEXP .................................................. 63
5.9.2.46 RTERR_SEQORDER ................................................. 63
5.9.2.47 RTERR_SEQOVFLW .............................................. 63
5.9.2.48 RTERR_SETDUPL ................................................ 63
5.9.2.49 RTERR_SETMISRQ ............................................... 63
5.9.2.50 RTERR_SOAPERR ............................................... 63
5.9.2.51 RTERR_STRMINUSE ............................................. 64
5.9.2.52 RTERR_STROVFLW ............................................. 64
5.9.2.53 RTERR_TOOBIG ................................................ 64
5.9.2.54 RTERR_TOODEEP ............................................... 64
5.9.2.55 RTERR_UNEXPELEM ........................................... 64
5.9.2.56 RTERR_UNREACHABLE ......................................... 64
5.9.2.57 RTERR_WRITEERR ............................................. 64
5.9.2.58 RTERR_XMLPARSE ............................................. 65
5.9.2.59 RTERR_XMLSTATE ............................................... 65

5.10 Error Formatting and Print Functions .................................. 66
  5.10.1 Detailed Description ............................................. 68
  5.10.2 Define Documentation .......................................... 68
      5.10.2.1 LOG_RTERR .................................................. 68
      5.10.2.2 OSRTASSERT ............................................... 68
      5.10.2.3 OSRTCHECKPARAM ......................................... 68
  5.10.3 Function Documentation ....................................... 69
      5.10.3.1 rtxErrAddCtxtBufParm .................................. 69
      5.10.3.2 rtxErrAddDoubleParm ................................... 69
      5.10.3.3 rtxErrAddElemNameParm ................................ 69
      5.10.3.4 rtxErrAddErrorTableEntry ................................ 70
      5.10.3.5 rtxErrAddInt64Parm ..................................... 70
      5.10.3.6 rtxErrAddIntParm ......................................... 70
      5.10.3.7 rtxErrAddStrnParm ....................................... 71
      5.10.3.8 rtxErrAddStrParm ......................................... 71
      5.10.3.9 rtxErrAddUInt64Parm .................................... 71
      5.10.3.10 rtxErrAddUIntParm ..................................... 71
      5.10.3.11 rtxErrAddUniStrParm .................................. 72
      5.10.3.12 rtxErrAssertionFailed .................................. 72
      5.10.3.13 rtxErrFmtMsg ........................................... 72
      5.10.3.14 rtxErrFreeParms ......................................... 72
5.16.2.8 rtxIsPlusInfinity .................................................. 104

5.17 Doubly-Linked List Utility Functions ........................................ 105

5.17.1 Detailed Description .................................................. 105

5.17.2 Function Documentation .............................................. 106

5.17.2.1 rtxScalarDListAppendDouble .................................... 106

5.17.2.2 rtxScalarDListAppendNode ........................................ 106

5.17.2.3 rtxScalarDListFindByIndex ...................................... 106

5.17.2.4 rtxScalarDListFreeNode ........................................... 107

5.17.2.5 rtxScalarDListFreeNodes ......................................... 107

5.17.2.6 rtxScalarDListInit ................................................. 107

5.17.2.7 rtxScalarDListInsertNode ......................................... 107

5.17.2.8 rtxScalarDListRemove ............................................. 108

5.18 TCP/IP or UDP socket utility functions .................................... 109

5.18.1 Typedef Documentation .............................................. 110

5.18.1.1 OSIPADDR .......................................................... 110

5.18.2 Function Documentation .............................................. 110

5.18.2.1 rtxSocketAccept .................................................. 110

5.18.2.2 rtxSocketAddrToStr ............................................... 110

5.18.2.3 rtxSocketBind ..................................................... 110

5.18.2.4 rtxSocketClose .................................................... 111

5.18.2.5 rtxSocketConnect ............................................... 111

5.18.2.6 rtxSocketCreate .................................................. 111

5.18.2.7 rtxSocketGetHost ............................................... 112

5.18.2.8 rtxSocketListen .................................................. 112

5.18.2.9 rtxSocketParseURL ............................................... 112

5.18.2.10 rtxSocketRecv .................................................. 113

5.18.2.11 rtxSocketSend .................................................. 113

5.18.2.12 rtxSocketsInit .................................................. 113

5.18.2.13 rtxSocketStrToAddr ............................................. 113

5.19 Input/Output Data Stream Utility Functions ................................... 115

5.19.1 Detailed Description .................................................. 116

5.19.2 Typedef Documentation .............................................. 117

5.19.2.1 OSRTStreamBlockingReadProc .................................. 117

5.19.2.2 OSRTStreamCloseProc ........................................... 117

5.19.2.3 OSRTStreamFlushProc ........................................... 117

5.19.2.4 OSRTStreamMarkProc ............................................ 117

5.19.2.5 OSRTStreamReadProc ............................................ 118
5.19.2.6 OSRTStreamResetProc ................................................. 118
5.19.2.7 OSRTStreamSkipProc ................................................ 118
5.19.2.8 OSRTStreamWriteProc ............................................... 118

5.19.3 Function Documentation ............................................. 118
5.19.3.1 rtxStreamBlockingRead ........................................... 118
5.19.3.2 rtxStreamClose ...................................................... 119
5.19.3.3 rtxStreamFlush ..................................................... 119
5.19.3.4 rtxStreamGetCapture .............................................. 119
5.19.3.5 rtxStreamGetIOBytes ............................................. 119
5.19.3.6 rtxStreamInit ....................................................... 120
5.19.3.7 rtxStreamIsOpened ............................................... 120
5.19.3.8 rtxStreamIsReadable ............................................. 120
5.19.3.9 rtxStreamIsWritable ............................................ 120
5.19.3.10 rtxStreamMark .................................................. 121
5.19.3.11 rtxStreamMarkSupported ..................................... 121
5.19.3.12 rtxStreamRead .................................................. 121
5.19.3.13 rtxStreamRelease .............................................. 122
5.19.3.14 rtxStreamReset .................................................. 122
5.19.3.15 rtxStreamSetCapture ........................................ 122
5.19.3.16 rtxStreamSkip .................................................. 122
5.19.3.17 rtxStreamWrite ................................................. 123

5.20 File stream functions ................................................. 124
5.20.1 Detailed Description ................................................. 124
5.20.2 Function Documentation ............................................ 124
5.20.2.1 rtxStreamFileAttach ............................................ 124
5.20.2.2 rtxStreamFileCreateReader .................................. 124
5.20.2.3 rtxStreamFileCreateWriter ................................ 125
5.20.2.4 rtxStreamFileOpen ............................................. 125

5.21 Memory stream functions ........................................... 126
5.21.1 Detailed Description ................................................. 126
5.21.2 Function Documentation ............................................ 126
5.21.2.1 rtxStreamMemoryAttach ...................................... 126
5.21.2.2 rtxStreamMemoryCreate ..................................... 127
5.21.2.3 rtxStreamMemoryCreateReader ............................ 127
5.21.2.4 rtxStreamMemoryCreateWriter ................................ 127
5.21.2.5 rtxStreamMemoryGetBuffer .................................. 128

5.22 Socket stream functions ............................................ 129
5.22.1 Detailed Description ................................................. 129
5.22.2 Function Documentation ........................................... 129
  5.22.2.1 rtxStreamSocketAttach ...................................... 129
  5.22.2.2 rtxStreamSocketClose ....................................... 130
  5.22.2.3 rtxStreamSocketCreateWriter ................................ 130
  5.22.2.4 rtxStreamSocketSetOwnership .............................. 130

5.23 UTF-8 String Functions ........................................... 131
  5.23.1 Detailed Description ........................................... 133
  5.23.2 Define Documentation ........................................... 134
    5.23.2.1 RTUTF8STRCMPL ......................................... 134
  5.23.3 Function Documentation ........................................ 134
    5.23.3.1 rtxUTF8CharSize ......................................... 134
    5.23.3.2 rtxUTF8CharToWC ....................................... 134
    5.23.3.3 rtxUTF8DecodeChar ...................................... 134
    5.23.3.4 rtxUTF8EncodeChar ...................................... 135
    5.23.3.5 rtxUTF8Len ............................................... 135
    5.23.3.6 rtxUTF8LenBytes ........................................ 135
    5.23.3.7 rtxUTF8RemoveWhiteSpace ............................... 135
    5.23.3.8 rtxUTF8StrChr ........................................... 136
    5.23.3.9 rtxUTF8Strcmp ........................................... 136
    5.23.3.10 rtxUTF8Strcpy .......................................... 136
    5.23.3.11 rtxUTF8Strdup .......................................... 136
    5.23.3.12 rtxUTF8StrEqual ....................................... 137
    5.23.3.13 rtxUTF8StrHash ........................................ 137
    5.23.3.14 rtxUTF8StrJoin ......................................... 137
    5.23.3.15 rtxUTF8Strncmp .......................................... 138
    5.23.3.16 rtxUTF8Strncpy .......................................... 138
    5.23.3.17 rtxUTF8Strndup ......................................... 138
    5.23.3.18 rtxUTF8StrnEqual ..................................... 139
    5.23.3.19 rtxUTF8StrNextTok .................................... 139
    5.23.3.20 rtxUTF8StrnToBool .................................... 139
    5.23.3.21 rtxUTF8StrnToDouble ................................ 140
    5.23.3.22 rtxUTF8StrnToDynHexStr ............................... 140
    5.23.3.23 rtxUTF8StrnToInt ...................................... 140
    5.23.3.24 rtxUTF8StrnToInt64 .................................. 141
    5.23.3.25 rtxUTF8StrnToUInt .................................... 141
    5.23.3.26 rtxUTF8StrnToUInt64 .................................. 141
6 Class Documentation

6.1 OSBitMapItem Struct Reference

6.1.1 Detailed Description

6.2 OSCTXT Struct Reference

6.2.1 Detailed Description

6.3 OSDynOctStr Struct Reference

6.3.1 Detailed Description

6.4 OSNumDateTime Struct Reference

6.4.1 Detailed Description

6.5 OSRTBuffer Struct Reference

6.5.1 Detailed Description

6.6 OSRTBufSave Struct Reference

6.6.1 Detailed Description

6.7 OSRTDList Struct Reference

6.7.1 Detailed Description

6.7.2 Member Data Documentation

6.7.2.1 count

6.7.2.2 head

6.7.2.3 tail

6.8 OSRTDListNode Struct Reference

6.8.1 Detailed Description

6.8.2 Member Data Documentation

6.8.2.1 data

6.8.2.2 next
6.8.2.3 prev ................................. 153
6.9 OSRTRerrInfo Struct Reference ................................. 154
   6.9.1 Detailed Description .................................... 154
6.10 OSRTRerrLocn Struct Reference ................................. 155
   6.10.1 Detailed Description .................................... 155
6.11 OSRTRprintStream Struct Reference ......................... 156
   6.11.1 Detailed Description .................................... 156
6.12 OSRTRScalarDList Struct Reference ............................... 157
   6.12.1 Detailed Description .................................... 157
6.13 OSRTRScalarDListNode Struct Reference ...................... 158
   6.13.1 Detailed Description .................................... 158
6.14 OSRTRSTREAM Struct Reference ................................. 159
   6.14.1 Detailed Description .................................... 159
6.15 OSXMLFullQName Struct Reference .............................. 160
   6.15.1 Detailed Description .................................... 160
6.16 OSXMLSTRING Struct Reference ................................. 161
   6.16.1 Detailed Description .................................... 161
6.17 OSXSDAny Struct Reference ................................. 162
   6.17.1 Detailed Description .................................... 162
6.18 OSXSDDateTime Struct Reference ................................. 163
   6.18.1 Detailed Description .................................... 163

7 File Documentation 164
7.1 rtxArrayList.h File Reference ................................. 164
   7.1.1 Detailed Description .................................... 165
   7.1.2 Function Documentation ................................ 165
      7.1.2.1 rtxArrayListAdd ................................... 165
      7.1.2.2 rtxArrayListGetIndexed ......................... 165
      7.1.2.3 rtxArrayListHasNextItem ......................... 166
      7.1.2.4 rtxArrayListIndexOf ............................. 166
      7.1.2.5 rtxArrayListInit ...................... 166
      7.1.2.6 rtxArrayListInitIter ......................... 166
      7.1.2.7 rtxArrayListInsert ...................... 167
      7.1.2.8 rtxArrayListNextItem ......................... 167
      7.1.2.9 rtxArrayListRemove .............................. 167
      7.1.2.10 rtxArrayListRemoveIndexed .................... 167
      7.1.2.11 rtxArrayListReplace ......................... 168
7.14.1 Detailed Description ............................................. 185

7.15 rtxDiagBitTrace.h File Reference ................................ 186

7.15.1 Detailed Description ............................................. 187

7.15.2 Function Documentation ........................................ 187

7.15.2.1 rtxDiagBitFieldListInit ................................... 187
7.15.2.2 rtxDiagBitFldAppendNamePart ............................... 187
7.15.2.3 rtxDiagBitTraceEnabled ..................................... 187
7.15.2.4 rtxDiagBitTracePrint ........................................ 188
7.15.2.5 rtxDiagBitTracePrintHTML ................................. 188
7.15.2.6 rtxDiagInsBitFieldLen ...................................... 188
7.15.2.7 rtxDiagNewBitField ........................................ 188
7.15.2.8 rtxDiagSetBitFldCount ..................................... 188
7.15.2.9 rtxDiagSetBitFldDisabled ................................. 189
7.15.2.10 rtxDiagSetBitFldNameSuffix ............................. 189
7.15.2.11 rtxDiagSetBitFldOffset ................................. 189
7.15.2.12 rtxDiagSetBitTraceEnabled .............................. 189

7.16 rtxDList.h File Reference ........................................ 190

7.16.1 Detailed Description ............................................. 191

7.17 rtxDynBitSet.h File Reference .................................... 192

7.17.1 Detailed Description ............................................. 192

7.17.2 Function Documentation ........................................ 192

7.17.2.1 rtxDynBitSetClearBit ..................................... 192
7.17.2.2 rtxDynBitSetCopy ........................................ 193
7.17.2.3 rtxDynBitSetFree .......................................... 193
7.17.2.4 rtxDynBitSetInit .......................................... 193
7.17.2.5 rtxDynBitSetInsertBit .................................. 194
7.17.2.6 rtxDynBitSetSetBit ..................................... 194
7.17.2.7 rtxDynBitSetSetBitToValue .............................. 194
7.17.2.8 rtxDynBitSetTestBit ..................................... 195

7.18 rtxDynPtrArray.h File Reference .................................. 196

7.18.1 Detailed Description ............................................. 196

7.18.2 Function Documentation ........................................ 196

7.18.2.1 rtxDynPtrArrayAppend .................................. 196
7.18.2.2 rtxDynPtrArrayInit ..................................... 196

7.19 rtxEnum.h File Reference .......................................... 198

7.19.1 Detailed Description ............................................. 198

7.20 rtxErrCodes.h File Reference ..................................... 199
7.30.2.1 RT_MH_DONTKEEPFREE ............................................. 220
7.31 rtxPattern.h File Reference ............................................... 221
  7.31.1 Detailed Description .................................................. 221
7.32 rtxPrint.h File Reference ................................................. 222
  7.32.1 Detailed Description .................................................. 224
7.33 rtxPrintStream.h File Reference ........................................ 225
  7.33.1 Detailed Description .................................................. 225
7.34 rtxPrintToStream.h File Reference ..................................... 226
  7.34.1 Detailed Description .................................................. 227
7.35 rtxReal.h File Reference .................................................. 228
  7.35.1 Detailed Description .................................................. 228
7.36 rtxScalarDList.h File Reference ....................................... 229
  7.36.1 Detailed Description .................................................. 230
7.37 rtxSOAP.h File Reference .................................................. 231
  7.37.1 Detailed Description .................................................. 231
    7.37.2 Function Documentation ........................................... 231
      7.37.2.1 rtxSoapAcceptConn ............................................ 231
      7.37.2.2 rtxSoapConnect ............................................... 232
      7.37.2.3 rtxSoapInitConn ............................................... 232
      7.37.2.4 rtxSoapRecvHttp .............................................. 232
      7.37.2.5 rtxSoapRecvHttpContent .................................... 232
      7.37.2.6 rtxSoapSendHttp .............................................. 233
7.38 rtxSocket.h File Reference .............................................. 234
  7.38.1 Detailed Description .................................................. 235
7.39 rtxStream.h File Reference .............................................. 236
  7.39.1 Detailed Description .................................................. 238
7.40 rtxStreamBuffered.h File Reference .................................. 239
  7.40.1 Detailed Description .................................................. 239
7.41 rtxStreamFile.h File Reference ....................................... 240
  7.41.1 Detailed Description .................................................. 240
7.42 rtxStreamMemory.h File Reference ................................... 241
  7.42.1 Detailed Description .................................................. 241
7.43 rtxStreamSocket.h File Reference ................................... 242
  7.43.1 Detailed Description .................................................. 242
7.44 rtxUnicode.h File Reference ........................................... 243
  7.44.1 Detailed Description .................................................. 243
    7.44.2 Function Documentation ......................................... 244
7.44.2.1 rtxUCSIsBaseChar .................................................. 244
7.44.2.2 rtxUCSIsBlank ..................................................... 244
7.44.2.3 rtxUCSIsChar ....................................................... 244
7.44.2.4 rtxUCSIsCombining ............................................... 244
7.44.2.5 rtxUCSIsDigit ...................................................... 245
7.44.2.6 rtxUCSIsExtender .................................................. 245
7.44.2.7 rtxUCSIsIdeographic ............................................. 245
7.44.2.8 rtxUCSIsLetter ..................................................... 245
7.44.2.9 rtxUCSIsPubidChar ............................................... 245
7.44.2.10 rtxUCSToDynUTF8 ............................................. 246
7.44.2.11 rtxUCSToUTF8 ................................................... 246

7.45 rtxUTF8.h File Reference .................................................. 247
7.45.1 Detailed Description .................................................. 249

7.46 rtxXmlQName.h File Reference ......................................... 250
7.46.1 Detailed Description .................................................. 250
7.46.2 Function Documentation ............................................. 251
7.46.2.1 rtxNewFullQName .................................................. 251
7.46.2.2 rtxNewFullQNameDeepCopy .................................... 251
7.46.2.3 rtxQNameDeepCopy ............................................... 251
7.46.2.4 rtxQNameFreeMem ................................................ 252
7.46.2.5 rtxQNameHash ...................................................... 252
7.46.2.6 rtxQNamesEqual .................................................. 252
7.46.2.7 rtxQNameToString ............................................... 252

7.47 rtxXmlStr.h File Reference ............................................. 253
7.47.1 Detailed Description ................................................ 253
7.47.2 Function Documentation ............................................ 253
7.47.2.1 rtxCreateCopyXmlStr ............................................. 253
7.47.2.2 rtxCreateXmlStr .................................................. 254
Chapter 1

Main Page

C Common Runtime Library Functions

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

The categories of functions provided are as follows:

- Context management functions handle the allocation, initialization, and destruction of context variables (variables of type OSCTXT) that handle the working data used during the encoding or decoding of a message.
- Memory allocation macros and functions provide an optimized memory management interface.
- Doubly linked list (DList) functions are used to manipulate linked list structures that are used to model repeating XSD types and elements.
- UTF-8 and Unicode character string functions provide support for conversions to and from these formats in C or C++.
- Date/time conversion functions provide utilities for converting system and structured numeric date/time values to XML schema standard string format.
- Pattern matching function compare strings against patterns specified using regular expressions (regexp’s).
- Diagnostic trace functions allow the output of trace messages to standard output.
- Error formatting and print functions allow information about encode/decode errors to be added to a context block structure and printed out.
- Memory buffer management functions handle the allocation, expansion, and de-allocation of dynamic memory buffers used by some encode/decode functions.
- Formatted print functions allow binary data to be formatted and printed to standard output and other output devices.
- Big Integer helper functions are arbitrary-precision integer manipulating functions used to maintain big integers.
Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Bit String Functions .................................................. 6
Character string functions ......................................... 9
Context Management Functions ................................... 14
Date/time conversion functions ................................. 21
Decimal number utility functions ................................. 36
Diagnostic trace functions ......................................... 37
Doubly-Linked List Utility Functions ......................... 43
Enumeration utility functions .................................. 50
Run-time error status codes .................................... 52
Error Formatting and Print Functions ......................... 66
Memory Buffer Management Functions ...................... 77
Memory Allocation Macros and Functions .................. 82
Pattern matching functions ....................................... 87
Print Functions .......................................................... 88
Print-To-Stream Functions ......................................... 96
Floating-point number utility functions ................... 103
Doubly-Linked List Utility Functions ......................... 105
TCP/IP or UDP socket utility functions .................... 109
Input/Output Data Stream Utility Functions ............... 115
File stream functions ............................................... 124
Memory stream functions ......................................... 126
Socket stream functions ........................................... 129
UTF-8 String Functions ............................................ 131
Chapter 3

Class Index

3.1 Class List

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

- **OSBitMapItem** (Named bit in a bit map) .......................................................... 146
- **OSCTX** (Run-time context structure) ................................................................. 147
- **OSDynOctStr** (Dynamic binary string structure) ............................................... 148
- **OSNumDateTime** (Numeric date/time structure) ................................................ 149
- **OSRBuffer** (Run-time message buffer structure) ............................................. 150
- **OSRBufSave** (Structure to save the current message buffer state) .................. 151
- **OSRTDList** (This is the main list structure) ....................................................... 152
- **OSRTDListNode** (This structure is used to hold a single data item within the list) 153
- **OSRTErrInfo** (Run-time error information structure) ....................................... 154
- **OSRTErrLocn** (Run-time error location structure) ............................................ 155
- **OSRTPrintStream** (Structure to hold information about a global PrintStream) .... 156
- **OSRTScalarDList** (This is the main list structure) .......................................... 157
- **OSRTScalarDListNode** (This structure is used to hold a single data item within the list) 158
- **OSRTSTREAM** (The stream control block) ....................................................... 159
- **OSXMLFullQName** (This version of QName contains complete namespace info (prefix + URI)) 160
- **OSXMLSTRING** (XML UTF-8 character string structure) ............................... 161
- **OSXSDAny** (Structure to hold xsd:any data in binary and XML text form) .......... 162
- **OSXSDDateTime** (Numeric date/time structure) .................................................. 163
Chapter 4

File Index

4.1 File List

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

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

- **osSysTypes.h**

- **rtxArrayList.h**
  - ArrayList functions

- **rtxBase64.h**

- **rtxBigInt.h**

- **rtxBitDecode.h**
  - Bit decode functions

- **rtxBitEncode.h**
  - Bit encode functions

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

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

- **rtxCharStr.h**

- **rtxClock.h**

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

- **rtxContext.h**
  - Common run-time context definitions

- **rtxCtype.h**

- **rtxDateTime.h**
  - Common runtime functions for converting to and from various standard date/time formats

- **rtxDecimal.h**
  - Common runtime functions for working with xsd:decimal numbers

- **rtxDiag.h**
  - Common runtime functions for diagnostic tracing and debugging

- **rtxDiagBitTrace.h**
  - Common runtime functions for tracing bit patterns written to or read from a stream

- **rtxDList.h**
  - Doubly-Linked List Utility Functions

- **rtxDynBitSet.h**
  - Implementation of a dynamic bit set similar to the Java BitSet class

- **rtxDynPtrArray.h**
  - Implementation of a dynamic pointer array

- **rtxEnum.h**
  - Common runtime types and functions for performing operations on enumerated data items
<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rtxErrCodes.h</code></td>
<td>(List of numeric status codes that can be returned by common run-time functions and generated code)</td>
<td>199</td>
</tr>
<tr>
<td><code>rtxError.h</code></td>
<td>(Error handling function and macro definitions)</td>
<td>203</td>
</tr>
<tr>
<td><code>rtxExternDefs.h</code></td>
<td>(Common definitions of external function modifiers used to define the scope of functions used in DLL’s (Windows only))</td>
<td>206</td>
</tr>
<tr>
<td><code>rtxFile.h</code></td>
<td>(Common runtime functions for reading from or writing to files)</td>
<td>207</td>
</tr>
<tr>
<td><code>rtxFloat.h</code></td>
<td></td>
<td>210</td>
</tr>
<tr>
<td><code>rtxHashMap.h</code></td>
<td>(Generic hash map interface)</td>
<td>211</td>
</tr>
<tr>
<td><code>rtxHashMapStr2Int.h</code></td>
<td>(String-to-integer hash map interface)</td>
<td>215</td>
</tr>
<tr>
<td><code>rtxHashMapStr2UInt.h</code></td>
<td>(String-to-unsigned integer hash map interface)</td>
<td>216</td>
</tr>
<tr>
<td><code>rtxHashMapUndef.h</code></td>
<td>(Undefine all hash map symbols to allow reuse of the basic definitions in a different of the map)</td>
<td>217</td>
</tr>
<tr>
<td><code>rtxMemBuf.h</code></td>
<td></td>
<td>218</td>
</tr>
<tr>
<td><code>rtxMemory.h</code></td>
<td>(Memory management function and macro definitions)</td>
<td>219</td>
</tr>
<tr>
<td><code>rtxPattern.h</code></td>
<td>(Pattern matching functions)</td>
<td>221</td>
</tr>
<tr>
<td><code>rtxPrint.h</code></td>
<td></td>
<td>222</td>
</tr>
<tr>
<td><code>rtxPrintStream.h</code></td>
<td>(Functions that allow printing diagnostic message to a stream using a callback function)</td>
<td>225</td>
</tr>
<tr>
<td><code>rtxPrintToStream.h</code></td>
<td></td>
<td>226</td>
</tr>
<tr>
<td><code>rtxReal.h</code></td>
<td>(Common runtime functions for working with floating-point numbers)</td>
<td>228</td>
</tr>
<tr>
<td><code>rtxScalarDList.h</code></td>
<td>(Doubly-linked list utility functions to hold scalar data variables)</td>
<td>229</td>
</tr>
<tr>
<td><code>rtxSOAP.h</code></td>
<td>(: common SOAP socket communications functions)</td>
<td>231</td>
</tr>
<tr>
<td><code>rtxSocket.h</code></td>
<td></td>
<td>234</td>
</tr>
<tr>
<td><code>rtxStream.h</code></td>
<td>(Input/output data stream type definitions and function prototypes)</td>
<td>236</td>
</tr>
<tr>
<td><code>rtxStreamBuffered.h</code></td>
<td></td>
<td>239</td>
</tr>
<tr>
<td><code>rtxStreamCtxtBuf.h</code></td>
<td></td>
<td>??</td>
</tr>
<tr>
<td><code>rtxStreamFile.h</code></td>
<td></td>
<td>240</td>
</tr>
<tr>
<td><code>rtxStreamMemory.h</code></td>
<td></td>
<td>241</td>
</tr>
<tr>
<td><code>rtxStreamSocket.h</code></td>
<td></td>
<td>242</td>
</tr>
<tr>
<td><code>rtxStreamZlib.h</code></td>
<td></td>
<td>??</td>
</tr>
<tr>
<td><code>rtxUnicode.h</code></td>
<td>(This is an open source header file derived from the libxml2 project)</td>
<td>243</td>
</tr>
<tr>
<td><code>rtxUTF8.h</code></td>
<td>(Utility functions for handling UTF-8 strings)</td>
<td>247</td>
</tr>
<tr>
<td><code>rtxUtil.h</code></td>
<td></td>
<td>??</td>
</tr>
<tr>
<td><code>rtxXmlQName.h</code></td>
<td>(XML QName type definition and associated utility functions)</td>
<td>250</td>
</tr>
<tr>
<td><code>rtxXmlStr.h</code></td>
<td></td>
<td>253</td>
</tr>
</tbody>
</table>
Chapter 5

Module Documentation

5.1 Bit String Functions

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

Defines

- #define OSRTBYTEARRAYSIZE(numbits) (((numbits-1)/8)+1)

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

Functions

- EXTERNRT int rtxSetBit (OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)

  This function sets the specified bit in the bit string.

- EXTERNRT OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)

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

- EXTERNRT int rtxClearBit (OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)

  This function clears the specified bit in the bit string.

- EXTERNRT OSBOOL rtxTestBit (const OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)

  This function tests the specified bit in the bit string.

5.1.1 Detailed Description

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

5.1.2 Function Documentation

5.1.2.1 EXTERNRT int rtxClearBit (OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)

  This function clears the specified bit in the bit string.
Parameters:

\textbf{pBits} Pointer to octets of bit string.
\textbf{numbits} Number of bits in the bit string.
\textbf{bitIndex} Index of bit to be cleared. The bit with index 0 is a most significant bit in the octet with index 0.

Returns:

If successful, returns the previous state of bit. If bit was set the return value is positive, if bit was not set the return value is zero. Otherwise, return value is an error code:

- \textbf{RTERR\_OUTOFBND} = bitIndex is out of bounds

5.1.2.2 \textsc{EXTERNRT int} \texttt{rtxSetBit (OSOCTET} \texttt{*pBits, OSUINT32 numbits, OSUINT32 bitIndex)}

This function sets the specified bit in the bit string.

Parameters:

\textbf{pBits} Pointer to octets of bit string.
\textbf{numbits} Number of bits in the bit string.
\textbf{bitIndex} Index of bit to be set. The bit with index 0 is a most significant bit in the octet with index 0.

Returns:

If successful, returns the previous state of bit. If bit was set the return value is positive, if bit was not set the return value is zero. Otherwise, return value is an error code:

- \textbf{RTERR\_OUTOFBND} = bitIndex is out of bounds

5.1.2.3 \textsc{EXTERNRT OSUINT32} \texttt{rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)}

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

Parameters:

\textbf{flags} Flags to which mask will be applied.
\textbf{mask} Mask with one or more bits set that will be applied to pBitMask.
\textbf{action} Boolean action indicating if bits in flags should be set (TRUE) or cleared (FALSE).

Returns:

Updated flags after mask is applied.

5.1.2.4 \textsc{EXTERNRT OSBOOL} \texttt{rtxTestBit (const OSOCTET} \texttt{*pBits, OSUINT32 numbits, OSUINT32 bitIndex)}

This function tests the specified bit in the bit string.

Parameters:

\textbf{pBits} Pointer to octets of bit string.
**numbits**  Number of bits in the bit string.

**bitIndex**  Index of bit to be tested. The bit with index 0 is a most significant bit in the octet with index 0.

**Returns:**

True if bit set or false if not set or array index is beyond range of number of bits in the string.
5.2 Character string functions

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

Functions

- **EXTERNRT char * rtxStrcat (char *dest, size_t bufsiz, const char *src)**
  
  *This function concatenates the given string onto the string buffer.*

- **EXTERNRT char * rtxStrncat (char *dest, size_t bufsiz, const char *src, size_t nchars)**
  
  *This function concatenates the given number of characters from the given string onto the string buffer.*

- **EXTERNRT char * rtxStrcpy (char *dest, size_t bufsiz, const char *src)**
  
  *This function copies a null-terminated string to a target buffer.*

- **EXTERNRT char * rtxStrncpy (char *dest, size_t bufsiz, const char *src, size_t nchars)**
  
  *This function copies the given number of characters from a string to a target buffer.*

- **EXTERNRT int rtxIntToCharStr (OSINT32 value, char *dest, size_t bufsiz, char padchar)**
  
  *This function converts a signed 32-bit integer into a character string.*

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

- **EXTERNRT int rtxInt64ToCharStr (OSINT64 value, char *dest, size_t bufsiz, char padchar)**
  
  *This function converts a signed 64-bit integer into a character string.*

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

- **EXTERNRT int rtxHexCharsToBinCount (const char *hexstr, size_t nchars)**
  
  *This function returns a count of the number of bytes the would result from the conversion of a hexadecimal character string to binary.*

- **EXTERNRT int rtxHexCharsToBin (const char *hexstr, size_t nchars, OSOCTET *binbuf, size_t bufsize)**
  
  *This function converts the given hex string to binary.*

5.2.1 Detailed Description

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

5.2.2 Function Documentation

5.2.2.1 **EXTERNRT int rtxHexCharsToBin (const char *hexstr, size_t nchars, OSOCTET *binbuf, size_t bufsize)**

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

Parameters:

- **hexstr** Hex character string to convert.
- **nchars** Number of characters in string. If zero, characters are read up to null-terminator.
- **binbuf** Buffer to hold converted binary data.
- **bufsize** Size of the binary data buffer.

Returns:

- Number of bytes or negative status value if fail.

5.2.2.2 **EXTERNRT int rtxHexCharsToBinCount (const char ∗hexstr, size_t nchars)**

This function returns a count of the number of bytes the would result from the conversion of a hexadecimal character string to binary.

Any whitespace characters in the string are ignored.

Parameters:

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

Returns:

- Number of bytes or negative status value if fail.

5.2.2.3 **EXTERNRT int rtxInt64ToCharStr (OSINT64 value, char ∗dest, size_t bufsize, char padchar)**

This function converts a signed 64-bit integer into a character string.

It is similar to the C `itoa` function.

Parameters:

- **value** Integer to convert.
- **dest** Pointer to destination buffer to receive string.
- **bufsize** Size of the destination buffer.
- **padchar** Left pad char, set to zero for no padding.

Returns:

- Number of characters or negative status value if fail.

5.2.2.4 **EXTERNRT int rtxIntToCharStr (OSINT32 value, char ∗dest, size_t bufsize, char padchar)**

This function converts a signed 32-bit integer into a character string.

It is similar to the C `itoa` function.
Parameters:

- **value**: Integer to convert.
- **dest**: Pointer to destination buffer to receive string.
- **bufsiz**: Size of the destination buffer.
- **padchar**: Left pad char, set to zero for no padding.

Returns:

Number of characters or negative status value if fail.

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

This function concatenates the given string onto the string buffer.

It is similar to the C `strcat` function except more secure because it checks for buffer overrun.

Parameters:

- **dest**: Pointer to destination buffer to receive string.
- **bufsiz**: Size of the destination buffer.
- **src**: Pointer to null-terminated string to copy.

Returns:

Pointer to destination buffer or NULL if copy failed.

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

This function copies a null-terminated string to a target buffer.

It is similar to the C `strcpy` function except more secure because it checks for buffer overrun.

Parameters:

- **dest**: Pointer to destination buffer to receive string.
- **bufsiz**: Size of the destination buffer.
- **src**: Pointer to null-terminated string to copy.

Returns:

Pointer to destination buffer or NULL if copy failed.

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

This function concatenates the given number of characters from the given string onto the string buffer.

It is similar to the C `strncat` function except more secure because it checks for buffer overrun.

Parameters:

- **dest**: Pointer to destination buffer to receive string.
5.2.2.8 **EXTERNRT char* rtxStrncpy (char * dest, size_t bufsiz, const char * src, size_t nchars)**

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

**Parameters:**
- **dest** Pointer to destination buffer to receive string.
- **bufsiz** Size of the destination buffer.
- **src** Pointer to null-terminated string to copy.
- **nchars** Number of characters to copy.

**Returns:**
- Pointer to destination buffer or NULL if copy failed.

5.2.2.9 **EXTERNRT int rtxUInt64ToCharStr (OSUINT64 value, char * dest, size_t bufsiz, char padchar)**

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

**Parameters:**
- **value** Integer to convert.
- **dest** Pointer to destination buffer to receive string.
- **bufsiz** Size of the destination buffer.
- **padchar** Left pad char, set to zero for no padding.

**Returns:**
- Number of characters or negative status value if fail.

5.2.2.10 **EXTERNRT int rtxUIntToCharStr (OSUINT32 value, char * dest, size_t bufsiz, char padchar)**

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

**Parameters:**
- **value** Integer to convert.
**dest** Pointer to destination buffer to receive string.

**bufsz** Size of the destination buffer.

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

**Returns:**

Number of characters or negative status value if fail.
5.3 Context Management Functions

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

Classes

- struct OSRTErrLocn
  
  Run-time error location structure.

- struct OSRTErrInfo
  
  Run-time error information structure.

- struct OSRTBuffer
  
  Run-time message buffer structure.

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

- struct OSCTXT
  
  Run-time context structure.

Defines

- #define rtxCtxtGetMsgPtr(pctxt) (pctxt) → buffer.data
  
  This macro returns the start address of an encoded message.

- #define rtxCtxtGetMsgLen(pctxt) (pctxt) → buffer.byteIndex
  
  This macro returns the length of an encoded message.

- #define rtxCtxtTestFlag(pctxt, mask) ((pctxt → flags & mask) != 0)
  
  This macro tests if the given bit flag is set in the context.

- #define rtxByteAlign(pctxt)
  
  This macro will byte-align the context buffer.

Functions

- EXTERNRT int rtxInitContextUsingKey (OSCTXT *pctxt, const OSOCTET *key, size_t keylen)
  
  This function initializes a context using a run-time key.

- EXTERNRT int rtxInitContext (OSCTXT *pctxt)
  
  This function initializes an OSCTXT block.

- EXTERNRT int rtxInitContextExt (OSCTXT *pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
  
  This function initializes an OSCTXT block.
• EXTERNRT int rtxInitThreadContext (OSCTXT *pctxt, const OSCTXT *pSrcCtxt)
  
  This function initializes a context for use in a thread.

• EXTERNRT int rtxInitContextBuffer (OSCTXT *pctxt, OSOCTET *bufaddr, size_t bufsiz)
  
  This function assigns a message buffer to a context block.

• EXTERNRT int rtxCtxSetBufPtr (OSCTXT *pctxt, OSOCTET *bufaddr, size_t bufsiz)
  
  This function is used to set the internal buffer pointer for in-memory encoding or decoding.

• EXTERNRT size_t rtxCtxGetIOByteCount (OSCTXT *pctxt)
  
  This function returns the count of bytes either written to a stream or memory buffer.

• EXTERNRT int rtxCheckContext (OSCTXT *pctxt)
  
  This function verifies that the given context structure is initialized and ready for use.

• EXTERNRT void rtxFreeContext (OSCTXT *pctxt)
  
  This function frees all dynamic memory associated with a context.

• EXTERNRT void rtxCtxSetFlag (OSCTXT *pctxt, OSUINT32 mask)
  
  This function is used to set a processing flag within the context structure.

• EXTERNRT void rtxCtxClearFlag (OSCTXT *pctxt, OSUINT32 mask)
  
  This function is used to clear a processing flag within the context structure.

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

• EXTERNRT const OSUTF8CHAR * rtxCtxPopElemName (OSCTXT *pctxt)
  
  This function pops the last element name from the context stack.

5.3.1 Detailed Description

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

These variables hold all of the working data used during the process of encoding or decoding a message. The context provides thread safe operation by isolating what would otherwise be global variables within this structure. The context variable is passed from function to function as a message is encoded or decoded and maintains state information on the encoding or decoding process.

5.3.2 Define Documentation

5.3.2.1 #define rtxCtxGetMsgLen(pctxt) (pctxt) → buffer.byteIndex

This macro returns the length of an encoded message.

It will only work for in-memory encoding, not for encode to stream.

Note that this macro will not work with ASN.1 BER in-memory encoding. In this case, the BER-specific version of the function must be used.
Parameters:

\textit{pctxt} Pointer to a context structure.

Definition at line 377 of file rtxContext.h.

5.3.2.2 \texttt{#define rtxCtxtGetMsgPtr(pctxt) (pctxt)} \rightarrow \texttt{buffer.data}

This macro returns the start address of an encoded message.
If a static buffer was used, this is simply the start address of the buffer. If dynamic encoding was done, this will return the start address of the dynamic buffer allocated by the encoder.
Note that this macro will not work with ASN.1 BER in-memory encoding. In this case, the BER-specific version of the function must be used.

Parameters:

\textit{pctxt} Pointer to a context structure.

Definition at line 366 of file rtxContext.h.

5.3.2.3 \texttt{#define rtxCtxtTestFlag(pctxt, mask) ((pctxt} \rightarrow \texttt{flags & mask) != 0)}

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

Parameters:

\textit{pctxt} - A pointer to a context structure.
\textit{mask} - Bit flag to be tested

Definition at line 442 of file rtxContext.h.

5.3.3 Function Documentation

5.3.3.1 \texttt{EXTERNRT int rtxCheckContext (OSCTXarah * pctxt)}

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

Parameters:

\textit{pctxt} Pointer to a context structure.

Returns:

Completion status of operation:
\begin{itemize}
  \item 0 = success,
  \item RTERR\_NOTINIT status code if not initialized
\end{itemize}
5.3.3.2  EXTERNRT void rtxCtxtClearFlag (OSCTXT * pctxt, OSUINT32 mask)

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

Parameters:

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

5.3.3.3  EXTERNRT size_t rtxCtxtGetIOByteCount (OSCTXT * pctxt)

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

Parameters:

  - pctxt - Pointer to a context structure.

Returns:

  - I/O byte count.

5.3.3.4  EXTERNRT const OSUTF8CHAR* rtxCtxtPopElemName (OSCTXT * pctxt)

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

Parameters:

  - pctxt - Pointer to a context structure.

Returns:

  - Element name popped from stack or NULL if stack is empty.

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

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

Parameters:

  - pctxt - Pointer to a context structure.
  - elemName - Name of element to be pushed on stack. Note that a copy of the name is not made, the pointer to the name that is passed is stored.

Returns:

  - Completion status of operation:
    - 0 = success,
    - RTERR_NOMEM if mem alloc for name fails.
5.3.3.6 EXTERNRT int rtxCtxtSetBufPtr (OSCTXT * pctxt, OSOCTET * bufaddr, size_t bufsiz)

This function is used to set the internal buffer pointer for in-memory encoding or decoding. It must be called after the context variable is initialized before any other compiler generated or run-time library encode function.

Parameters:

- `pctxt` Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
- `bufaddr` A pointer to a memory buffer to use to encode a message or that holds a message to be decoded. The buffer should be declared as an array of unsigned characters (OCTETs). This parameter can be set to NULL to specify dynamic encoding (i.e., the encode functions will dynamically allocate a buffer to hold the encoded message).
- `bufsiz` The length of the memory buffer in bytes. Should be set to zero if NULL was specified for bufaddr (i.e. dynamic encoding was selected).

5.3.3.7 EXTERNRT void rtxCtxtSetFlag (OSCTXT * pctxt, OSUINT32 mask)

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

Parameters:

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

5.3.3.8 EXTERNRT void rtxFreeContext (OSCTXT * pctxt)

This function frees all dynamic memory associated with a context. This includes all memory allocated using the rtxMem functions using the given context parameter.

Parameters:

- `pctxt` Pointer to a context structure.

5.3.3.9 EXTERNRT int rtxInitContext (OSCTXT * pctxt)

This function initializes an OSCTXT block. It sets all key working parameters to their correct initial state values. It is required that this function be invoked before using a context variable.

Parameters:

- `pctxt` Pointer to the context structure variable to be initialized.

Returns:

Completion status of operation:
- 0 = success,
- negative return value is error.
5.3.3.10 EXTERNRT int rtxInitContextBuffer (OSCTXT * pctxt, OSOCTET * bufaddr, size_t bufsiz)

This function assigns a message buffer to a context block. The block should have been previously initialized by rtxInitContext.

Parameters:

- \texttt{pctxt} The pointer to the context structure variable to be initialized.
- \texttt{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.
- \texttt{bufsiz} The size of the memory buffer. For encoding, this argument may be set to zero to indicate a dynamic memory buffer should be used.

Returns:

Completion status of operation:

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

5.3.3.11 EXTERNRT int rtxInitContextExt (OSCTXT * pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

This function initializes an OSCTX block. It sets all key working parameters to their correct initial state values. It is required that this function be invoked before using a context variable.

Parameters:

- \texttt{pctxt} Pointer to the context structure variable to be initialized.
- \texttt{malloc_func} Pointer to the memory allocation function.
- \texttt{realloc_func} Pointer to the memory reallocation function.
- \texttt{free_func} Pointer to the memory deallocation function.

Returns:

Completion status of operation:

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

5.3.3.12 EXTERNRT int rtxInitContextUsingKey (OSCTXT * pctxt, const OSOCTET * key, size_t 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 rtxInitContext in the rtkey.h file that will invoke this function with the generated run-time key.

Parameters:

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

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

5.3.3.13  EXTERNRT int rtxInitThreadContext (OSCTXT ∗ pctxt, const OSCTXT ∗ pSrcCntx)

This function initializes a context for use in a thread.
It is the same as rtxInitContext except that it copies the pointer to constant data from the given source context into the newly initialized thread context. It is assumed that the source context has been initialized and the custom generated global initialization function has been called. The main purpose of this function is to prevent multiple copies of global static data from being created within different threads.

Parameters:
pctxt  Pointer to the context structure variable to be initialized.
pSrcCntx  Pointer to source context which has been fully initialized including a pointer to global constant data initialized via a call to a generated 'Init_<project>_Global' function.

Returns:
Completion status of operation:
• 0 = success,
• negative return value is error.
5.4 Date/time conversion functions

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

Functions

- EXTERN RT 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).

- EXTERN RT 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]).

- EXTERN RT 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]).

- EXTERN RT int rtxGYearToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  
  This function formats a gregorian year value to a string (CCYY).

- EXTERN RT int rtxGYearMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  
  This function formats a gregorian year and month value to a string (CCYY-MM).

- EXTERN RT int rtxGMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  
  This function formats a gregorian month value to a string (MM).

- EXTERN RT int rtxGMonthDayToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  
  This function formats a gregorian month and day value to a string (MM-DD).

- EXTERN RT int rtxGDayToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  
  This function formats a gregorian day value to a string (DD).

- EXTERN RT int rtxGetCurrDateTime (OSNumDateTime *pvalue)
  
  This function reads the system date and time and stores the value in the given OSNumDateTime structure variable.

- EXTERN RT int rtxCmpDate (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
  
  This function compares the date part of two OSNumDateTime structures and returns the result of the comparison.

- EXTERN RT 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.

- EXTERN RT int rtxCmpTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
  
  This function compares the time part of two OSNumDateTime structures and returns the result of the comparison.
• EXTERNRT int rtxCmpTime2 (const OSNumDateTime *pvalue, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
  This function compares the time part of OSNumDateTime structure and time components, specified as parameters.

• EXTERNRT int rtxCmpDateTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
  This function compares two OSNumDateTime structures and returns the result of the comparison.

• EXTERNRT int rtxCmpDateTime2 (const OSNumDateTime *pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
  This function compares the OSNumDateTime structure and dateTime components, specified as parameters.

• EXTERNRT int rtxParseDateString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a date value from a supplied string and sets the given OSNumDateTime argument to the decoded date value.

• EXTERNRT int rtxParseTimeString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a time value from a supplied string and sets the given OSNumDateTime structure to the decoded time value.

• EXTERNRT int rtxParseDateTimeString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a datetime value from a supplied string and sets the given OSNumDateTime to the decoded date and time value.

• EXTERNRT int rtxParseGYearString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a gregorian year value from a supplied string and sets the year in the given OSNumDateTime to the decoded value.

• EXTERNRT int rtxParseGYearMonthString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a gregorian year and month value from a supplied string and sets the year and month fields in the given OSNumDateTime to the decoded values.

• EXTERNRT int rtxParseGMonthString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a gregorian month value from a supplied string and sets the month field in the given OSNumDateTime to the decoded value.

• EXTERNRT int rtxParseGMonthDayString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a gregorian month and day value from a supplied string and sets the month and day fields in the given OSNumDateTime to the decoded values.

• EXTERNRT int rtxParseGDayString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
  This function decodes a gregorian day value from a supplied string and sets the day field in the given OSNumDateTime to the decoded value.
• EXTERNRT int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR *buf, OSUINT32 bufsize)
  This function converts milliseconds to a duration string with format "PnYnMnDTnHnMnS".

• EXTERNRT int rtxDurationToMSecs (OSUTF8CHAR *buf, OSUINT32 bufsize, OSINT32 *msecs)
  This function converts a duration string to milliseconds.

• EXTERNRT int rtxSetDateTime (OSNumDateTime *pvalue, struct tm *timeStruct)
  This function converts a structure of type ‘struct tm’ to an OSNumDateTime structure.

• EXTERNRT int rtxSetLocalDateTime (OSNumDateTime *pvalue, time_t timeMs)
  This function converts a local date and time value to an OSNumDateTime structure.

• EXTERNRT int rtxSetUtcDateTime (OSNumDateTime *pvalue, time_t timeMs)
  This function converts a UTC date and time value to an OSNumDateTime structure.

• EXTERNRT int rtxGetDateTime (const OSNumDateTime *pvalue, time_t *timeMs)
  This function converts an OSNumDateTime structure to a calendar time encoded as a value of type time_t.

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

• EXTERNRT OSBOOL rtxTimeIsValid (const OSNumDateTime *pvalue)
  This function verifies that time members (hour, minute, second, timezone) of the OSNumDateTime structure contains valid values.

• EXTERNRT OSBOOL rtxDateTimeIsValid (const OSNumDateTime *pvalue)
  This function verifies that all members of the OSNumDateTime structure contains valid values.

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

5.4.2 Function Documentation

5.4.2.1 EXTERNRT int rtxCmpDate (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
This function compares the date part of two OSNumDateTime structures and returns the result of the comparison.

Parameters:
  pvalue1 Pointer to OSNumDateTime structure.
  pvalue2 Pointer to OSNumDateTime structure.

Returns:
  Completion status of operation:
  • 0 Dates are same
  • +1 First Date/Time is greater than second.
  • -1 First Date/Time is less than second.
5.4.2.2 EXTERNRT int rtxCmpDate2 (const OSNumDateTime * pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSBOOL tzflag, OSINT32 tzo)

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

Parameters:

pvalue Pointer to OSNumDateTime structure.
year Year (-inf..inf)
mon Month (1..12)
day Day (1..31)
tzflag TRUE, if time zone offset is set (see tzo parameter).
tzo Time zone offset (-840..840).

Returns:

Completion status of operation:
• 0 Dates are same
• +1 First Date/Time is greater than second.
• -1 First Date/Time is less than second.

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

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

Parameters:

pvalue1 Pointer to OSNumDateTime structure.
pvalue2 Pointer to OSNumDateTime structure.

Returns:

Completion status of operation:
• 0 Dates are same
• +1 First Date/Time is greater than second.
• -1 First Date/Time is less than second.

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

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

Parameters:

pvalue Pointer to OSNumDateTime structure.
year Year (-inf..inf)
mon Month (1..12)
day  Day (1..31)
hour  Hour (0..23)
min  Minutes (0..59)
sec  Seconds (0.0..59.(9))
tzflag  TRUE, if time zone offset is set (see tzo parameter).
tzo  Time zone offset (-840..840).

Returns:
Completion status of operation:
• 0 Dates are same
• +1 First Date/Time is greater than second.
• -1 First Date/Time is less than second.

5.4.2.5  EXTERNRT int rtxCmpTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
This function compares the time part of two OSNumDateTime structures and returns the result of the comparison.

Parameters:

 pvalue1  Pointer to OSNumDateTime structure.
pvalue2  Pointer to OSNumDateTime structure.

Returns:
Completion status of operation:
• 0 Times are same
• +1 First Date/Time is greater than second.
• -1 First Date/Time is less than second.

5.4.2.6  EXTERNRT int rtxCmpTime2 (const OSNumDateTime *pvalue, OSUINT8 hour, OSUINT8 min,
OSREAL sec, OSBOOL tzo, OSINT32 tzo)
This function compares the time part of OSNumDateTime structure and time components, specified as parameters.

Parameters:

 pvalue  Pointer to OSNumDateTime structure.
hour  Hour (0..23)
min  Minutes (0..59)
sec  Seconds (0.0..59.(9))
tzflag  TRUE, if time zone offset is set (see tzo parameter).
tzo  Time zone offset (-840..840).

Returns:
Completion status of operation:
• 0 Times are same
• +1 First Date/Time is greater than second.
• -1 First Date/Time is less than second.
5.4.2.7 EXTERNRT OSBOOL rtxDateIsValid (const OSNumDateTime * pvalue)

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

Parameters:

   pvalue Pointer to OSNumDateTime structure to be checked.

Returns:

   Boolean result: true means data is valid.

5.4.2.8 EXTERNRT OSBOOL rtxDateTimeIsValid (const OSNumDateTime * pvalue)

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

Parameters:

   pvalue Pointer to OSNumDateTime structure to be checked.

Returns:

   Boolean result: true means data is valid.

5.4.2.9 EXTERNRT int rtxDateTimeToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)

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

Parameters:

   pvalue Pointer to OSNumDateTime structure containing date/time components to be formatted.
   buffer Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string.
   bufsize Size of the buffer to receive the formatted date.

Returns:

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

5.4.2.10 EXTERNRT int rtxDateToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)

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

Parameters:

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

buFSIZE Size of the buffer to receive the formatted date.

Returns:
Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error

5.4.2.11 EXTERNRT int rtxDurationToMSecs (OSUTF8CHAR * buf, OSUINT32 bufsize, OSINT32 * msecs)
This function converts a duration string to milliseconds.
In the case of a string prepended with a minus sign (-) the duration in milliseconds will have negative value.

Parameters:
buf Pointer to OSUTF8CHAR array.
bufsize OSINT32 indicates the bufsize to be read.
msecs OSINT32 updated after calculation.

Returns:
Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_TOOBIG). Return value is taken from rtxErrCodes.h header file

5.4.2.12 EXTERNRT int rtxGDayToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)
This function formats a gregorian day value to a string (DD).

Parameters:
pvalue Pointer to OSNumDateTime structure containing day value to be formatted.
buffer Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 3 characters long).
bufsize Size of the buffer to receive the formatted date.

Returns:
Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error
5.4.2.13 EXTERNRT int rtxGetCurrDateTime (OSNumDateTime * pvalue)

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

Parameters:

pvalue Pointer to OSNumDateTime structure.

Returns:

Completion status of operation:
- 0 in case success
- negative in case failure

5.4.2.14 EXTERNRT int rtxGetDateTime (const OSNumDateTime * pvalue, time_t * timeMs)

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

Parameters:

pvalue The pointed OSNumDateTime structure variable to be converted.
timeMs A pointer to time_t value to be set.

Returns:

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

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

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

Parameters:

pvalue Pointer to OSNumDateTime structure containing month and day value to be formatted.
buffer Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 6 characters long).
bufsize Size of the buffer to receive the formatted date.

Returns:

Completion status of operation:
- 0(RT_OK) = success,
- negative return value is error
5.4.2.16 EXTERNRT int rtxGMonthToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)

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

Parameters:

pvalue Pointer to OSNumDateTime structure containing month value to be formatted.
buffer Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 3 characters long).
bufsize Size of the buffer to receive the formatted date.

Returns:

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

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

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

Parameters:

pvalue Pointer to OSNumDateTime structure containing year and month value to be formatted.
buffer Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 8 characters long).
bufsize Size of the buffer to receive the formatted date.

Returns:

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

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

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

Parameters:

pvalue Pointer to OSNumDateTime structure containing year value to be formatted.
buffer Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 5 characters long).
bufsize Size of the buffer to receive the formatted date.

Returns:

Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error
5.4.2.19 EXTERNRT int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR * buf, OSUINT32 bufsize)

This function converts milliseconds to a duration string with format "PnYnMnDTnHnMnS". 
In case of negative duration a minus sign is prepended to the output string

Parameters:

- *msecs*  Number of milliseconds.
- *buf*   Output buffer to receive formatted duration.
- *bufsize*   Output buffer size.

Returns:

Completion status of operation: 0 successful are same -1 unsuccessul

5.4.2.20 EXTERNRT int rtxParseDateString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a date value from a supplied string and sets the given OSNumDateTime argument to the decoded date value.

Parameters:

- *inpdata*  Date string to be parsed/decoded as OSNumDateTime.
  - The format of date is CCYY-MM-DD
  - The value of CCYY is from 0000-9999
  - The value of MM is 01 - 12
  - The value of DD is 01 - XX (where XX is the Days in MM month in CCYY year)
- *inpdatalen*  For decoding, consider inpdata string up to this length.
- *pvalue*  The OSNumDateTime structure variable will be set to the decoded date value.
  - Only year, month,day value will be set.
  - The value of pvalue->year is in range 0 to 9999
  - The value of pvalue->mon is in range 1 to 12
  - The value of pvalue->day is in range 1 to XX

Returns:

Completion status of operation:

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

5.4.2.21 EXTERNRT int rtxParseDateTimeString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a datetime value from a supplied string and sets the given OSNumDateTime to the decoded date and time value.

Parameters:

- *inpdata*  Input date/time string to be parsed.

30
**inpdatalen** For decoding, consider the inpdata string up to this length.

**pvalue** The pointed OSNumDateTime structure variable will be set to the decoded date and time value.

Returns:

Completion status of operation:

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

5.4.2.22  EXTERNRT int rtxParseGDayString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a gregorian day value from a supplied string and sets the day field in the given OSNumDateTime to the decoded value.

Parameters:

- **inpdata** Input string to be parsed.
- **inpdatalen** For decoding, consider the inpdata string up to this length.
- **pvalue** The day field in the given OSNumDateTime variable will be set to the decoded value.

Returns:

Completion status of operation:

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

5.4.2.23  EXTERNRT int rtxParseGMonthDayString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a gregorian month and day value from a supplied string and sets the month and day fields in the given OSNumDateTime to the decoded values.

Parameters:

- **inpdata** Input string to be parsed.
- **inpdatalen** For decoding, consider the inpdata string up to this length.
- **pvalue** The month and day fields in the given OSNumDateTime variable will be set to the decoded values.

Returns:

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file
5.4.2.24 EXTERNRT int rtxParseGMonthString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a gregorian month value from a supplied string and sets the month field in the given OSNumDateTime to the decoded value.

Parameters:

inpdata  Input string to be parsed.
inpdatalen  For decoding, consider the inpdata string up to this length.
pvalue  The month field in the given OSNumDateTime variable will be set to the decoded value.

Returns:

Completion status of operation:

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

5.4.2.25 EXTERNRT int rtxParseGYearMonthString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a gregorian year and month value from a supplied string and sets the year and month fields in the given OSNumDateTime to the decoded values.

Parameters:

inpdata  Input string to be parsed.
inpdatalen  For decoding, consider the inpdata string up to this length.
pvalue  The year and month fields in the given OSNumDateTime variable will be set to the decoded value.

Returns:

Completion status of operation:

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

5.4.2.26 EXTERNRT int rtxParseGYearString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a gregorian year value from a supplied string and sets the year in the given OSNumDateTime to the decoded value.

Parameters:

inpdata  Input string to be parsed.
inpdatalen  For decoding, consider the inpdata string up to this length.
pvalue  The year field in the given OSNumDateTime structure variable will be set to the decoded value.
Returns:

Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

5.4.2.27 EXTERNRT int rtxParseTimeString (const OSUTF8CHAR * inpdata, size_t inpdatalen, OSNumDateTime * pvalue)

This function decodes a time value from a supplied string and sets the given OSNumDateTime structure to the decoded time value.

Parameters:

inpdata The inpdata is a time string to be parsed/decoded as OSNumDateTime.
• The format of date is hh:mm:ss.ss (1) or hh:mm:ssZ (2) or hh:mm:ss+HH:MM (3) or hh:mm:ss.ss-HH:MM (4)
• The value of hh is from 00-23
• The value of mm is 00 - 59
• The value of ss.ss is 00.00 - 59.99
• The value of HH:MM is 00.00 - 24.00

inpdatalen For decoding, consider the inpdata string up to this length.

pvalue The OSNumDateTime structure variable will be set to the decoded time value.
• Only hour, min, sec value will be set.
• The value of pvalue->hour is in range 0 to 23
• The value of pvalue->mon is in range 0 to 59
• The value of pvalue->day is in range 0 to 59.99
• The value of pvalue->tz_flag is FALSE for format(1) otherwise TRUE
• The value of pvalue->tzo is 0 for format(2) otherwise Calculation of pvalue->tzo for format (3),(4) is HH*60+MM
• The value of pvalue->tzo is -840 <= tzo <= 840 for format(3),(4) otherwise

Returns:

Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

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

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

Parameters:

pvalue The pointed OSNumDateTime structure variable will be set to time value.

timeStruct A pointer to tm structure to be converted.
Returns:
Completion status of operation:
• 0(RT_OK) = success,
• negative return value is error.

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

Parameters:

pvalue The pointed OSNumDateTime structure variable will be set to time value.
timeMs A calendar time encoded as a value of type time_t.

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

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

Parameters:

pvalue The pointed OSNumDateTime structure variable will be set to time value.
timeMs A calendar time encoded as a value of type time_t. The time is represented as seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC).

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

5.4.2.31 EXTERNRT OSBOOL rtxTimeIsValid (const OSNumDateTime ∗ pvalue)
This function verifies that time members (hour, minute, second, timezone) of the OSNumDateTime structure contains valid values.

Parameters:

pvalue Pointer to OSNumDateTime structure to be checked.

Returns:
Boolean result: true means data is valid.
5.4.2.32  EXTERNRT int rtxTimeToString (const OSNumDateTime * pvalue, OSUTF8CHAR * buffer, size_t bufsize)

This function formats a numeric time value consisting of individual time components (hour, minute, second, fraction-of-second, time zone) into XML schema standard format (HH:MM:SS[.frac][TZ]).

**Parameters:**

- **pvalue**  Pointer to OSNumDateTime structure containing time components to be formatted.
- **buffer**  Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string.
- **bufsize**  Size of the buffer to receive the formatted date.

**Returns:**

Completion status of operation:
- 0(RT_OK) = success,
- negative return value is error
5.5 Decimal number utility functions

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

5.5.1 Detailed Description

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

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

Classes

- struct OSRTPrintStream
  
  Structure to hold information about a global PrintStream.

Typedefs

- typedef void(* rtxPrintCallback)(void*pPrntStrmInfo, const char*fmtspec, va_list arglist)
  
  Callback function definition for print stream.

Functions

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

- EXTERNRT OSBOOL rtxSetDiag (OSCTXTPctx, OSBOOL value)
  
  This function is used to turn diagnostic tracing on or off at run-time on a per-context basis.

- EXTERNRT OSBOOL rtxSetGlobalDiag (OSBOOL value)
  
  This function is used to turn diagnostic tracing on or off at run-time on a global basis.

- EXTERNRT void rtxDiagPrint (OSCTXTPctx, const char*fmtspec,...)
  
  This function is used to print a diagnostics message to stdout.

- EXTERNRT void rtxDiagStream (OSCTXTPctx, const char*fmtspec,...)
  
  This function conditionally outputs diagnostic trace messages to an output stream defined within the context.

- EXTERNRT void rtxDiagHexDump (OSCTXTPctx, const OSOCTET*data, OSUINT32 numocts)
  
  This function is used to print a diagnostics hex dump of a section of memory.

- EXTERNRT void rtxDiagStreamHexDump (OSCTXTPctx, const OSOCTET*data, OSUINT32 numocts)
  
  This function is used to print a diagnostics hex dump of a section of memory to a print stream.

- EXTERNRT void rtxDiagPrintChars (OSCTXTPctx, const char*data, OSUINT32 nchars)
  
  This function is used to print a given number of characters to standard output.

- EXTERNRT void rtxDiagStreamPrintChars (OSCTXTPctx, const char*data, OSUINT32 nchars)
  
  This function is used to print a given number of characters to a print stream.

- EXTERNRT void rtxDiagSetTraceLevel (OSCTXTPctx, OSRTDiagTraceLevel level)
  
  This function is used to set the maximum trace level for diagnostic trace messages.

- EXTERNRT int rtxSetPrintStream (OSCTXTPctx, rtxPrintCallback myCallback, void*pStrmInfo)
This function is for setting the callback function for a PrintStream.

- **EXTERNRT int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void *pStrmInfo)**
  
  This function is for setting the callback function for a PrintStream.

- **EXTERNRT int rtxPrintToStream (OSCTXT *pctxt, const char *fmtspec,...)**
  
  Print-to-stream function which in turn calls the user registered callback function of the context for printing.

- **EXTERNRT int rtxDiagToStream (OSCTXT *pctxt, const char *fmtspec, va_list arglist)**
  
  Diagnostics print-to-stream function.

- **EXTERNRT int rtxPrintStreamRelease (OSCTXT *pctxt)**
  
  This function releases the memory held by PrintStream in the context.

**Variables**

- **OSRTPrintStream g_PrintStream**

  Global PrintStream.

### 5.6.1 Detailed Description

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

Calls to these macros and functions are added when the `-trace` command-line argument is used. The diagnostic message can be turned on and off at both C compile and run-time. To C compile the diagnostics in, the `_TRACE` macro must be defined (`-D_TRACE`). To turn the diagnostics on at run-time, the `rtxSetDiag` function must be invoked with the `value` argument set to TRUE.

### 5.6.2 Function Documentation

#### 5.6.2.1 EXTERNRT OSBOOL rtxDiagEnabled (OSCTXT *pctxt)

This function is used to determine if diagnostic tracing is currently enabled for the specified context.

It returns true if enabled, false otherwise.

**Parameters:**

- **pctxt** Pointer to context structure.

**Returns:**

- Boolean result.

#### 5.6.2.2 EXTERNRT void rtxDiagHexDump (OSCTXT *pctxt, const OSOCTET *data, OSUINT32 numocts)

This function is used to print a diagnostics hex dump of a section of memory.
Parameters:

- \texttt{pctxt} Pointer to context structure.
- \texttt{data} Start address of memory to dump.
- \texttt{numocts} Number of bytes to dump.

5.6.2.3 \texttt{EXTERNRT} void \texttt{rtxDiagPrint} (OSCTXT * \texttt{pctxt}, const char * \texttt{fmtspec}, ...) 

This function is used to print a diagnostics message to \texttt{stdout}.

Its parameter specification is similar to that of the C runtime \texttt{printf} method. The \texttt{fmtspec} argument may contain \% style modifiers into which variable arguments are substituted. This function is called in the generated code via the RTDIAG macros to allow diagnostic trace call to easily be compiled out of the object code.

Parameters:

- \texttt{pctxt} Pointer to context structure.
- \texttt{fmtspec} A printf-like format specification string describing the message to be printed (for example, "string s, ivalue d "). A special character sequence (~L) may be used at the beginning of the string to select a trace level (L would be replaced with E for Error, W for warning, I for info, or D for debug).
  
  ... Variable list of parameters to be substituted into the format string.

5.6.2.4 \texttt{EXTERNRT} void \texttt{rtxDiagPrintChars} (OSCTXT * \texttt{pctxt}, const char * \texttt{data}, OSUINT32 \texttt{nchars}) 

This function is used to print a given number of characters to standard output.

The buffer containing the characters does not need to be null-terminated.

Parameters:

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

5.6.2.5 \texttt{EXTERNRT} void \texttt{rtxDiagSetTraceLevel} (OSCTXT * \texttt{pctxt}, OSRTDiagTraceLevel \texttt{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 \texttt{rtxDiagPrint} function documentation is used to set a message level to be compared with the trace level.

Parameters:

- \texttt{pctxt} Pointer to context structure.
- \texttt{level} Trace level to be set.
5.6.2.6 EXTERNRT void rtxDiagStream (OSCTXT *pctxt, const char *fmtspec, ...) 

This function conditionally outputs diagnostic trace messages to an output stream defined within the context.

A code generator embeds calls to this function into the generated source code when the -trace option is specified on the command line (note: it may embed the macro version of these calls - RTDIAGSTREAMx - so that these calls can be compiled out of the final code.

See also:

rtxDiagPrint

5.6.2.7 EXTERNRT void rtxDiagStreamHexDump (OSCTXT *pctxt, const OSOCTET *data, OSUINT32 numocts) 

This function is used to print a diagnostics hex dump of a section of memory to a print stream.

Parameters:

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

5.6.2.8 EXTERNRT void rtxDiagStreamPrintChars (OSCTXT *pctxt, const char *data, OSUINT32 nchars) 

This function is used to print a given number of characters to a print stream.

The buffer containing the characters does not need to be null-terminated.

Parameters:

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

5.6.2.9 EXTERNRT int rtxDiagToStream (OSCTXT *pctxt, const char *fmtspec, va_list arglist) 

Diagnostics print-to-stream function.

This is the same as the rtxPrintToStream function except that it checks if diagnostic tracing is enabled before invoking the callback function.

Parameters:

- pctxt Pointer to context to be used.
- fmtspec A printf-like format specification string describing the message to be printed (for example, "string s, ivalue d").
- arglist A variable list of arguments passed as va_list

Returns:

Completion status, 0 on success, negative value on failure
5.6.2.10 EXTERNRT int rtxPrintStreamRelease (OSCTXT * pctxt)

This function releases the memory held by PrintStream in the context.

Parameters:

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

Returns:

  Completion status, 0 on success, negative value on failure

5.6.2.11 EXTERNRT int rtxPrintToStream (OSCTXT * pctxt, const char * fmtspec, ...)

Print-to-stream function which in turn calls the user registered callback function of the context for printing. If no callback function is registered it prints to standard output by default.

Parameters:

  pctxt  Pointer to context to be used.

  fmtspec  A printf-like format specification string describing the message to be printed (for example, "string s, ivalue d ").

  ...  A variable list of arguments.

Returns:

  Completion status, 0 on success, negative value on failure

5.6.2.12 EXTERNRT OSBOOL rtxSetDiag (OSCTXT * pctxt, OSBOOL value)

This function is used to turn diagnostic tracing on or off at run-time on a per-context basis. Code generated using ASN1C or XBinder or a similar code generator must use the -trace command line option to enable diagnostic messages. The generated code must then be C compiled with _TRACE defined for the code to be present.

Parameters:

  pctxt  Pointer to context structure.

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

Returns:

  Prior setting of the diagnostic trace switch in the context.

5.6.2.13 EXTERNRT OSBOOL rtxSetGlobalDiag (OSBOOL value)

This function is used to turn diagnostic tracing on or off at run-time on a global basis. It is similar to rtxSetDiag except tracing is enabled within all contexts.
Parameters:

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

Returns:

Prior setting of the diagnostic trace switch in the context.

5.6.2.14 EXTERNRT int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void * pStrmInfo)

This function is for setting the callback function for a PrintStream.
This version of the function sets a callback at the global level.

Parameters:

myCallback  Pointer to a callback print function.

pStrmInfo  Pointer to user defined PrintInfo structure where users can store information required by the callback function across calls. Ex. An open File handle for callbak function which directs stream to a file.

Returns:

Completion status, 0 on success, negative value on failure

5.6.2.15 EXTERNRT int rtxSetPrintStream (OSCTXT * pctxt, rtxPrintCallback myCallback, void * pStrmInfo)

This function is for setting the callback function for a PrintStream.
Once a callback function is set, then all print and debug output ia sent to the defined callback function.

Parameters:

pctxt  Pointer to a context in which callback print function will be set

myCallback  Pointer to a callback print function.

pStrmInfo  Pointer to user defined PrintInfo structure where users can store information required by the callback function across calls. Ex. An open File handle for callbak function which directs stream to a file.

Returns:

Completion status, 0 on success, negative value on failure
5.7 Doubly-Linked List Utility Functions

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

Classes

• struct OSRTDListNode
  
  *This structure is used to hold a single data item within the list.*

• struct OSRTDList
  
  *This is the main list structure.*

Functions

• EXTERNRT void rtxDListInit (OSRTDList *pList)
  
  *This function initializes a doubly linked list structure.*

• EXTERNRT OSRTDListNode * rtxDListAppend (struct OSCTXT *pctxt, OSRTDList *pList, void *pData)
  
  *This function appends an item to the linked list structure.*

• EXTERNRT OSRTDListNode * rtxDListInsert (struct OSCTXT *pctxt, OSRTDList *pList, OSUINT32 index, void *pData)
  
  *This function inserts an item into the linked list structure.*

• EXTERNRT OSRTDListNode * rtxDListInsertBefore (struct OSCTXT *pctxt, OSRTDList *pList, OSRTDListNode *node, void *pData)
  
  *This function inserts an item into the linked list structure before the specified element.*

• EXTERNRT OSRTDListNode * rtxDListInsertAfter (struct OSCTXT *pctxt, OSRTDList *pList, OSRTDListNode *node, void *pData)
  
  *This function inserts an item into the linked list structure after the specified element.*

• EXTERNRT OSRTDListNode * rtxDListFindByIndex (const OSRTDList *pList, OSUINT32 index)
  
  *This function will return the node pointer of the indexed entry in the list.*

• EXTERNRT OSRTDListNode * rtxDListFindByData (const OSRTDList *pList, void *data)
  
  *This function will return the node pointer of the given data item within the list or NULL if the item is not found.*

• EXTERNRT int rtxDListFindIndexByData (const OSRTDList *pList, void *data)
  
  *This function will return the index of the given data item within the list or -1 if the item is not found.*

• EXTERNRT void rtxDListFreeNode (struct OSCTXT *pctxt, OSRTDList *pList, OSRTDListNode *node)
  
  *This function will remove the given node from the list and free memory.*

• EXTERNRT void rtxDListRemove (OSRTDList *pList, OSRTDListNode *node)
  
  *This function will remove the given node from the list.*

• EXTERNRT void rtxDListFreeNodes (struct OSCTXT *pctxt, OSRTDList *pList)
This function will free all of the dynamic memory used to hold the list node pointers.

- EXTERNRT void rtxDListFreeAll (struct OSCTXT *pctxt, OSRTDList *pList)
  This function will free all of the dynamic memory used to hold the list node pointers and the data items.

- EXTERNRT int rtxDListToArray (struct OSCTXT *pctxt, OSRTDList *pList, void **ppArray, OSUINT32 *pElemCount, size_t elemSize)
  This function converts a doubly linked list to an array.

- EXTERNRT int rtxDListAppendArray (struct OSCTXT *pctxt, OSRTDList *pList, void *pArray, OSUINT32 numElements, size_t elemSize)
  This function appends pointers to items in the given array to a doubly linked list structure.

- EXTERNRT int rtxDListAppendArrayCopy (struct OSCTXT *pctxt, OSRTDList *pList, const void *pArray, OSUINT32 numElements, size_t elemSize)
  This function appends a copy of each item in the given array to a doubly linked list structure.

- EXTERNRT int rtxDListToUTF8Str (struct OSCTXT *pctxt, OSRTDList *pList, OSUTF8CHAR **ppstr, char sep)
  This function concatenates all of the components in the given list to form a UTF-8 string.

5.7.1 Detailed Description

The doubly-linked list utility functions provide common routines for managing linked lists. These lists are used to model XSD list and repeating element types within the generated code. This list type contains forward and backward pointers allowing the list to be traversed in either direction.

5.7.2 Function Documentation

5.7.2.1 EXTERNRT OSRTDListNode* rtxDListAppend (struct OSCTXT *pctxt, OSRTDList *pList, void *pData)

This function appends an item to the linked list structure.

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

Parameters:

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

Returns:

A pointer to an allocated node structure used to link the given data value into the list.
5.7.2.2 EXTERNRT int rtxDListAppendArray (struct OSCTXT *pctxt, OSRTDList *pList, void *pArray, OSUINT32 numElements, size_t elemSize)

This function appends pointers to items in the given array to a doubly linked list structure. The array is assumed to hold an array of values as opposed to pointers. The actual address of each item in the array is stored - a copy of each item is not made.

Parameters:
- *pctxt  A pointer to a context structure.
- *pList  A pointer to the linked list structure onto which the array items will be appended.
- *pArray A pointer to the source array to be converted.
- numElements  The number of elements in the array.
- elemSize  The size of one element in the array. Use the sizeof() operator to pass this parameter.

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

5.7.2.3 EXTERNRT int rtxDListAppendArrayCopy (struct OSCTXT *pctxt, OSRTDList *pList, const void *pArray, OSUINT32 numElements, size_t elemSize)

This function appends a copy of each item in the given array to a doubly linked list structure. In this case, the rtxMemAlloc function is used to allocate memory for each item and a copy is made.

Parameters:
- *pctxt  A pointer to a context structure.
- *pList  A pointer to the linked list structure onto which the array items will be appended.
- *pArray A pointer to the source array to be converted.
- numElements  The number of elements in the array.
- elemSize  The size of one element in the array. Use the sizeof() operator to pass this parameter.

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

5.7.2.4 EXTERNRT OSRTDListNode* rtxDListFindByData (const OSRTDList *pList, void *data)

This function will return the node pointer of the given data item within the list or NULL if the item is not found.

Parameters:
- *pList  A pointer to a linked list structure.
- data  Pointer to the data item to search for. Note that comparison of pointer values is done; not the items pointed at by the pointers.

Returns:
A pointer to an allocated linked list node structure.
5.7.2.5 EXTERNRT OSRTDListNode * rtxListFindByIndex (const OSRTDList * pList, OSUINT32 index)

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

Parameters:

pList A pointer to a linked list structure.

index Zero-based index into list where the specified item is located. If the list contains fewer items than the index, NULL is returned.

Returns:

A pointer to an allocated linked list node structure. To get the actual data item, the data member variable pointer within this structure must be dereferenced.

5.7.2.6 EXTERNRT int rtxListFindIndexByData (const OSRTDList * pList, void * data)

This function will return the index of the given data item within the list or -1 if the item is not found.

Parameters:

pList A pointer to a linked list structure.

data Pointer to the data item to search for. Note that comparison of pointer values is done; not the items pointed at by the pointers.

Returns:

Index of item within the list or -1 if not found.

5.7.2.7 EXTERNRT void rtxListFreeAll (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 rtMemAlloc function was used to allocate memory for the data items.

Parameters:

pctxt A pointer to a context structure.

pList A pointer to a linked list structure.

5.7.2.8 EXTERNRT void rtxListFreeNode (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 rtMemFree or rtFreeContext function is called with this context.

Parameters:

pctxt A pointer to a context structure.

pList A pointer to a linked list structure.

node Pointer to the list node to be removed.
5.7.2.9  EXTERNRT void rtxDListFreeNodes (struct OSCTXT * pctxt, OSRTDList * pList)

This function will free all of the dynamic memory used to hold the list node pointers.
It does not free the data items because it is unknown how the memory was allocated for these items.

Parameters:

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

5.7.2.10 EXTERNRT void rtxDListInit (OSRTDList * pList)

This function initializes a doubly linked list structure.
It sets the number of elements to zero and sets all internal pointer values to NULL. A doubly linked-list structure is
described by the OSRTDList type. Nodes of the list are of type OSRTDListNode.
Memory for the structures is allocated using the rtxMemAlloc run-time function and is maintained within the
context structure that is a required parameter to all rtDList functions. This memory is released when rtxMemFree is
called or the context is released. Unless otherwise noted, all data passed into the list functions is simply stored on the
list by value (i.e. a deep-copy of the data is not done).

Parameters:

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

5.7.2.11  EXTERNRT OSRTDListNode* rtxDListInsert (struct OSCTXT * pctxt, OSRTDList * pList, OSUINT32 index, void * pData)

This function inserts an item into the linked list structure.
The data item is passed into the function as a void pointer that can point to an object of any type. The rtxMemAlloc
function is used to allocate memory for the list node structure; therefore, all internal list memory will be released when
the rtxMemFree function is called.

Parameters:

    pctxt  A pointer to a context structure. This provides a storage area for the function to store all working variables
           that must be maintained between function calls.
    pList  A pointer to a linked list structure into which the data item is to be inserted.
    index  Zero-based index into list where the specified item is to be inserted.
    pData  A pointer to the data item to be inserted to the list.

Returns:

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

5.7.2.12  EXTERNRT OSRTDListNode* rtxDListInsertAfter (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node, void * pData)

This function inserts an item into the linked list structure after the specified element.
The rtxMemAlloc function is used to allocate memory for the list node structure; therefore, all internal list memory
will be released when the rtxMemFree function is called.
Parameters:

- **pctxt**: A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
- **pList**: A pointer to a linked list structure into which the data item is to be inserted.
- **node**: The position in the list where the item is to be inserted. The item will be inserted after this node or added as the head element if node is null.
- **pData**: A pointer to the data item to be inserted to the list.

Returns:

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

5.7.2.13 **EXTERNRT OSRTDListNode* rtxDListInsertBefore (struct OSCTXT * pctxt, OSRTDList * pList, OSRTDListNode * node, void * pData)**

This function inserts an item into the linked list structure before the specified element.

The rtxMemAlloc function is used to allocate memory for the list node structure; therefore, all internal list memory will be released when the rtxMemFree function is called.

Parameters:

- **pctxt**: A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
- **pList**: A pointer to a linked list structure into which the data item is to be inserted.
- **node**: The position in the list where the item is to be inserted. The item will be inserted before this node or appended to the list if node is null.
- **pData**: A pointer to the data item to be inserted to the list.

Returns:

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

5.7.2.14 **EXTERNRT void rtxDListRemove (OSRTDList * pList, OSRTDListNode * node)**

This function will remove the given node from the list.

The node memory is not freed. It will be released when the rtxMemFree or rtFreeContext function is called with this context.

Parameters:

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

5.7.2.15 **EXTERNRT int rtxDListToArray (struct OSCTXT * pctxt, OSRTDList * pList, void ** ppArray, OSUINT32 * pElemCount, size_t elemSize)**

This function converts a doubly linked list to an array.
Parameters:

- **pctxt** A pointer to a context structure.
- **pList** A pointer to a linked list structure.
- **ppArray** A pointer to a pointer to the destination array.
- **pElemCount** A pointer to the number of elements already allocated in ppArray. If pElements is NULL, or pElements is less than the number of nodes in the list, then a new array is allocated and the pointer is stored in ppArray. Memory is allocated via calls to the rtxMemAlloc function.
- **elemSize** The size of one element in the array. Use the sizeof() operator to pass this parameter.

Returns:

The number of elements in the returned array.

5.7.2.16 **EXTERNRT int rtxDListToUTF8Str (struct OSCTXT *pctxt, OSRTDList *pList, OSUTF8CHAR **ppstr, char sep)**

This function concatenates 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 character is inserted after each list component. The rtxMemAlloc function is used to allocate memory for the output string.

Parameters:

- **pctxt** A pointer to a context structure.
- **pList** A pointer to the linked list structure onto which the array items will be appended.
- **ppstr** A pointer to a char pointer to hold output string.
- **sep** Separator character to add between string components.

Returns:

Completion status of operation: 0 (0) = success, negative return value is error.
5.8 Enumeration utility functions

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

Functions

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

- **EXTERNRT OSINT32 rtxLookupEnumByValue** (OSINT32 value, const OSEnumItem enumTable[], size_t enumTableSize)
  
  *Lookup enum by integer value.*

- **EXTERNRT int rtxTestNumericEnum** (OSINT32 ivalue, const OSNumericEnumItem enumTable[], OSUINT16 enumTableSize)
  
  *This function determines if the given numeric enumerated value is within the defined numeration set.*

5.8.1 Detailed Description

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

5.8.2 Function Documentation

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

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

Parameters:

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

Returns:

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

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

 Lookup enum by integer value.

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

Parameters:

- **value** Integer value of the enumerated item.
**enumTable**  Table containing the defined enumeration

**enumTableSize**  Number of rows in the table

Returns:

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

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

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

Parameters:

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

Returns:

Zero (0) if item in table, RTERR_INVENUM if not
5.9 Run-time error status codes.

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

Defines

- `#define RT_OK 0`
  
  *Normal completion status.*

- `#define RT_OK_FRAG 2`
  
  *Message fragment return status.*

- `#define RTERR_BUFOVFLW -1`
  
  *Encode buffer overflow.*

- `#define RTERR_ENDOFBUF -2`
  
  *Unexpected end-of-buffer.*

- `#define RTERR_IDNOTFOU -3`
  
  *Expected identifier not found.*

- `#define RTERR_INVENUM -4`
  
  *Invalid enumerated identifier.*

- `#define RTERR_SETDUPL -5`
  
  *Duplicate element in set.*

- `#define RTERR_SETMISRQ -6`
  
  *Missing required element in set.*

- `#define RTERR_NOTINSET -7`
  
  *Element not in set.*

- `#define RTERR_SEQOVFLW -8`
  
  *Sequence overflow.*

- `#define RTERR_INVOPT -9`
  
  *Invalid option in choice.*

- `#define RTERR_NOMEM -10`
  
  *No dynamic memory available.*

- `#define RTERR_INVHEXS -11`
  
  *Invalid hexadecimal string.*

- `#define RTERR_INVREAL -12`
  
  *Invalid real number value.*

- `#define RTERR_STROVFLW -13`
String overflow.

- #define RTERR_BADVALUE -14
  Bad value.

- #define RTERR_TOODEEP -15
  Nesting level too deep.

- #define RTERR_CONSVIO -16
  Constraint violation.

- #define RTERR_ENDOFFILE -17
  Unexpected end-of-file error.

- #define RTERR_INVUTF8 -18
  Invalid UTF-8 character encoding.

- #define RTERR_OUTOFBND -19
  Array index out-of-bounds.

- #define RTERR_INVPARAM -20
  Invalid parameter passed to a function of method.

- #define RTERR_INVFORMAT -21
  Invalid value format.

- #define RTERR_NOTINIT -22
  Context not initialized.

- #define RTERR_TOOBIG -23
  Value will not fit in target variable.

- #define RTERR_INVCHAR -24
  Invalid character.

- #define RTERR_XMLSTATE -25
  XML state error.

- #define RTERR_XMLPARSE -26
  XML parser error.

- #define RTERR_SEQORDER -27
  Sequence order error.

- #define RTERR_FILNOTFOU -28
  File not found.

- #define RTERR_READERR -29
  Read error.
• #define RTERR_WRITEERR -30
   Write error.

• #define RTERR_INVBASE64 -31
   Invalid Base64 encoding.

• #define RTERR_INVSOCKET -32
   Invalid socket.

• #define RTERR_INVATTR -33
   Invalid attribute.

• #define RTERR_REGEXP -34
   Invalid regular expression.

• #define RTERR_PATMATCH -35
   Pattern match error.

• #define RTERR_ATTRMISRQ -36
   Missing required attribute.

• #define RTERR_HOSTNOTFOU -37
   Host name could not be resolved.

• #define RTERR_HTTPERR -38
   HTTP protocol error.

• #define RTERR_SOAPERR -39
   SOAP error.

• #define RTERR_EXPIRED -40
   Evaluation license expired.

• #define RTERR_UNEXPELEM -41
   Unexpected element encountered.

• #define RTERR_INVOCCUR -42
   Invalid number of occurrences.

• #define RTERR_INVMSGBUF -43
   Invalid message buffer has been passed to decode or validate method.

• #define RTERR_DECELEMFAIL -44
   Element decode failed.

• #define RTERR_DECATTRFAIL -45
   Attribute decode failed.

• #define RTERR_STRMINUSE -46
   Stream in-use.
• #define RTERR_NULLPTR -47
   Null pointer.

• #define RTERR_FAILED -48
   General failure.

• #define RTERR_ATTRFIXEDVAL -49
   Attribute fixed value mismatch.

• #define RTERR_MULTIPLE -50
   Multiple errors occurred during an encode or decode operation.

• #define RTERR_NOTYPEINFO -51
   This error is returned when decoding a derived type definition and no information exists as to what type of data is in the element content.

• #define RTERR_ADDRINUSE -52
   Address already in use.

• #define RTERR_CONNRESET -53
   Remote connection was reset.

• #define RTERR_UNREACHABLE -54
   Network failure.

• #define RTERR_NOCONN -55
   Not connected.

• #define RTERR_CONNREFUSED -56
   Connection refused.

• #define RTERR_INVSOCKOPT -57
   Invalid option.

• #define RTERR_SOAPFAULT -58
   This error is returned when decoded SOAP envelope is fault message.

• #define RTERR_MARKNOTSUP -59
   This error is returned when an attempt is made to mark a stream position on a stream type that does not support it.

• #define RTERR_NOTSUPP -60
   Feature is not supported.

5.9.1 Detailed Description

This is a list of status codes that can be returned by the common run-time functions and generated code. In many cases, additional information and parameters for the different errors are stored in the context structure at the time the error in raised. This additional information can be output using the rtxErrPrint or rtxErrLogUsingCB run-time functions.
5.9.2 Define Documentation

5.9.2.1 #define RT_OK_FRAG 2

Message fragment return status.
This is returned when a part of a message is successfully decoded. The application should continue to invoke the
decode function until a zero status is returned.
Definition at line 53 of file rtxErrCodes.h.

5.9.2.2 #define RTERR_ADDRINUSE -52

Address already in use.
This status code is returned when an attempt is made to bind a socket to an address that is already in use.
Definition at line 471 of file rtxErrCodes.h.

5.9.2.3 #define RTERR_ATTRFIXEDVAL -49

Attribute fixed value mismatch.
The attribute contained a value that was different than the fixed value defined in the schema for the attribute.
Definition at line 444 of file rtxErrCodes.h.

5.9.2.4 #define RTERR_ATTRMISRQ -36

Missing required attribute.
This status code is returned by the decoder when an XML instance is missing a required attribute value as defined in
the XML schema.
Definition at line 347 of file rtxErrCodes.h.

5.9.2.5 #define RTERR_BADVALUE -14

Bad value.
This status code is returned anywhere where an API is expecting a value to be within a certain range and it not within
this range. An example is the encoding or decoding date values when the month or day value is not within the legal
range (1-12 for month and 1 to whatever the max days is for a given month).
Definition at line 170 of file rtxErrCodes.h.

5.9.2.6 #define RTERR_BUFOVFLW -1

Encode buffer overflow.
This status code is returned when encoding into a static buffer and there is no space left for the item currently being
encoded.
Definition at line 60 of file rtxErrCodes.h.
5.9.2.7  #define RTERR_CONNREFUSED -56

Connection refused.
This status code is returned when an attempt to communicate on an open socket is refused by the host.
Definition at line 495 of file rtxErrCodes.h.

5.9.2.8  #define RTERR_CONNRESET -53

Remote connection was reset.
This status code is returned when the connection is reset by the remote host (via explicit command or a crash.
Definition at line 477 of file rtxErrCodes.h.

5.9.2.9  #define RTERR_CONSVIO -16

Constraint violation.
This status code is returned when constraints defined the schema are violated. These include XSD facets such as min/maxOccurs, min/maxLength, patterns, etc.. Also ASN.1 value range, size, and permitted alphabet constraints.
Definition at line 185 of file rtxErrCodes.h.

5.9.2.10  #define RTERR_DECATTRFAIL -45

Attribute decode failed.
This status code and parameters are added to the failure status by the decoder to allow the specific attribute on which a decode error was detected to be identified.
Definition at line 412 of file rtxErrCodes.h.

5.9.2.11  #define RTERR_DECELEMFAIL -44

Element decode failed.
This status code and parameters are added to the failure status by the decoder to allow the specific element on which a decode error was detected to be identified.
Definition at line 405 of file rtxErrCodes.h.

5.9.2.12  #define RTERR_ENDOFBUF -2

Unexpected end-of-buffer.
This status code is returned when decoding and the decoder expects more data to be available but instead runs into the end of the decode buffer.
Definition at line 67 of file rtxErrCodes.h.

5.9.2.13  #define RTERR_ENDOFFILE -17

Unexpected end-of-file error.
This status code is returned when an unexpected end-of-file condition is detected on decode. It is similar to the ENDOFBUF error code described above except that in this case, decoding is being done from a file stream instead of from a memory buffer.
Definition at line 193 of file rtxErrCodes.h.

5.9.2.14  \#define RTERR_EXPIRED -40

Evaluation license expired.
This error is returned from evaluation versions of the run-time library when the hard-coded evaluation period is expired.
Definition at line 375 of file rtxErrCodes.h.

5.9.2.15  \#define RTERR_FAILED -48

General failure.
Low level call returned error.
Definition at line 433 of file rtxErrCodes.h.

5.9.2.16  \#define RTERR_FILNOTFOU -28

File not found.
This status code is returned if an attempt is made to open a file input stream for decoding and the given file does not exist.
Definition at line 283 of file rtxErrCodes.h.

5.9.2.17  \#define RTERR_HOSTNOTFOU -37

Host name could not be resolved.
This status code is returned from run-time socket functions when they are unable to connect to a given host computer.
Definition at line 354 of file rtxErrCodes.h.

5.9.2.18  \#define RTERR_HTTPERR -38

HTTP protocol error.
This status code is returned by functions doing HTTP protocol operations such as SOAP functions. It is returned when a protocol error is detected. Details on the specific error can be obtained by calling rtxErrPrint.
Definition at line 362 of file rtxErrCodes.h.

5.9.2.19  \#define RTERR_IDNOTFOU -3

Expected identifier not found.
This status is returned when the decoder is expecting a certain element to be present at the current position and instead something different is encountered. An example is decoding a sequence container type in which the declared elements are expected to be in the given order. If an element is encountered that is not the one expected, this error is raised.
Definition at line 77 of file rtxErrCodes.h.
5.9.2.20  #define RTERR_INVATTR -33

Invalid attribute.
This status code is returned by the decoder when an attribute is encountered in an XML instance that was not defined in the XML schema.
Definition at line 322 of file rtxErrCodes.h.

5.9.2.21  #define RTERR_INVBASE64 -31

Invalid Base64 encoding.
This status code is returned when an error is detected in decoding base64 data.
Definition at line 303 of file rtxErrCodes.h.

5.9.2.22  #define RTERR_INVCHAR -24

Invalid character.
This status code is returned when a character is encountered that is not valid for a given data type. For example, if an integer value is being decoded and a non-numeric character is encountered, this error will be raised.
Definition at line 254 of file rtxErrCodes.h.

5.9.2.23  #define RTERR_INVENUM -4

Invalid enumerated identifier.
This status is returned when an enumerated value is being encoded or decoded and the given value is not in the set of values defined in the enumeration facet.
Definition at line 84 of file rtxErrCodes.h.

5.9.2.24  #define RTERR_INVFORMAT -21

Invalid value format.
This status code is returned when a value is received or passed into a function that is not in the expected format. For example, the time string parsing function expects a string in the form "nn:nn:nn" where n’s are numbers. If not in this format, this error code is returned.
Definition at line 224 of file rtxErrCodes.h.

5.9.2.25  #define RTERR_INVHEXS -11

Invalid hexadecimal string.
This status code is returned when decoding a hexadecimal string value and a character is encountered in the string that is not in the valid hexadecimal character set ([0-9A-Fa-f] or whitespace).
Definition at line 143 of file rtxErrCodes.h.
5.9.2.26  #define RTERR_INVMSGBUF -43

Invalid message buffer has been passed to decode or validate method.

This status code is returned by decode or validate method when the used message buffer instance has type different from OSMensajeBufferIF::XMLDecode.

Definition at line 398 of file rtxErrCodes.h.

5.9.2.27  #define RTERR_INVOCCUR -42

Invalid number of occurrences.

This status code is returned by the decoder when an XML instance contains a number of occurrences of a repeating element that is outside the bounds (minOccurs/maxOccurs) defined for the element in the XML schema.

Definition at line 390 of file rtxErrCodes.h.

5.9.2.28  #define RTERR_INVOPT -9

Invalid option in choice.

This status code is returned when encoding or decoding an ASN.1 CHOICE or XSD xsd:choice construct. When encoding, it occurs when a value in the generated 't' member variable is outside the range of indexes of items in the content model group. It occurs on the decode side when an element is received that is not defined in the content model group.

Definition at line 128 of file rtxErrCodes.h.

5.9.2.29  #define RTERR_INVPARAM -20

Invalid parameter passed to a function of method.

This status code is returned by a function or method when it does an initial check on the values of parameters passed in. If a parameter is found to not have a value in the expected range, this error code is returned.

Definition at line 215 of file rtxErrCodes.h.

5.9.2.30  #define RTERR_INVREAL -12

Invalid real number value.

This status code is returned when decoding a numeric floating-point value and an invalid character is received (i.e. not numeric, decimal point, plus or minus sign, or exponent character).

Definition at line 151 of file rtxErrCodes.h.

5.9.2.31  #define RTERR_INVSOCKET -32

Invalid socket.

This status code is returned when an attempt is made to read or write from a socket and the given socket handle is invalid. This may be the result of not having established a proper connection before trying to use the socket handle variable.

Definition at line 311 of file rtxErrCodes.h.
5.9.2.32  #define RTERR_INVSOCKOPT -57
Invalid option.
This status code is returned when an invalid option is passed to socket.
Definition at line 501 of file rtxErrCodes.h.

5.9.2.33  #define RTERR_INVUTF8 -18
Invalid UTF-8 character encoding.
This status code is returned by the decoder when an invalid sequence of bytes is detected in a UTF-8 character string.
Definition at line 200 of file rtxErrCodes.h.

5.9.2.34  #define RTERR_MULTIPLE -50
Multiple errors occurred during an encode or decode operation.
See the error list within the context structure for a full list of all errors.
Definition at line 454 of file rtxErrCodes.h.

5.9.2.35  #define RTERR_NOCONN -55
Not connected.
This status code is returned when an operation is issued on an unconnected socket.
Definition at line 489 of file rtxErrCodes.h.

5.9.2.36  #define RTERR_NOMEM -10
No dynamic memory available.
This status code is returned when a dynamic memory allocation request is made and an insufficient amount of memory is available to satisfy the request.
Definition at line 135 of file rtxErrCodes.h.

5.9.2.37  #define RTERR_NOTINIT -22
Context not initialized.
This status code is returned when the run-time context structure (OSCTX) 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 there is a license violation (for example, evaluation license expired).
Definition at line 234 of file rtxErrCodes.h.

5.9.2.38  #define RTERR_NOTINSET -7
Element not in set.
This status code is returned when encoding or decoding an ASN.1 SET or XSD xsd:all construct. When encoding, it occurs when a value in the generated _order member variable is outside the range of indexes of items in the content model group. It occurs on the decode side when an element is received that is not defined in the content model group.

Definition at line 110 of file rtxErrCodes.h.

5.9.2.39  
#define RTERR_NOTSUPP -60

Feature is not supported.
This status code is returned when a feature that is currently not supported is encountered. Support may be added in a future release.
Definition at line 528 of file rtxErrCodes.h.

5.9.2.40  
#define RTERR_NOTYPEINFO -51

This error is returned when decoding a derived type definition and no information exists as to what type of data is in the element content.
When decoding XML, this normally means that an xsi:type attribute was not found identifying the type of content.
Definition at line 465 of file rtxErrCodes.h.

5.9.2.41  
#define RTERR_NULLPTR -47

Null pointer.
This status code is returned when a null pointer is encountered in a place where it is expected that the pointer value is to be set.
Definition at line 428 of file rtxErrCodes.h.

5.9.2.42  
#define RTERR_OUTOFBND -19

Array index out-of-bounds.
This status code is returned when an attempt is made to add something to an array and the given index is outside the defined bounds of the array.
Definition at line 207 of file rtxErrCodes.h.

5.9.2.43  
#define RTERR_PATMATCH -35

Pattern match error.
This status code is returned by the decoder when a value in an XML instance does not match the pattern facet defined in the XML schema. It can also be returned by numeric encode functions that cannot format a numeric value to match the pattern specified for that value.
Definition at line 340 of file rtxErrCodes.h.

5.9.2.44  
#define RTERR_READERR -29

Read error.
This status code if returned if a read I/O error is encountered when reading from an input stream associated with a physical device such as a file or socket.
Definition at line 290 of file rtxErrCodes.h.

5.9.2.45  #define RTERR_REGEXP -34
Invalid regular expression.
This status code is returned when a syntax error is detected in a regular expression value. Details of the syntax error can be obtained by invoking rtxErrPrint to print the details of the error contained within the context variable.
Definition at line 331 of file rtxErrCodes.h.

5.9.2.46  #define RTERR_SEQORDER -27
Sequence order error.
This status code is returned when decoding an ASN.1 SEQUENCE or XSD xsd:sequence construct. It is raised if the elements were received in an order different than that specified in the content model group definition.
Definition at line 276 of file rtxErrCodes.h.

5.9.2.47  #define RTERR_SEQOVFLW -8
Sequence overflow.
This status code is returned when decoding a repeating element (ASN.1 SEQUENCE OF or XSD element with min/maxOccurs > 1) and more instances of the element are received than were defined in the constraint.
Definition at line 118 of file rtxErrCodes.h.

5.9.2.48  #define RTERR_SETDUPL -5
Duplicate element in set.
This status code is returned when decoding an ASN.1 SET or XSD xsd:all construct. It is raised if a given element defined in the content model group occurs multiple times in the instance being decoded.
Definition at line 92 of file rtxErrCodes.h.

5.9.2.49  #define RTERR_SETMISRQ -6
Missing required element in set.
This status code is returned when decoding an ASN.1 SET or XSD xsd:all construct and all required elements in the content model group are not found to be present in the instance being decoded.
Definition at line 100 of file rtxErrCodes.h.

5.9.2.50  #define RTERR_SOAPERR -39
SOAP error.
This status code when an error is detected when trying to execute a SOAP operation.
Definition at line 368 of file rtxErrCodes.h.
5.9.2.51  #define RTERR_STRMINUSE -46

Stream in-use.
This status code is returned by stream functions when an attempt is made to initialize a stream or create a reader or writer when an existing stream is open in the context. The existing stream must first be closed before initializing a stream for a new operation.
Definition at line 421 of file rtxErrCodes.h.

5.9.2.52  #define RTERR_STROVFLW -13

String overflow.
This status code is returned when a fixed-sized field is being decoded as specified by a size constraint and the item contains more characters or bytes than this amount. It can occur when a run-time function is called with a fixed-sized static buffer and whatever operation is being done causes the bounds of this buffer to be exceeded.
Definition at line 161 of file rtxErrCodes.h.

5.9.2.53  #define RTERR_TOOBIG -23

Value will not fit in target variable.
This status is returned by the decoder when a target variable is not large enough to hold a decoded value. A typical case is an integer value that is too large to fit in the standard C integer type (typically a 32-bit value) on a given platform. If this occurs, it is usually necessary to use a configuration file setting to force the compiler to use a different data type for the item. For example, for integer, the <isBigInteger/> setting can be used to force use of a big integer type.
Definition at line 246 of file rtxErrCodes.h.

5.9.2.54  #define RTERR_TOODEEP -15

Nesting level too deep.
This status code is returned when a preconfigured maximum nesting level for elements within a content model group is exceeded.
Definition at line 177 of file rtxErrCodes.h.

5.9.2.55  #define RTERR_UNEXPELEM -41

Unexpected element encountered.
This status code is returned when an element is encountered in a position where something else (for example, an attribute) was expected.
Definition at line 382 of file rtxErrCodes.h.

5.9.2.56  #define RTERR_UNREACHABLE -54

Network failure.
This status code is returned when the network or host is down or otherwise unreachable.
Definition at line 483 of file rtxErrCodes.h.
5.9.2.57  #define RTERR_WRITEERR -30

Write error.
This status code if returned if a write I/O error is encountered when attempting to output data to an output stream associated with a physical device such as a file or socket.
Definition at line 297 of file rtxErrCodes.h.

5.9.2.58  #define RTERR_XMLPARSE -26

XML parser error.
This status code in returned when the underlying XML parser application (by default, this is Expat) returns an error code. The parser error code or text is returned as a parameter in the errInfo structure within the context structure.
Definition at line 268 of file rtxErrCodes.h.

5.9.2.59  #define RTERR_XMLSTATE -25

XML state error.
This status code is returned when the XML parser is not in the correct state to do a certain operation.
Definition at line 260 of file rtxErrCodes.h.
5.10 Error Formatting and Print Functions

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

Defines

- \#define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat,\_\_FILE\_\_\_LINE\_\_)
  
  This macro is used to log a run-time error in the context.

- \#define OSRTASSERT(condition) if (!\(condition\)) { rtxErrAssertionFailed(\#condition,\_\_LINE\_\_\_\_FILE\_\_\_\_\_); }

  This macro is used to check an assertion.

- \#define OSRTCHECKPARAM(condition) if (condition) { /* do nothing */ }

  This macro check a condition but takes no action.

Functions

- EXTERNRT OSBOOL rtxErrAddCtxBufParm (OSCTX\_\_t pctxt)
  
  This function adds the contents of the context buffer to the error information structure in the context.

- EXTERNRT OSBOOL rtxErrAddDoubleParm (OSCTX\_\_t pctxt, double errParm)

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

- EXTERNRT OSBOOL rtxErrAddErrorTableEntry (const char \*\* \_\_ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)

  This function adds a set of error codes to the global error table.

- EXTERNRT OSBOOL rtxErrAddElemNameParm (OSCTX\_\_t pctxt)

  This function adds an element name parameter to the context error information structure.

- EXTERNRT OSBOOL rtxErrAddIntParm (OSCTX\_\_t pctxt, int errParm)

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

- EXTERNRT OSBOOL rtxErrAddInt64Parm (OSCTX\_\_t pctxt, OSINT64 errParm)

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

- EXTERNRT OSBOOL rtxErrAddStrParm (OSCTX\_\_t pctxt, const char \*pErrParm)

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

- EXTERNRT OSBOOL rtxErrAddStrnParm (OSCTX\_\_t pctxt, const char \*pErr Parm, size_t nchars)

  This function adds a given number of characters from a character string parameter to an error information structure.

- EXTERNRT OSBOOL rtxErrAddUniStrParm (OSCTX\_\_t pctxt, const OSUNICHAR \*pErrParm)

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

- EXTERNRT OSBOOL rtxErrAddUIntParm (OSCTX\_\_t pctxt, unsigned int errParm)
This function adds an unsigned integer parameter to an error information structure.

- EXTERNRT OSBOOL rtxErrAddUInt64Parm (OSCTXT *pctxt, USINT64 errParm)
  This function adds an unsigned 64-bit integer parameter to an error information structure.

- EXTERNRT void rtxErrAssertionFailed (const char *conditionText, int lineNo, const char *fileName)
  This function is used to record an assertion failure.

- EXTERNRT const char * rtxErrFmtMsg (OSRTErrInfo *pErrInfo, char *bufp, size_t bufsiz)
  This function formats a given error structure from the context into a finished status message including substituted parameters.

- EXTERNRT void rtxErrFreeParms (OSCTXT *pctxt)
  This function is used to free dynamic memory that was used in the recording of error parameters.

- EXTERNRT char * rtxErrGetText (OSCTXT *pctxt, char *pBuf, size_t *pBufSize)
  This function returns error text in a memory buffer.

- EXTERNRT char * rtxErrGetTextBuf (OSCTXT *pctxt, char *pbuf, size_t bufsiz)
  This function returns error text in the given fixed-size memory buffer.

- EXTERNRT OSRTErrInfo * rtxErrNewNode (OSCTXT *pctxt)
  This function creates a new empty error record for the passed context.

- EXTERNRT void rtxErrInit ()
  This function is a one-time initialization function that must be called before any other error processing functions can be called.

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

- EXTERNRT void rtxErrLogUsingCB (OSCTXT *pctxt, OSErrCbFunc cb, void *cbArg_p)
  This function allows error information to be logged using a user-defined callback routine.

- EXTERNRT void rtxErrPrint (OSCTXT *pctxt)
  This function is used to print the error information stored in the context to the standard output device.

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

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

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

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

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

- EXTERNRT OSUINT32 rtxErrGetErrorCnt (const OSCTX *pctxt)
  This function returns the total number of error records.

- EXTERNRT int rtxErrGetStatus (const OSCTX *pctxt)
  This function returns the status value from the context.

- EXTERNRT int rtxErrResetLastErrors (OSCTX *pctxt, int errorsToReset)
  This function resets last `errorsToReset` errors in the context.

5.10.1 Detailed Description

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

5.10.2 Define Documentation

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

This macro is used to log a run-time error in the context.

It calls the rtxErrSetData function to set the status and error parameters. The C built-in __FILE__ and __LINE__ macros are used to record the position in the source file of the error.

Parameters:
- pctxt A pointer to a context structure.
- stat Error status value from rtxErrCodes.h

Definition at line 60 of file rtxError.h.

5.10.2.2 #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }

This macro is used to check an assertion.

This is a condition that is expected to be true. The rtxErrAssertionFailed function is called if the condition is not true. The C built-in __FILE__ and __LINE__ macros are used to record the position in the source file of the failure.

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

Definition at line 82 of file rtxError.h.

5.10.2.3 #define OSRTCHECKPARAM(condition) if (condition) { /* do nothing */ }

This macro check a condition but takes no action.

Its main use is to supress VC++ level 4 "argument not used" warnings.
Parameters:

\emph{condition} Condition to check (for example, "(ptr != NULL)")

Definition at line 91 of file rtxError.h.

5.10.3 Function Documentation

5.10.3.1 EXTERNRT OSBOOL rtxErrAddCtxBufParm (OSCTXT * \textit{pctxt})

This function adds the contents of the context buffer to the error information structure in the context.
The buffer contents are assumed to contain only printable characters.

Parameters:

\textit{pctxt} A pointer to a context structure.

Returns: 

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

5.10.3.2 EXTERNRT OSBOOL rtxErrAddDoubleParm (OSCTXT * \textit{pctxt}, double \textit{errParm})

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

Parameters:

\textit{pctxt} A pointer to a context structure.
\textit{errParm} The double error parameter.

Returns: 

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

5.10.3.3 EXTERNRT OSBOOL rtxErrAddElemNameParm (OSCTXT * \textit{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 character (?) is inserted for the name.

Parameters:

\textit{pctxt} A pointer to a context structure.

Returns: 

The status of the operation (TRUE if the parameter was sucessfully added).
5.10.3.4 EXTERNRT OSBOOL rtxErrAddErrorTableEntry (const char * ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)

This function adds a set of error codes to the global error table. It is called within context initialization functions to add errors defined for a specific domain (for example, ASN.1 encoding/decoding) to be added to the global list of errors.

Parameters:
- **ppStatusText** Pointer to table of error status text messages.
- **minErrCode** Minimum error status code.
- **maxErrCode** Maximum error status code.

Returns:
- The status of the operation (TRUE if entry sucessfully added to table).

5.10.3.5 EXTERNRT OSBOOL rtxErrAddInt64Parm (OSCTXT * pctxt, OSINT64 errParm)

This function adds a 64-bit integer parameter to an error information structure. Parameter substitution is done in much the same way as it is done in C printf statments. The base error message specification that goes along with a particular status code may have variable fields built in using “ modifiers. These would be replaced with actual parameter data.

Parameters:
- **pctxt** A pointer to a context structure.
- **errParm** The 64-bit integer error parameter.

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

5.10.3.6 EXTERNRT OSBOOL rtxErrAddIntParm (OSCTXT * pctxt, int errParm)

This function adds an integer parameter to an error information structure. Parameter substitution is done in much the same way as it is done in C printf statments. The base error message specification that goes along with a particular status code may have variable fields built in using “ modifiers. These would be replaced with actual parameter data.

Parameters:
- **pctxt** A pointer to a context structure.
- **errParm** The integer error parameter.

Returns:
- The status of the operation (TRUE if the parameter was sucessfully added).
5.10.3.7 EXTERNRT OSBOOL rtxErrAddStrParm (OSCTXT * pctxt, const char * pErrParm, size_t nchars)

This function adds a given number of characters from a character string parameter to an error information structure.

Parameters:

- **pctxt** A pointer to a context structure.
- **pErrParm** The character string error parameter.
- **nchars** Number of characters to add from pErrParm.

Returns:

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

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

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

Parameters:

- **pctxt** A pointer to a context structure.
- **pErrParm** The character string error parameter.

Returns:

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

5.10.3.9 EXTERNRT OSBOOL rtxErrAddUInt64Parm (OSCTXT * pctxt, OSUINT64 errParm)

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

Parameters:

- **pctxt** A pointer to a context structure.
- **errParm** The unsigned 64-bit integer error parameter.

Returns:

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

5.10.3.10 EXTERNRT OSBOOL rtxErrAddUIntParm (OSCTXT * pctxt, unsigned int errParm)

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

Parameters:

- **pctxt** A pointer to a context structure.
- **errParm** The unsigned integer error parameter.

Returns:

The status of the operation (TRUE if the parameter was sucessfully added).
5.10.3.11 EXTERNRT OSBOOL rtxErrAddUniStrParm (OSCTX pctxt, const OSUNICHAR * pErrParm)

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

Parameters:
- pctxt A pointer to a context structure.
- pErrParm The Unicode string error parameter.

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

5.10.3.12 EXTERNRT void rtxErrAssertionFailed (const char * conditionText, int lineNo, const char * fileName)

This function is used to record an assertion failure. It is used in conjunction with the OTRTASSERT macro. It outputs information on the condition to stderr and then calls exit to terminate the program.

Parameters:
- conditionText The condition that failed (for example, ptr != NULL)
- lineNo Line number in the program of the failure.
- fileName Name of the C source file in which the assertion failure occurred.

5.10.3.13 EXTERNRT const char * rtxErrFmtMsg (OSRTErrInfo * pErrInfo, char * bufp, size_t bufsiz)

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

Parameters:
- pErrInfo Pointer to error information structure.
- bufp Pointer to a destination buffer to receive text.
- bufsiz Size of the buffer.

Returns:
Pointer to the buffer (bufp).

5.10.3.14 EXTERNRT void rtxErrFreeParms (OSCTX pctxt)

This function is used to free dynamic memory that was used in the recording of error parameters. After an error is logged, this function should be called to prevent memory leaks.

Parameters:
- pctxt A pointer to a context structure.
5.10.3.15 **EXTERNRT OSUINT32 rtxErrGetErrorCnt (const OSCTXT * pctxt)**

This function returns the total number of error records.

**Parameters:**

- *pctxt* A pointer to a context structure.

**Returns:**

The total number of error records in the context.

5.10.3.16 **EXTERNRT int rtxErrGetFirstError (const OSCTXT * pctxt)**

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

**Parameters:**

- *pctxt* A pointer to a context structure.

**Returns:**

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

5.10.3.17 **EXTERNRT int rtxErrGetLastError (const OSCTXT * pctxt)**

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

**Parameters:**

- *pctxt* A pointer to a context structure.

**Returns:**

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

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

This function returns the status value from the context.

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

**Parameters:**

- *pctxt* A pointer to a context structure.

**Returns:**

Status code corresponding to errors in the context.
5.10.3.19 EXTERNRT char* rtxErrGetText (OSCTXT * pctxt, char * pBuf, size_t * pBufSize)

This function returns error text in a memory buffer.

If buffer pointer and buffer size are specified in parameters (not NULL) then error text will be copied in the passed buffer. Otherwise, this function allocates memory using the 'rtxMemAlloc' function. This memory is automatically freed at the time the 'rtxMemFree' or 'rtxFreeContext' functions are called. The calling function may free the memory by using 'rtxMemFreePtr' function.

Parameters:

pctxt    A pointer to a context structure.
pBuf     A pointer to a destination buffer to obtain the error text. If NULL, dynamic buffer will be allocated.
pBufSize A pointer to buffer size. If pBuf is NULL and this parameter is not, it will receive the size of allocated dynamic buffer.

Returns:

A pointer to a buffer with error text. If pBuf is not NULL, the return pointer will be equal to it. Otherwise, returns newly allocated buffer with error text. NULL, if error occurred.

5.10.3.20 EXTERNRT char* rtxErrGetTextBuf (OSCTXT * pctxt, char * pbuf, size_t bufsize)

This function returns error text in the given fixed-size memory buffer.

If the text will not fit in the buffer, it is truncated.

Parameters:

pctxt    A pointer to a context structure.
pbuf     Pointer to a destination buffer to receive text.
bufsize  Size of the buffer.

Returns:

Pointer to the buffer (pbuf).

5.10.3.21 EXTERNRT void rtxErrInit ()

This function is a one-time initialization function that must be called before any other error processing functions can be called.

It adds the common error status text codes to the global error table.

5.10.3.22 EXTERNRT void rtxErrLogUsingCB (OSCTXT * pctxt, OSErrCbFunc cb, void * cbArg_p)

This function allows error information to be logged using a user-defined callback routine.

The callback information 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). The prototype of the callback function to be passed is as follows:

int cb (const char* ptext, void* cbArg_p);

where ptext is a pointer to the formatted error text string and cbArg_p is a pointer to a user-defined callback argument.
Parameters:

\( pctxt \) A pointer to a context structure.
\( cb \) Callback function pointer.
\( cbArg_p \) Pointer to a user-defined argument that is passed to the callback function.

5.10.3.23 EXTERNRT OSRTErrInfo \( \ast \) rtxErrNewNode (OSCTXT \( \ast \ pctxt \))

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.

Parameters:

\( pctxt \) A pointer to a context structure.

Returns:

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

5.10.3.24 EXTERNRT void rtxErrPrint (OSCTXT \( \ast \ pctxt \))

This function is used to print the error information stored in the context to the standard output device.
Parameter substitution is done so that recorded parameters are added to the output message text.

Parameters:

\( pctxt \) A pointer to a context structure.

5.10.3.25 EXTERNRT void rtxErrPrintElement (OSRTErrInfo \( \ast \ pErrInfo \))

This function is used to print the error information stored in the error information element to the standard output device.
Parameter substitution is done so that recorded parameters are added to the output message text.

Parameters:

\( pErrInfo \) A pointer to an error information element.

5.10.3.26 EXTERNRT int rtxErrReset (OSCTXT \( \ast \ pctxt \))

This function is used to reset the error state recorded in the context to successful.
It is used in the generated code in places where automatic error correction can be done.

Parameters:

\( pctxt \) A pointer to a context structure.
5.10.3.27 EXTERNRT int rtxErrResetLastErrors (OSCTXT * pctxt, int errorsToReset)

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

**Parameters:**

- `pctxt` A pointer to a context structure.
- `errorsToReset` A number of errors to reset, starting from the last record.

**Returns:**

Completion status of operation:

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

5.10.3.28 EXTERNRT int rtxErrSetData (OSCTXT * pctxt, int status, const char * module, int lineno)

This function is used to record an error in the context structure.

It is typically called via the LOG_RTERR macro in the generated code to trap error conditions.

**Parameters:**

- `pctxt` A pointer to a context structure.
- `status` The error status code from rtxErrCodes.h
- `module` The C source file in which the error occurred.
- `lineno` The line number within the source file of the error.

**Returns:**

The status code that was passed in.

5.10.3.29 EXTERNRT int rtxErrSetNewData (OSCTXT * pctxt, int status, const char * module, int lineno)

This function is used to record an error in the context structure.

It is typically called via the LOG_RTERRNEW macro in the generated code to trap error conditions. It always allocates new error record.

**Parameters:**

- `pctxt` A pointer to a context structure.
- `status` The error status code from rtxErrCodes.h
- `module` The C source file in which the error occurred.
- `lineno` The line number within the source file of the error.

**Returns:**

The status code that was passed in.
5.11 Memory Buffer Management Functions

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

Functions

- **EXTERNRT int rtxMemBufAppend (OSRTMEMBUF *pMemBuf, const OSOCTET *pdata, size_t nbytes)**
  This function appends the data to the end of a memory buffer.

- **EXTERNRT int rtxMemBufCut (OSRTMEMBUF *pMemBuf, size_t fromOffset, size_t nbytes)**
  This function cuts off the part of memory buffer.

- **EXTERNRT void rtxMemBufFree (OSRTMEMBUF *pMemBuf)**
  This function frees the memory buffer.

- **EXTERNRT OSOCTET * rtxMemBufGetData (OSRTMEMBUF *pMemBuf, int *length)**
  This function returns the pointer to the used part of a memory buffer.

- **EXTERNRT int rtxMemBufGetDataLen (OSRTMEMBUF *pMemBuf)**
  This function returns the length of the used part of a memory buffer.

- **EXTERNRT void rtxMemBufInit (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, size_t segsize)**
  This function initializes a memory buffer structure.

- **EXTERNRT void rtxMemBufInitBuffer (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, OSOCTET *buf, size_t bufsize, size_t segsize)**
  This function assigns a static buffer to the memory buffer structure.

- **EXTERNRT int rtxMemBufPreAllocate (OSRTMEMBUF *pMemBuf, size_t nbytes)**
  This function allocates a buffer with a predetermined amount of space.

- **EXTERNRT void rtxMemBufReset (OSRTMEMBUF *pMemBuf)**
  This function resets the memory buffer structure.

- **EXTERNRT int rtxMemBufSet (OSRTMEMBUF *pMemBuf, OSOCTET value, size_t nbytes)**
  This function sets part of a memory buffer to a specified octet value.

- **EXTERNRT OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF *pMemBuf, OSBOOL isExpandable)**
  This function sets "isExpandable" flag for the memory buffer object.

- **EXTERNRT int rtxMemBufTrimW (OSRTMEMBUF *pMemBuf)**
  This function trims white space of the memory buffer.
5.11.1 Detailed Description

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

Dynamic memory buffers are buffers that can grow or shrink to hold variable sized amounts of data. This group of functions allows data to be appended to buffers, to be set within buffers, and to be retrieved from buffers. Currently, these functions are used within the generated SAX decode routines to collect data as it is parsed by an XML parser.

5.11.2 Function Documentation

5.11.2.1 EXTERNRT int rtxMemBufAppend (OSRTMEMBUF * pMemBuf, const OSOCTET * pdata, size_t nbytes)

This function appends the data to the end of a memory buffer.

If the buffer was dynamic and full then the buffer will be reallocated. If it is static (the static buffer was assigned by a call to rtxMemBufInitBuffer) or it is empty (no memory previously allocated) then a new buffer will be allocated.

Parameters:
- **pMemBuf** A pointer to a memory buffer structure.
- **pdata** The pointer to the buffer to be appended. The data will be copied at the end of the memory buffer.
- **nbytes** The number of bytes to be copied from pData.

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

5.11.2.2 EXTERNRT int rtxMemBufCut (OSRTMEMBUF * pMemBuf, size_t fromOffset, size_t nbytes)

This function cuts off the part of memory buffer.

The beginning of the cutting area is specified by offset “fromOffset” and the length is specified by "nbytes". All data in this part will be lost. The data from the offset "fromOffset + nbytes" will be moved to “fromOffset” offset.

Parameters:
- **pMemBuf** A pointer to a memory buffer structure.
- **fromOffset** The offset of the beginning part, being cut off.
- **nbytes** The number of bytes to be cut off from the memory buffer.

Returns:
Completion status of operation:
- 0 = success,
- negative return value is error.
5.11.2.3 EXTERNRT void rtxMemBufFree (OSRTMEMBUF *pMemBuf)

This function frees the memory buffer.
If memory was allocated then it will be freed. Do not use the memory buffer structure after this function is called.

Parameters:

\[pMemBuf\] A pointer to a memory buffer structure.

5.11.2.4 EXTERNRT OSOCTET * rtxMemBufGetData (OSRTMEMBUF *pMemBuf, int *length)

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

Parameters:

\[pMemBuf\] A pointer to a memory buffer structure.
\[length\] The pointer to the length of the used part of the memory buffer.

Returns:

The pointer to the used part of the memory buffer.

5.11.2.5 EXTERNRT int rtxMemBufGetDataLen (OSRTMEMBUF *pMemBuf)

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

Parameters:

\[pMemBuf\] A pointer to a memory buffer structure.

Returns:

The length of the used part of the buffer.

5.11.2.6 EXTERNRT void rtxMemBufInit (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, size_t segsize)

This function initializes a memory buffer structure.
It does not allocate memory; it sets the fields of the structure to the proper states. This function must be called before any operations with the memory buffer.

Parameters:

\[pCtxt\] A provides a storage area for the function to store all working variables that must be maintained between function calls.
\[pMemBuf\] A pointer to the initialized memory buffer structure.
\[segsize\] The number of bytes in which the memory buffer will be expanded incase it is full.
5.11.2.7 EXTERNRT void rtxMemBufInitBuffer (OSCTXT * pCtxt, OSRTMEMBUF * pMemBuf, OSOCTET * buf, size_t bufsize, size_t segsize)

This function assigns a static buffer to the memory buffer structure.

It does not allocate memory; it sets the pointer to the passed buffer. If additional memory is required (for example, additional data is appended to the buffer using rtxMemBufAppend), a dynamic buffer will be allocated and all data copied to the new buffer.

Parameters:

pCtxt A pointer to a context structure. This provides a storage area for the function to 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.

5.11.2.8 EXTERNRT int rtxMemBufPreAllocate (OSRTMEMBUF * pMemBuf, size_t nbytes)

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

Parameters:

pMemBuf A pointer to a memory buffer structure.

nbytes The number of bytes to be copied from pData.

Returns:

Completion status of operation:

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

5.11.2.9 EXTERNRT void rtxMemBufReset (OSRTMEMBUF * pMemBuf)

This function resets the memory buffer structure.

It does not free memory, just sets the pointer to the beginning and the used length to zero.

Parameters:

pMemBuf A pointer to a memory buffer structure.

5.11.2.10 EXTERNRT int rtxMemBufSet (OSRTMEMBUF * pMemBuf, OSOCTET value, size_t nbytes)

This function sets part of a memory buffer to a specified octet value.

The filling is started from the end of the memory buffer. If the buffer is dynamic and full, then the buffer will be reallocated. If it is static (a static buffer was assigned by a call to rtxMemBufInitBuffer) or it is empty (no memory previously was allocated) then a new buffer will be allocated.
Parameters:

\textit{pMemBuf} A pointer to a memory buffer structure.
\textit{value} The pointer to the buffer to be appended. The data will be copied at the end of the memory buffer.
\textit{nbytes} The number of bytes to be copied from pData.

Returns:

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

5.11.2.11 \texttt{EXTERNRT OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF * pMemBuf, OSBOOL isExpandable)}

This function sets "isExpandable" flag for the memory buffer object.

By default, this flag is set to TRUE, thus, memory buffer could be expanded, even if it was initialized by static buffer (see \texttt{rtMemBufInitBuffer}). If flag is cleared and buffer is full the rtMemBufAppend/rtMemBufPreAllocate functions will return error status.

Parameters:

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

Returns:

Previous state of "isExpandable" flag.

5.11.2.12 \texttt{EXTERNRT int rtxMemBufTrimW (OSRTMEMBUF * pMemBuf)}

This function trims white space of the memory buffer.

Parameters:

\textit{pMemBuf} A pointer to a memory buffer structure.

Returns:

Completion status of operation:
- $0 =$ success,
- negative return value is error.
5.12 Memory Allocation Macros and Functions

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

Defines

- `#define OSRTALLOCTYPE(pctxt, type) (type∗) rtxMemHeapAlloc (&(pctxt) → pMemHeap, sizeof(type))`
  
  This macro allocates a single element of the given type.

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

- `#define rtxMemAllocType(pctxt, ctype) (ctype∗) rtxMemHeapAlloc(&(pctxt) → pMemHeap, sizeof(ctype))`
  
  Allocate type.

- `#define rtxMemAllocTypeZ(pctxt, ctype) (ctype∗) rtxMemHeapAllocZ(&(pctxt) → pMemHeap, sizeof(ctype))`
  
  Allocate type and zero memory.

- `#define rtxMemAllocArray(pctxt, n, type) (type∗) rtxMemHeapAlloc (&(pctxt) → pMemHeap, sizeof(type)*n)`
  
  Allocate a dynamic array.

- `#define rtxMemNewAutoPtr(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt) → pMemHeap, nbytes)`
  
  This function allocates a new block of memory and creates an auto-pointer with reference count set to one.

- `#define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt) → pMemHeap, (void∗)(ptr))`
  
  This function increments the auto-pointer reference count.

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

- `#define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&(pctxt) → pMemHeap, (void∗)(ptr))`
  
  This function returns the reference count of the given pointer.

Functions

- `EXTERNRT void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)`
  
  This function sets the pointers to standard allocation functions.

- `EXTERNRT void rtxMemSetDefBlkSize (OSUINT32 blkSize)`
  
  This function sets the minimum size and the granularity of memory blocks for newly created memory heaps.

- `EXTERNRT OSUINT32 rtxMemGetDefBlkSize ()`
  
  This function returns the actual granularity of memory blocks.
5.12.1 Detailed Description

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

5.12.2 Define Documentation

5.12.2.1 #define OSRTALLOC(pctxt, type) (type *) rtxMemHeapAlloc (&(pctxt) → pMemHeap, sizeof(type))

This macro allocates a single element of the given type.

Parameters:

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

Definition at line 67 of file rtxMemory.h.

5.12.2.2 #define OSRTALLOCZ(pctxt, type) (type *) rtxMemHeapAllocZ (&(pctxt) → pMemHeap, sizeof(type))

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

Parameters:

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

Definition at line 76 of file rtxMemory.h.

5.12.2.3 #define rtxMemAllocArray(pctxt, n, type) (type *) rtxMemHeapAlloc (&(pctxt) → pMemHeap, sizeof(type)*n)

Allocate a dynamic array.

This macro allocates a dynamic array of records of the given type. The pointer to the allocated array is returned to the caller.

Parameters:

- *pctxt* - Pointer to a context block
- *n* - Number of records to allocate
- *type* - Data type of an array record

Definition at line 382 of file rtxMemory.h.
5.12.2.4  #define rtxMemAllocType(pctxt, ctype) (ctype ∗)rtxMemHeapAlloc(&((pctxt) → pMemHeap,sizeof(ctype)))

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

Parameters:
  - *pctxt* - Pointer to a context block
  - *ctype* - Name of C typedef

Returns:
  - Pointer to allocated memory or NULL if insufficient memory was available to fulfill the request.

Definition at line 344 of file rtxMemory.h.

5.12.2.5  #define rtxMemAllocTypeZ(pctxt, ctype) (ctype ∗)rtxMemHeapAllocZ(&((pctxt) → pMemHeap,sizeof(ctype)))

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

Parameters:
  - *pctxt* - Pointer to a context block
  - *ctype* - Name of C typedef

Returns:
  - Pointer to allocated memory or NULL if insufficient memory was available to fulfill the request.

Definition at line 356 of file rtxMemory.h.

5.12.2.6  #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&((pctxt) → pMemHeap, (void ∗)(ptr)))

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

Parameters:
  - *pctxt*  Pointer to a context structure.
  - *ptr*  Pointer on which reference count is to be fetched.

Returns:
  Pointer reference count.

Definition at line 467 of file rtxMemory.h.
5.12.2.7  #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt) → pMemHeap, (void*)ptr)

This function increments the auto-pointer reference count.

Parameters:
   pctxt  Pointer to a context structure.
   ptr   Pointer on which reference count is to be incremented.

Returns:
   Referenced pointer value (ptr argument) or NULL if reference count could not be incremented.

Definition at line 443 of file rtxMemory.h.

5.12.2.8  #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref(&(pctxt) → pMemHeap, (void*)ptr)

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

Parameters:
   pctxt  Pointer to a context structure.
   ptr   Pointer on which reference count is to be decremented.

Returns:
   Positive reference count or a negative error code. If zero, memory held by pointer will have been freed.

Definition at line 456 of file rtxMemory.h.

5.12.2.9  #define rtxMemNewAutoPtr(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt) → pMemHeap, nbytes)

This function allocates a new block of memory and creates an auto-pointer with reference count set to one.
The rtxMemAutoPtrRef and rtxMemAutoPtrUnref functions can be used to increment and decrement the
reference count. When the count goes to zero, the memory held by the pointer is freed.

Parameters:
   pctxt  Pointer to a context structure.
   nbytes Number of bytes to allocate.

Returns:
   Pointer to allocated memory or NULL if not enough memory is available.

Definition at line 432 of file rtxMemory.h.
5.12.3 Function Documentation

5.12.3.1 EXTERNRT OSUINT32 rtxMemGetDefBlkSize ()

This function returns the actual granularity of memory blocks.

Returns:

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

5.12.3.2 EXTERNRT OSBOOL rtxMemIsZero (const void * pmem, size_t memsiz)

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

Parameters:

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

Returns:

Boolean result: true if memory is all zero

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

This function sets the pointers to standard allocation functions.

These functions are used to allocate/reallocate/free memory blocks. By default, standard C functions - ‘malloc’, ‘realloc’ and ‘free’ - are used. But if some platforms do not support these functions (or some other reasons exist) they can be overloaded. The functions being overloaded should have the same prototypes as the standard functions.

Parameters:

  malloc_func  Pointer to the memory allocation function (’malloc’ by default).
  realloc_func Pointer to the memory reallocation function (’realloc’ by default).
  free_func    Pointer to the memory deallocation function (’free’ by default).

5.12.3.4 EXTERNRT void rtxMemSetDefBlkSize (OSUINT32 blkSize)

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

Parameters:

  blkSize  The minimum size and the granularity of memory blocks.
5.13 Pattern matching functions

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

Functions

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

  This function compares the given string to the given pattern.

5.13.1 Detailed Description

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

5.13.2 Function Documentation

5.13.2.1 EXTERNRT OSBOOL rtxMatchPattern (OSCTXT *pctxt, const OSUTF8CHAR *text, const OSUTF8CHAR *pattern)

This function compares the given string to the given pattern.

It returns true if match, false otherwise.

Parameters:

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

Returns:

Boolean result.
5.14 Print Functions

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

Functions

- EXTERNRT int rtxByteToHexChar (OSOCTET byte, char *buf, size_t bufsize)
  This function converts a byte value into its hex string equivalent.

- EXTERNRT void rtxPrintBoolean (const char *name, OSBOOL value)
  Prints a boolean value to stdout.

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

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

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

- EXTERNRT void rtxPrintInteger (const char *name, OSINT32 value)
  Prints an integer value to stdout.

- EXTERNRT void rtxPrintInt64 (const char *name, OSINT64 value)
  Prints a 64-bit integer value to stdout.

- EXTERNRT void rtxPrintUnsigned (const char *name, OSUINT32 value)
  Prints an unsigned integer value to stdout.

- EXTERNRT void rtxPrintUInt64 (const char *name, OSUINT64 value)
  Prints an unsigned 64-bit integer value to stdout.

- EXTERNRT void rtxPrintHexStr (const char *name, OSUINT32 numocts, const OSOCTET *data)
  This function prints the value of a binary string in hex format to standard output.

- EXTERNRT void rtxPrintHexBinary (const char *name, OSUINT32 numocts, const OSOCTET *data)
  Prints an octet string value in hex binary format to stdout.

- EXTERNRT void rtxPrintCharStr (const char *name, const char *cstring)
  Prints an ASCII character string value to stdout.

- EXTERNRT void rtxPrintUTF8CharStr (const char *name, const OSUTF8CHAR *cstring)
  Prints a UTF-8 encoded character string value to stdout.

- EXTERNRT void rtxPrintUnicodeCharStr (const char *name, const OSUNICHAR *str, int nchars)
  This function prints a Unicode string to standard output.

- EXTERNRT void rtxPrintReal (const char *name, OSREAL value)
- EXTERNRT void rtxPrintNull (const char *name)
  
  Prints a NULL value to stdout.

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

- EXTERNRT int rtxPrintFile (const char *filename)
  
  This function prints the contents of a text file to stdout.

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

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

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

- EXTERNRT void rtxPrintCloseBrace (void)
  
  This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.

- EXTERNRT void rtxPrintOpenBrace (const char *)
  
  This function opens a braced region by printing indent spaces, printing the name and opening brace, and increasing the indent level.

- EXTERNRT void rtxHexDumpToNamedFile (const char *filename, const OSOCTET *data, OSUINT32 numocts)
  
  This function outputs a hexadecimal dump of the current buffer contents to the file with the given name.

- EXTERNRT void rtxHexDumpToFile (FILE *fp, const OSOCTET *data, OSUINT32 numocts)
  
  This function outputs a hexadecimal dump of the current buffer contents to a file.

- EXTERNRT void rtxHexDumpToFileEx (FILE *fp, const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)
  
  This function outputs a hexadecimal dump of the current buffer to a file, but it may output the dump as an array of bytes, words, or double words.

- EXTERNRT void rtxHexDump (const OSOCTET *data, OSUINT32 numocts)
  
  This function outputs a hexadecimal dump of the current buffer contents to stdout.

- EXTERNRT void rtxHexDumpEx (const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)
  
  This function outputs a hexadecimal dump of the current buffer contents to stdout, but it may display the dump as an array or bytes, words, or double words.

- EXTERNRT int rtxHexDumpToString (const OSOCTET *data, OSUINT32 numocts, char *buffer, int bufferIndex, int bufferSize)
  
  This function formats a hexadecimal dump of the current buffer contents to a string.
**EXTERNRT int rtxHexDumpToStringEx (const OSOCTET *data, OSUINT32 numocts, char *buffer, int bufferIndex, int bufferSize, int bytesPerUnit)**

*This function formats a hexadecimal dump of the current buffer contents to a string, but it may output the dump as an array of bytes, words, or double words.*

### 5.14.1 Detailed Description

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

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

### 5.14.2 Function Documentation

#### 5.14.2.1 EXTERNRT int rtxByteToHexChar (OSOCTET byte, char *buf, size_t bufsize)

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

**Parameters:**

- **byte** Byte to format.
- **buf** Output buffer.
- **bufsize** Output buffer size.

#### 5.14.2.2 EXTERNRT void rtxHexDump (const OSOCTET *data, OSUINT32 numocts)

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

**Parameters:**

- **data** The pointer to a buffer to be displayed.
- **numocts** The number of octets to be displayed.

#### 5.14.2.3 EXTERNRT void rtxHexDumpEx (const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)

This function outputs a hexadecimal dump of the current buffer contents to stdout, but it may display the dump as an array of bytes, words, or double words.

**Parameters:**

- **data** The pointer to a buffer to be displayed.
- **numocts** The number of octets to be displayed.
- **bytesPerUnit** The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).
5.14.2.4 EXTERNRT void rtxHexDumpToFile (FILE *fp, const OSOCTET *data, OSUINT32 numocts)

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

Parameters:

- *fp* A pointer to FILE structure. The file should be opened for writing.
- *data* The pointer to a buffer to be displayed.
- *numocts* The number of octets to be displayed

5.14.2.5 EXTERNRT void rtxHexDumpToFileEx (FILE *fp, const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)

This function outputs a hexadecimal dump of the current buffer to a file, but it may output the dump as an array of bytes, words, or double words.

Parameters:

- *fp* A pointer to FILE structure. The file should be opened for writing.
- *data* The pointer to a buffer to be displayed.
- *numocts* The number of octets to be displayed.
- *bytesPerUnit* The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

5.14.2.6 EXTERNRT void rtxHexDumpToNamedFile (const char *filename, const OSOCTET *data, OSUINT32 numocts)

This function outputs a hexadecimal dump of the current buffer contents to the file with the given name.

The file is opened or created and then closed after the writer operation is complete.

Parameters:

- *filename* Full path to file to which data should be output.
- *data* The pointer to a buffer to be displayed.
- *numocts* The number of octets to be displayed

5.14.2.7 EXTERNRT int rtxHexDumpToString (const OSOCTET *data, OSUINT32 numocts, char *buffer, int bufferIndex, int bufferSize)

This function formats a hexadecimal dump of the current buffer contents to a string.

Parameters:

- *data* The pointer to a buffer to be displayed.
- *numocts* The number of octets to be displayed.
- *buffer* The destination string buffer.
- *bufferIndex* The starting position in the destination buffer. The formatting of the dump will begin at this position.
- *bufferSize* The total size of the destination buffer.

Returns:

The length of the final string.
5.14.2.8  EXTERNRT int rtxHexDumpToStringEx (const OSOCTET *data, OSUINT32 numocts, char *buffer, int bufferIndex, int bufferSize, int bytesPerUnit)

This function formats a hexadecimal dump of the current buffer contents to a string, but it may output the dump as an array of bytes, words, or double words.

Parameters:
  
  *data  The pointer to a buffer to be displayed.
  *numocts  The number of octets to be displayed.
  *buffer  The destination string buffer.
  bufferIndex  The starting position in the destination buffer. The formatting of the dump will begin at this position.
  bufferSize  The total size of the destination buffer.
  *bytesPerUnit  The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

Returns:
  The length of the final string.

5.14.2.9  EXTERNRT void rtxPrintBoolean (const char *name, OSBOOL value)

Prints a boolean value to stdout.

Parameters:
  
  *name  The name of the variable to print.
  value  Boolean value to print.

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

Prints an ASCII character string value to stdout.

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

5.14.2.11  EXTERNRT void rtxPrintDate (const char *name, const OSNumDateTime *pvalue)

Prints a date value to stdout.

Parameters:
  
  *name  Name of the variable to print.
  *pvalue  Pointer to a structure that holds numeric DateTime value to print.
5.14.2.12 EXTERNRT void rtxPrintDateTime (const char * name, const OSNumDateTime * pvalue)

Prints a dateTime value to stdout.

Parameters:

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

5.14.2.13 EXTERNRT int rtxPrintFile (const char * filename)

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

Parameters:

  filename  The name of the text file to print.

Returns:

  Status of operation, 0 if success.

5.14.2.14 EXTERNRT void rtxPrintHexBinary (const char * name, OSUINT32 numocts, const OSOCTET * data)

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

Parameters:

  name  The name of the variable to print.
  numocts  The number of octets to be printed.
  data  A pointer to the data to be printed.

5.14.2.15 EXTERNRT void rtxPrintHexStr (const char * name, OSUINT32 numocts, const OSOCTET * data)

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

If the string is 32 bytes or less, it is printed on a single line with a `0x` prefix. If longer, a formatted hex dump showing both hex and ascii codes is done.

Parameters:

  name  The name of the variable to print.
  numocts  The number of octets to be printed.
  data  A pointer to the data to be printed.
5.14.2.16 EXTERNRT void rtxPrintInt64 (const char *name, OSINT64 value)

Prints a 64-bit integer value to stdout.

Parameters:

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

5.14.2.17 EXTERNRT void rtxPrintInteger (const char *name, OSINT32 value)

Prints an integer value to stdout.

Parameters:

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

5.14.2.18 EXTERNRT void rtxPrintNull (const char *name)

Prints a NULL value to stdout.

Parameters:

- name The name of the variable to print.

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

Prints a name-value pair to stdout.

Parameters:

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

5.14.2.20 EXTERNRT void rtxPrintReal (const char *name, OSREAL value)

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

Parameters:

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

5.14.2.21 EXTERNRT void rtxPrintTime (const char *name, const ONumDateTime *pvalue)

Prints a time value to stdout.

Parameters:

- name Name of the variable to print.
- pvalue Pointer to a structure that holds numeric DateTime value to print.
5.14.2.22 EXTERNRT void rtxPrintUInt64 (const char * name, OSUINT64 value)

Prints an unsigned 64-bit integer value to stdout.

Parameters:

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

5.14.2.23 EXTERNRT void rtxPrintUnicodeCharStr (const char * name, const OSUNICHAR * str, int nchars)

This function prints a Unicode string to standard output. Characters in the string that are within the normal Ascii range are printed as single characters. Characters outside the Ascii range are printed as 4-byte hex codes (0xnnnn).

Parameters:

- `name` The name of the variable to print.
- `str` Pointer to unicode string to be printed. String is an array of C unsigned short data variables.
- `nchars` Number of characters in the string. If value is negative, string is assumed to be null-terminated (i.e. ends with a 0x0000 character).

5.14.2.24 EXTERNRT void rtxPrintUnsigned (const char * name, OSUINT32 value)

Prints an unsigned integer value to stdout.

Parameters:

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

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

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

Parameters:

- `name` The name of the variable to print.
- `cstring` A pointer to the character string to be printed.
5.15 Print-To-Stream Functions

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

Functions

- EXTERNRT void rtxPrintToStreamBoolean (OSCTXT *pctxt, const char *name, OSBOOL value)
  
  Prints a boolean value to a print stream.

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

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

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

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

- EXTERNRT void rtxPrintToStreamInt64 (OSCTXT *pctxt, const char *name, OSINT64 value)
  
  Prints a 64-bit integer value to a print stream.

- EXTERNRT void rtxPrintToStreamUnsigned (OSCTXT *pctxt, const char *name, OSUINT32 value)
  
  Prints an unsigned integer value to a print stream.

- EXTERNRT void rtxPrintToStreamUInt64 (OSCTXT *pctxt, const char *name, OSUINT64 value)
  
  Prints an unsigned 64-bit integer value to a print stream.

- EXTERNRT void rtxPrintToStreamHexStr (OSCTXT *pctxt, const char *name, OSUINT32 numocts, const OSOCTET *data)
  
  This function prints the value of a binary string in hex format to standard output.

- EXTERNRT void rtxPrintToStreamHexBinary (OSCTXT *pctxt, const char *name, OSUINT32 numocts, const OSOCTET *data)
  
  Prints an octet string value in hex binary format to a print stream.

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

- EXTERNRT void rtxPrintToStreamUTF8CharStr (OSCTXT *pctxt, const char *name, const OSUTF8CHAR *cstring)
  
  Prints a UTF-8 encoded character string value to a print stream.

- EXTERNRT void rtxPrintToStreamUnicodeCharStr (OSCTXT *pctxt, const char *name, const OSUNICHAR *str, int nchars)
This function prints a Unicode string to standard output.

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

- EXTERNRT void rtxPrintToStreamNull (OSCTXT *pctxt, const char *name)
  *Prints a NULL value to a print stream.*

- EXTERNRT void rtxPrintToStreamNVP (OSCTXT *pctxt, const char *name, const OSUTF8NVP *value)
  *Prints a name-value pair to a print stream.*

- EXTERNRT int rtxPrintToStreamFile (OSCTXT *pctxt, const char *filename)
  *This function prints the contents of a text file to a print stream.*

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

- EXTERNRT void rtxPrintToStreamIncrIndent (void)
  *This function increments the current indentation level.*

- EXTERNRT void rtxPrintToStreamDecrIndent (void)
  *This function decrements the current indentation level.*

- EXTERNRT void rtxPrintToStreamCloseBrace (OSCTXT *pctxt)
  *This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.*

- EXTERNRT void rtxPrintToStreamOpenBrace (OSCTXT *pctxt, const char *)
  *This function opens a braced region by printing indent spaces, printing the name and opening brace, and increasing the indent level.*

- EXTERNRT void rtxHexDumpToStream (OSCTXT *pctxt, const OSOCTET *data, OSUINT32 numocts)
  *This function outputs a hexadecimal dump of the current buffer contents to a print stream.*

- EXTERNRT void rtxHexDumpToStreamEx (OSCTXT *pctxt, const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)
  *This function outputs a hexadecimal dump of the current buffer to a print stream, but it may output the dump as an array of bytes, words, or double words.*

### 5.15.1 Detailed Description

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

The output is redirected to the print stream defined within the context or to a global print stream. Print streams are set using the rtxSetPrintStream or rtxSetGlobalPrintStream function.
5.15.2 Function Documentation

5.15.2.1 EXTERNRT void rtxHexDumpToStream (OSCTXT * pctxt, const OSOCTET * data, OSUINT32 numocts)

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

Parameters:
- pctxt A pointer to a context structure.
- data The pointer to a buffer to be displayed.
- numocts The number of octets to be displayed.

5.15.2.2 EXTERNRT void rtxHexDumpToStreamEx (OSCTXT * pctxt, const OSOCTET * data, OSUINT32 numocts, int bytesPerUnit)

This function outputs a hexadecimal dump of the current buffer to a print stream, but it may output the dump as an array of bytes, words, or double words.

Parameters:
- pctxt A pointer to a context structure.
- data The pointer to a buffer to be displayed.
- numocts The number of octets to be displayed.
- bytesPerUnit The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

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

Prints a boolean value to a print stream.

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

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

Prints an ASCII character string value to a print stream.

Parameters:
- pctxt A pointer to a context structure.
- name The name of the variable to print.
- cstring A pointer to the character string to be printed.
5.15.2.5 EXTERNRT void rtxPrintToStreamDate (OSCTXT * pctxt, const char * name, const OSNumDateTime * pvalue)

Prints a date value to a print stream.

Parameters:

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

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

Prints a dateTime value to a print stream.

Parameters:

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

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

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

Parameters:

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

Returns:

  Status of operation, 0 if success.

5.15.2.8 EXTERNRT void rtxPrintToStreamHexBinary (OSCTXT * pctxt, const char * name, OSUINT32 numocts, const OSOCTET * data)

Prints an octet string value in hex binary format to a print stream.

Parameters:

  pctxt  A pointer to a context structure.
  name   The name of the variable to print.
  numocts The number of octets to be printed.
  data   A pointer to the data to be printed.
5.15.2.9 EXTERNRT void rtxPrintToStreamHexStr (OSCTXT * pctxt, const char * name, OSUINT32 numocts, const OSOCTET * data)

This function prints the value of a binary string in hex format to standard output. If the string is 32 bytes or less, it is printed on a single line with a ’0x’ prefix. If longer, a formatted hex dump showing both hex and ascii codes is done.

Parameters:

  pctxt A pointer to a context structure.
  name The name of the variable to print.
  numocts The number of octets to be printed.
  data A pointer to the data to be printed.

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

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

Parameters:

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

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

Prints an integer value to a print stream.

Parameters:

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

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

Prints a NULL value to a print stream.

Parameters:

  pctxt A pointer to a context structure.
  name The name of the variable to print.
5.15.2.13 EXTERNRT void rtxPrintToStreamNVP (OSCTXT * pctxt, const char * name, const OSUTF8NVP * value)

Prints a name-value pair to a print stream.

Parameters:
   pctxt A pointer to a context structure.
   name The name of the variable to print.
   value A pointer to name-value pair structure to print.

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

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

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

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

Prints a time value to a print stream.

Parameters:
   pctxt A pointer to a context structure.
   name Name of the variable to print.
   pvalue Pointer to a structure that holds numeric DateTime value to print.

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

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

Parameters:
   pctxt A pointer to a context structure.
   name The name of the variable to print.
   value Unsigned 64-bit integer value to print.
5.15.2.17 EXTERNRT void rtxPrintToStreamUnicodeCharStr (OSCTXT * pctxt, const char * name, const OSUNICHAR * str, int nchars)

This function prints a Unicode string to standard output. Characters in the string that are within the normal Ascii range are printed as single characters. Characters outside the Ascii range are printed as 4-byte hex codes (0xnnnn).

Parameters:
- pctxt  A pointer to a context structure.
- name   The name of the variable to print.
- str    Pointer to unicode string to be printed. String is an array of C unsigned short data variables.
- nchars Number of characters in the string. If value is negative, string is assumed to be null-terminated (i.e. ends with a 0x0000 character).

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

Prints an unsigned integer value to a print stream.

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

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

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

Parameters:
- pctxt  A pointer to a context structure.
- name   The name of the variable to print.
- cstring A pointer to the character string to be printed.
5.16 Floating-point number utility functions

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

Functions

- **EXTERNRT OSREAL rtxGetMinusInfinity (void)**
  Returns the IEEE negative infinity value.

- **EXTERNRT OSREAL rtxGetMinusZero (void)**
  Returns the IEEE minus zero value.

- **EXTERNRT OSREAL rtxGetNaN (void)**
  Returns the IEEE Not-A-Number (NaN) value.

- **EXTERNRT OSREAL rtxGetPlusInfinity (void)**
  Returns the IEEE positive infinity value.

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

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

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

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

5.16.1 Detailed Description

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

5.16.2 Function Documentation

5.16.2.1 **EXTERNRT OSREAL rtxGetMinusInfinity (void)**

Returns the IEEE negative infinity value.

This is defined as 0xfff0000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.
5.16.2.2 EXTERNRT OSREAL rtxGetMinusZero (void)

Returns the IEEE minus zero value.
This is defined as 0x8000000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

5.16.2.3 EXTERNRT OSREAL rtxGetNaN (void)

Returns the IEEE Not-A-Number (NaN) value.
This is defined as 0x7ff8000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

5.16.2.4 EXTERNRT OSREAL rtxGetPlusInfinity (void)

Returns the IEEE positive infinity value.
This is defined as 0x7ff0000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

5.16.2.5 EXTERNRT OSBOOL rtxIsMinusInfinity (OSREAL value)

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

Parameters:

   value  The input real value.

5.16.2.6 EXTERNRT OSBOOL rtxIsMinusZero (OSREAL value)

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

Parameters:

   value  The input real value.

5.16.2.7 EXTERNRT OSBOOL rtxIsNaN (OSREAL value)

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

Parameters:

   value  The input real value.

5.16.2.8 EXTERNRT OSBOOL rtxIsPlusInfinity (OSREAL value)

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

Parameters:

   value  The input real value.
5.17 Doubly-Linked List Utility Functions

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

Classes

• struct OSRTScalarDListNode
  
  This structure is used to hold a single data item within the list.

• struct OSRTScalarDList
  
  This is the main list structure.

Functions

• EXTERNRT void rtxScalarDListInit (OSRTScalarDList *pList)
  
  This function initializes a doubly linked list structure.

• EXTERNRT OSRTScalarDListNode * rtxScalarDListAppendDouble (struct OSCTX *pctxt, OSRTScalarDList *pList, OSDOUBLE value)
  
  This set of functions appends an item of the given scalar type to the linked list structure.

• EXTERNRT OSRTScalarDListNode * rtxScalarDListAppendNode (OSRTScalarDList *pList, OSRTScalarDListNode *pListNode)
  
  This function is used to append a node to the linked list.

• EXTERNRT OSRTScalarDListNode * rtxScalarDListInsertNode (OSRTScalarDList *pList, OSUINT32 index, OSRTScalarDListNode *pListNode)
  
  This function is used to insert a node into the linked list.

• EXTERNRT OSRTScalarDListNode * rtxScalarDListFindByIndex (const OSRTScalarDList *pList, OSUINT32 index)
  
  This function will return the node pointer of the indexed entry in the list.

• EXTERNRT void rtxScalarDListFreeNode (struct OSCTX *pctxt, OSRTScalarDList *pList, OSRTScalarDListNode *node)
  
  This function will remove the given node from the list and free memory.

• EXTERNRT void rtxScalarDListRemove (OSRTScalarDList *pList, OSRTScalarDListNode *node)
  
  This function will remove the given node from the list.

• EXTERNRT void rtxScalarDListFreeNodes (struct OSCTX *pctxt, OSRTScalarDList *pList)
  
  This function will free all of the dynamic memory used to hold the list node pointers.

5.17.1 Detailed Description

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

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

5.17.2.1 EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListAppendDouble (struct OSCTX ∗ pctxt, OSRTScalarDList ∗ pList, OSDOUBLE value)

This set of functions appends an item of the given scalar type to the linked list structure. Separate functions exist for all of the different supported scalar types.

Parameters:

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

Returns:

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

5.17.2.2 EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListAppendNode (OSRTScalarDList ∗ pList, OSRTScalarDListNode ∗ pListNode)

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

Parameters:

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

Returns:

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

5.17.2.3 EXTERNRT OSRTScalarDListNode ∗ rtxScalarDListFindByIndex (const OSRTScalarDList ∗ pList, OSUINT32 index)

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

Parameters:

- **pList** A pointer to a linked list structure.
- **index** Zero-based index into list where the specified item is located. If the list contains fewer items then the index, NULL is returned.

Returns:

A pointer to an allocated linked list node structure. To get the actual data item, the ident field must be examined to determine what type of value is stored in the union structure.
5.17.2.4 EXTERNRT void rtxScalarDListFreeNode (struct OSCTXT * pctxt, OSRTScalarDList * pList, OSRTScalarDListNode * node)

This function will remove the given node from the list and free memory.
It is assumed that memory for the list node structure was allocated using the rtxMemAlloc function.

Parameters:
- pctxt A pointer to a context structure.
- pList A pointer to a linked list structure.
- node Pointer to the list node to be removed.

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

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

Parameters:
- pctxt A pointer to a context structure.
- pList A pointer to a linked list structure.

5.17.2.6 EXTERNRT void rtxScalarDListInit (OSRTScalarDList * pList)

This function initializes a doubly linked list structure.
It sets the number of elements to zero and sets all internal pointer values to NULL. A doubly-linked scalar list structure
is described by the OSRTScalarDList type. Nodes of the list are of type OSRTScalarDListNode.

Parameters:
- pList A pointer to a linked list structure to be initialized.

5.17.2.7 EXTERNRT OSRTScalarDListNode * rtxScalarDListInsertNode (OSRTScalarDList * pList, OSUINT32 index, OSRTScalarDListNode * pListNode)

This function is used to insert a node into the linked list.

Parameters:
- pList A pointer to a linked list structure onto which the list node will be appended.
- index Zero-based index into list where the specified node is to be inserted.
- pListNode List node structure to be appended to the list. If this memory is to be released with the standard list
memory free function, then it must be allocated using the rtxMemAlloc function.

Returns:
A pointer to an allocated node structure used to link the given data value into the list. This is the node structure
that was passed in.
5.17.2.8  EXTERN RT void rtxScalarDLremove (OSRTScalarDLlist * pList, OSRTScalarDLListNode * node)

This function will remove the given node from the list.

Parameters:

  - `pList` A pointer to a linked list structure.
  - `node` Pointer to the list node to be removed.
5.18 TCP/IP or UDP socket utility functions

Typedefs

- typedef unsigned long OSIPADDR
  
  The IP address represented as unsigned long value.

Functions

- EXTERNRT int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET *pNewSocket, OSIPADDR *destAddr, int *destPort)
  
  This function permits an incoming connection attempt on a socket.

- EXTERNRT int rtxSocketAddrToStr (OSIPADDR ipAddr, char *pbuf, size_t bufsize)
  
  This function converts an IP address to its string representation.

- EXTERNRT int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)
  
  This function associates a local address with a socket.

- EXTERNRT int rtxSocketClose (OSRTSOCKET socket)
  
  This function closes an existing socket.

- EXTERNRT int rtxSocketConnect (OSRTSOCKET socket, const char *host, int port)
  
  This function establishes a connection to a specified socket.

- EXTERNRT int rtxSocketCreate (OSRTSOCKET *psocket)
  
  This function creates a TCP socket.

- EXTERNRT int rtxSocketGetHost (const char *host, struct in_addr *inaddr)
  
  This function resolves the given host name to an IP address.

- EXTERNRT int rtxSocketsInit ()
  
  This function initiates use of sockets by an application.

- EXTERNRT int rtxSocketListen (OSRTSOCKET socket, int maxConnection)
  
  This function places a socket a state where it is listening for an incoming connection.

- EXTERNRT int rtxSocketParseURL (char *url, char **protocol, char **address, int *port)
  
  This function parses a simple URL of the form <protocol>://<address>:<port> into its individual components.

- EXTERNRT int rtxSocketRecv (OSRTSOCKET socket, OSOCTET *pbuf, int bufsize)
  
  This function receives data from a connected socket.

- EXTERNRT int rtxSocketSend (OSRTSOCKET socket, const OSOCTET *pdata, int size)
  
  This function sends data on a connected socket.

- EXTERNRT int rtxSocketStrToAddr (const char *pIPAddrStr, OSIPADDR *pIPAddr)
  
  This function converts the string with IP address to a double word representation.
5.18.1 Typedef Documentation

5.18.1.1 typedef unsigned long OSIPADDR

The IP address represented as unsigned long value.
The most significant 8 bits in this unsigned long value represent the first number of the IP address. The least significant
8 bits represent the last number of the IP address.
Definition at line 57 of file rtxSocket.h.

5.18.2 Function Documentation

5.18.2.1 EXTERNRT int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET * pNewSocket, OSIPADDR * destAddr, int * destPort)

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

Parameters:

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

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

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

This function converts an IP address to its string representation.

Parameters:

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

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

5.18.2.3 EXTERNRT int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)

This function associates a local address with a socket.
It is used on an unconnected socket before subsequent calls to the rtxSocketConnect or rtxSocketListen functions. See
description of 'bind' socket function for further details.
Parameters:

socket The socket handle created by call to rtxSocketCreate function.
addr The local IP address to assign to the socket.
port The local port number to assign to the socket.

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

5.18.2.4 EXTERNRT int rtxSocketClose (OSRTSOCKET socket)

This function closes an existing socket.

Parameters:

socket The socket handle created by call to rtxSocketCreate or rtxSocketAccept function.

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

5.18.2.5 EXTERNRT int rtxSocketConnect (OSRTSOCKET socket, const char *host, int port)

This function establishes a connection to a specified socket.

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

Parameters:

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

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

5.18.2.6 EXTERNRT int rtxSocketCreate (OSRTSOCKET *psocket)

This function creates a TCP socket.

Parameters:

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

Returns:
Completion status of operation: 0 (0) = success, negative return value is error.
5.18.2.7 EXTERNRT int rtxSocketGetHost (const char * host, struct in_addr * inaddr)

This function resolves the given host name to an IP address.
The resulting address is stored in the given socket address structure.

Parameters:

host Host name to resolve
inaddr Socket address structure to receive resolved IP address

Returns:

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

5.18.2.8 EXTERNRT int rtxSocketListen (OSRTSOCKET socket, int maxConnection)

This function places a socket a state where it is listening for an incoming connection.
To accept connections, a socket is first created with the rtxSocketCreate function and bound to a local address with
the rtxSocketBind function, a maxConnection for incoming connections is specified with rtxSocketListen, and then
the connections are accepted with the rtxSocketAccept function. See description of 'listen' socket function for further
details.

Parameters:

socket The socket handle created by call to rtxSocketCreate function.
maxConnection Maximum length of the queue of pending connections.

Returns:

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

5.18.2.9 EXTERNRT int rtxSocketParseURL (char * url, char ** protocol, char ** address, int * port)

This function parses a simple URL of the form <protocol>://<address>[:<port>] into its individual components.
It is assumed that the buffer the URL is provided in is modifiable. Null-terminators are inserted in the buffer to delimit
the individual components. If the user needs to use the URL in unparsed form for any other purpose, they will need to
make a copy of it before calling this function.

Parameters:

url URL to be parsed. Buffer will be altered.
protocol Protocol string parsed from the URL.
address IP address or domain name parsed from URL.
port Optional port number. Zero if no port provided.

Returns:

Zero if parse successful or negative error code.
5.18.2.10 EXTERNRT int rtxSocketRecv (OSRTSOCKET socket, OSOCTET * pbuf, int bufsize)

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

Parameters:

- **socket** The socket handle created by call to rtxSocketCreate or rtxSocketAccept function.
- **pbuf** Pointer to the buffer for the incoming data.
- **bufsize** Length of the buffer.

Returns:

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

5.18.2.11 EXTERNRT int rtxSocketSend (OSRTSOCKET socket, const OSOCTET * pdata, int size)

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

Parameters:

- **socket** The socket handle created by call to rtxSocketCreate or rtxSocketAccept function.
- **pdata** Buffer containing the data to be transmitted.
- **size** Length of the data in pdata.

Returns:

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

5.18.2.12 EXTERNRT int rtxSocketsInit ()

This function initiates use of sockets by an application. This function must be called first before use sockets.

Returns:

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

5.18.2.13 EXTERNRT int rtxSocketStrToAddr (const char * pIPAddrStr, OSIPADDR * pIPAddr)

This function converts the string with IP address to a double word representation. The converted address may be used with the rtxSocketBind function.

Parameters:

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

**Returns:**

Completion status of operation: 0 (0) = success, negative return value is error.
5.19 Input/Output Data Stream Utility Functions

Stream functions are used for unbuffered stream operations.

Classes

- struct OSRTSTREAM
  
  The stream control block.

Typedefs

- typedef long( ∗ OSRTStreamReadProc )(struct OSRTSTREAM ∗pStream, OSOCTET ∗pbuffer, size_t bufSize)
  
  Stream read function pointer type.

- typedef long( ∗ OSRTStreamBlockingReadProc )(struct OSRTSTREAM ∗pStream, OSOCTET ∗pbuffer, size_t toReadBytes)
  
  Stream blockingRead function pointer type.

- typedef long( ∗ OSRTStreamWriteProc )(struct OSRTSTREAM ∗pStream, const OSOCTET ∗data, size_t numocts)
  
  Stream write function pointer type.

- typedef int( ∗ OSRTStreamFlushProc )(struct OSRTSTREAM ∗pStream)
  
  Stream flush function pointer type.

- typedef int( ∗ OSRTStreamCloseProc )(struct OSRTSTREAM ∗pStream)
  
  Stream close function pointer type.

- typedef int( ∗ OSRTStreamSkipProc )(struct OSRTSTREAM ∗pStream, size_t skipBytes)
  
  Stream skip function pointer type.

- typedef int( ∗ OSRTStreamMarkProc )(struct OSRTSTREAM ∗pStream, size_t readAheadLimit)
  
  Stream mark function pointer type.

- typedef int( ∗ OSRTStreamResetProc )(struct OSRTSTREAM ∗pStream)
  
  Stream reset function pointer type.

Functions

- EXTERNRT int rtxStreamClose (OSCTXT ∗pctxt)
  
  This function closes the input or output stream and releases any system resources associated with the stream.

- EXTERNRT int rtxStreamFlush (OSCTXT ∗pctxt)
  
  This function flushes the output stream and forces any buffered output octets to be written out.

- EXTERNRT int rtxStreamInit (OSCTXT ∗pctxt)
This function initializes a stream part of the context block.

- **EXTERNRT long rtxStreamRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t bufSize)**
  This function reads up to 'bufsize' bytes of data from the input stream into an array of octets.

- **EXTERNRT long rtxStreamBlockingRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t readBytes)**
  This function reads up to 'bufsize' bytes of data from the input stream into an array of octets.

- **EXTERNRT int rtxStreamSkip (OSCTXT *pctxt, size_t skipBytes)**
  This function skips over and discards the specified amount of data octets from this input stream.

- **EXTERNRT long rtxStreamWrite (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)**
  This function writes the specified amount of octets from the specified array to the output stream.

- **EXTERNRT int rtxStreamGetIOBytes (OSCTXT *pctxt, size_t *pPos)**
  This function returns the number of processed octets.

- **EXTERNRT int rtxStreamMark (OSCTXT *pctxt, size_t readAheadLimit)**
  Marks the current position in this input stream.

- **EXTERNRT int rtxStreamReset (OSCTXT *pctxt)**
  Repositions this stream to the position recorded by the last call to the rtxStreamMark function.

- **EXTERNRT OSBOOL rtxStreamMarkSupported (OSCTXT *pctxt)**
  Tests if this input stream supports the mark and reset methods.

- **EXTERNRT OSBOOL rtxStreamIsOpened (OSCTXT *pctxt)**
  Tests if this stream opened (for reading or writing).

- **EXTERNRT OSBOOL rtxStreamIsReadable (OSCTXT *pctxt)**
  Tests if this stream opened for reading.

- **EXTERNRT OSBOOL rtxStreamIsWritable (OSCTXT *pctxt)**
  Tests if this stream opened for writing.

- **EXTERNRT int rtxStreamRelease (OSCTXT *pctxt)**
  This function releases the stream’s resources.

- **EXTERNRT void rtxStreamSetCapture (OSCTXT *pctxt, OSRTMEMBUF *pmembuf)**
  This function sets a capture buffer for the stream.

- **EXTERNRT OSRTMEMBUF * rtxStreamGetCapture (OSCTXT *pctxt)**
  This function returns the capture buffer currently assigned to the stream.

### 5.19.1 Detailed Description

Stream functions are used for unbuffered stream operations.

All of the operations with streams are performed using a context block to maintain state information.
These functions may be used for any input/output operations with streams. Each stream should be initialized first by call to the\texttt{rtxStreamInit} function. After initialization, the stream may be opened for reading or writing by calling one of the following functions:

- \texttt{rtxStreamFileOpen}
- \texttt{rtxStreamFileAttach}
- \texttt{rtxStreamSocketAttach}
- \texttt{rtxStreamMemoryCreate}
- \texttt{rtxStreamMemoryAttach}

### 5.19.2 Typedef Documentation

#### 5.19.2.1 typedef long(\* OSRTStreamBlockingReadProc)(struct OSRTSTREAM \*pStream, OSOCTET \*pbuffer, size_t toReadBytes)

Stream blocking\texttt{Read} function pointer type.

A user may implement a customized \texttt{read} function for specific input streams. The \texttt{blockingRead} function is defined in the \texttt{OSRTSTREAM} control structure.

Definition at line 73 of file \texttt{rtxStream.h}.

#### 5.19.2.2 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 \texttt{OSRTSTREAM} control structure.

Definition at line 95 of file \texttt{rtxStream.h}.

#### 5.19.2.3 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 \texttt{OSRTSTREAM} control structure.

Definition at line 88 of file \texttt{rtxStream.h}.

#### 5.19.2.4 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 \texttt{OSRTSTREAM} control structure.

Definition at line 111 of file \texttt{rtxStream.h}.
5.19.2.5 typedef long (*OSRTStreamReadProc)(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t bufSize)

Stream read function pointer type.
A user may implement a customized read function for specific input streams. The read function is defined in the OSRTSTREAM control structure.
Definition at line 65 of file rtxStream.h.

5.19.2.6 typedef int (*OSRTStreamResetProc)(struct OSRTSTREAM *pStream)

Stream reset function pointer type.
A user may implement a customized function for a specific input stream type. The reset function is defined in the OSRTSTREAM control structure.
Definition at line 118 of file rtxStream.h.

5.19.2.7 typedef int (*OSRTStreamSkipProc)(struct OSRTSTREAM *pStream, size_t skipBytes)

Stream skip function pointer type.
A user may implement a customized function for a specific input stream type. The skip function is defined in the OSRTSTREAM control structure.
Definition at line 103 of file rtxStream.h.

5.19.2.8 typedef long (*OSRTStreamWriteProc)(struct OSRTSTREAM *pStream, const OSOCTET *data, size_t numocts)

Stream write function pointer type.
A user may implement a customized write function for any specific output streams. The write function is defined in the OSRTSTREAM control structure.
Definition at line 80 of file rtxStream.h.

5.19.3 Function Documentation

5.19.3.1 EXTERNRT long rtxStreamBlockingRead(OSCTXT *pctxt, OSOCTET *pbuffer, size_t readBytes)

This function reads up to 'bufsize' bytes of data from the input stream into an array of octets.
An attempt is made to read as many as bufsize octets, but a smaller number may be read, possibly zero. The number of octets actually read is returned as an integer. This function blocks until input data is available, end of file is detected, or another error is occurred.

Parameters:

- **pctxt** Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.
- **pbuffer** Pointer to a buffer to receive data.
- **readBytes** Number of bytes to read.
Returns:
The total number of octets read into the buffer, or negative value with error code if any error is occurred.

5.19.3.2 EXTERNRT int rtxStreamClose (OSCTXT * pctxt)
This function closes the input or output stream and releases any system resources associated with the stream.
For output streams this function also flushes all internal buffers to the stream.
Parameters:
  pctxt Pointer to a context structure variable which has been initialized for stream operations via a call to
  rtxStreamInit.

5.19.3.3 EXTERNRT int rtxStreamFlush (OSCTXT * pctxt)
This function flushes the output stream and forces any buffered output octets to be written out.
Parameters:
  pctxt Pointer to a context structure variable which has been initialized for stream operations via a call to
  rtxStreamInit.
Returns:
  Completion status of operation: 0 = success, negative return value is error.

5.19.3.4 EXTERNRT OSRTMEMBUF* rtxStreamGetCapture (OSCTXT * pctxt)
This function returns the capture buffer currently assigned to the stream.
Parameters:
  pctxt Pointer to a context structure variable that has been initialized for stream operations.
  pmembuf Pointer to an initialized memory buffer structure. This argument may be set to NULL to disable capture
  if previously set.
Returns:
  Pointer to memory buffer that was previously assigned as a capture buffer to the stream.

5.19.3.5 EXTERNRT int rtXStreamGetIOBytes (OSCTXT * pctxt, size_t * pPos)
This function returns the number of processed octets.
If the stream was opened as an input stream, then it returns the total number of read octets. If the stream was opened
as an output stream, then it returns the total number of written octets. Otherwise, this function returns an error code.
Parameters:
  pctxt Pointer to a context structure variable which has been initialized for stream operations via a call to
  rtxStreamInit.
**pPos**  Pointer to argument to receive total number of processed octets.

**Returns:**

The total number of processed octets or error code (negative value).

### 5.19.3.6 EXTERNRT int rtxStreamInit (OSCTXT * pctxt)

This function initializes a stream part of the context block.
This function should be called first before any operation with a stream.

**Parameters:**

- **pctxt**  Pointer to context structure variable, for which stream to be initialized.

**Returns:**

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

### 5.19.3.7 EXTERNRT OSBOOL rtxStreamIsOpened (OSCTXT * pctxt)

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

**Parameters:**

- **pctxt**  Pointer to a context structure variable that has been initialized for stream operations.

**Returns:**

TRUE if this stream is opened for reading or writing; FALSE otherwise.

### 5.19.3.8 EXTERNRT OSBOOL rtxStreamIsReadable (OSCTXT * pctxt)

Tests if this stream opened for reading.

**Parameters:**

- **pctxt**  Pointer to a context structure variable that has been initialized for stream operations.

**Returns:**

TRUE if this stream is opened for reading; FALSE otherwise.

### 5.19.3.9 EXTERNRT OSBOOL rtxStreamIsWritable (OSCTXT * pctxt)

Tests if this stream opened for writing.

**Parameters:**

- **pctxt**  Pointer to a context structure variable that has been initialized for stream operations.

**Returns:**

TRUE if this stream is opened for writing; FALSE otherwise.
5.19.3.10 EXTERNRT int rtxStreamMark (OSCTXT * pctxt, size_t readAheadLimit)

Marks the current position in this input stream.

A subsequent call to the rtxStreamReset function repositions this stream at the last marked position so that subsequent reads re-read the same bytes. The readAheadLimit argument tells this input stream to allow many bytes to be read before the mark position gets invalidated.

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.
readAheadLimit The maximum limit of bytes that can be read before the mark position becomes invalid.

Returns:

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

5.19.3.11 EXTERNRT OSBOOL rtxStreamMarkSupported (OSCTXT * pctxt)

Tests if this input stream supports the mark and reset methods.

Whether or not mark and reset are supported is an invariant property of a particular input stream instance. By default, it returns FALSE.

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.

Returns:

TRUE if this stream instance supports the mark and reset methods; FALSE otherwise.

5.19.3.12 EXTERNRT long rtxStreamRead (OSCTXT * pctxt, OSOCTET * pbuffer, size_t bufSize)

This function reads up to 'bufsize' bytes of data from the input stream into an array of octets.

An attempt is made to read as many as bufsize octets, but a smaller number may be read, possibly zero. The number of octets actually read is returned as an integer. This functions blocks until input data is available, end of file is detected, or another error is occurred.

Parameters:

pctxt Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.
pbuffer Pointer to a buffer to receive data.
bufSize Size of the buffer.

Returns:

The total number of octets read into the buffer, or negative value with error code if any error is occurred.
5.19.3.13 EXTERNRT int rtxStreamRelease (OSCTXT *pctxt)

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

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.

Returns:

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

5.19.3.14 EXTERNRT int rtxStreamReset (OSCTXT *pctxt)

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

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.

Returns:

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

5.19.3.15 EXTERNRT void rtxStreamSetCapture (OSCTXT *pctxt, OSRTMEMBUF *pmembuf)

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

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.

pmembuf Pointer to an initialized memory buffer structure. This argument may be set to NULL to disable capture
if previously set.

5.19.3.16 EXTERNRT int rtxStreamSkip (OSCTXT *pctxt, size_t skipBytes)

This function skips over and discards the specified amount of data octets from this input stream.

Parameters:

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

skipBytes The number of octets to be skipped.

Returns:

Completion status of operation: 0 = success, negative return value is error.
5.19.3.17  EXTERNRT long rtxStreamWrite (OSCTXT * pctxt, const OSOCTET * data, size_t numocts)

This function writes the specified amount of octets from the specified array to the output stream.

Parameters:

  pctxt  Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.
  data  The pointer to data to be written.
  numocts  The number of octets to write.

Returns:

  Completion status of operation: 0 = success, negative return value is error.
5.20 File stream functions.

File stream functions are used for stream operations with files.

Functions

- EXTERNRT int rtxStreamFileAttach (OSCTXT *pctxt, FILE *pFile, OSUINT16 flags)
  
  Attaches the existing file structure pointer to the stream.

- EXTERNRT int rtxStreamFileOpen (OSCTXT *pctxt, const char *pFilename, OSUINT16 flags)
  
  Opens a file stream.

- EXTERNRT int rtxStreamFileCreateReader (OSCTXT *pctxt, const char *pFilename)
  
  This function creates an input file stream using the specified file name.

- EXTERNRT int rtxStreamFileCreateWriter (OSCTXT *pctxt, const char *pFilename)
  
  This function creates an output file stream using the file name.

5.20.1 Detailed Description

File stream functions are used for stream operations with files.

5.20.2 Function Documentation

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

Attaches the existing file structure pointer to the stream.

The file should be already opened either for the reading or writing. The ‘flags’ parameter specifies the access mode for the stream - input or output.

Parameters:

- pctxt Pointer to a context structure variable that has been initialized for stream operations.
- pFile Pointer to FILE structure. File should be already opened either for the writing or reading.
- flags Specifies the access mode for the stream:
  - OSRTSTRMF_INPUT = input (reading) stream;
  - OSRTSTRMF_OUTPUT = output (writing) stream.

Returns:

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

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

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

- `pctxt` Pointer to a context structure variable that has been initialized for stream operations.
- `pFilename` Pointer to null-terminated string that contains the name of file.

Returns:

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

5.20.2.3 EXTERNRT int rtxStreamFileCreateWriter (OSCTXT * pctxt, const char * pFilename)

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

Parameters:

- `pctxt` Pointer to a context structure variable that has been initialized for stream operations.
- `pFilename` Pointer to null-terminated string that contains the name of file.

Returns:

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

5.20.2.4 EXTERNRT int rtxStreamFileOpen (OSCTXT * pctxt, const char * pFilename, OSUINT16 flags)

Opens a file stream.
The ‘flags’ parameter specifies the access mode for the stream - input or output.

Parameters:

- `pctxt` Pointer to a context structure variable that has been initialized for stream operations.
- `pFilename` Pointer to null-terminated string that contains the name of file.
- `flags` Specifies the access mode for the stream:
  - OSRTSTRMF_INPUT = input (reading) stream;
  - OSRTSTRMF_OUTPUT = output (writing) stream.

Returns:

Completion status of operation: 0 = success, negative return value is error.
5.21 Memory stream functions.

Memory stream functions are used for memory stream operations.

Functions

- EXTERNRT int rtxStreamMemoryCreate (OSCTXT *pctxt, OSUINT16 flags)
  
  Opens a memory stream.

- EXTERNRT int rtxStreamMemoryAttach (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize, OSUINT16 flags)
  
  Opens a memory stream using the specified memory buffer.

- EXTERNRT OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT *pctxt, size_t *pSize)
  
  This function returns the memory buffer and its size for the given memory stream.

- EXTERNRT int rtxStreamMemoryCreateReader (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
  
  This function creates an input memory stream using the specified buffer.

- EXTERNRT int rtxStreamMemoryCreateWriter (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
  
  This function creates an output memory stream using the specified buffer.

5.21.1 Detailed Description

Memory stream functions are used for memory stream operations.

5.21.2 Function Documentation

5.21.2.1 EXTERNRT int rtxStreamMemoryAttach (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize, OSUINT16 flags)

Opens a memory stream using the specified memory buffer.

The 'flags' parameter specifies the access mode for the stream - input or output.

Parameters:

- **pctxt** Pointer to a context structure variable that has been initialized for stream operations.
- **pMemBuf** The pointer to the buffer.
- **bufSize** The size of the buffer.
- **flags** Specifies the access mode for the stream:
  - OSRTSTRMF_INPUT = input (reading) stream;
  - OSRTSTRMF_OUTPUT = output (writing) stream.

Returns:

Completion status of operation: 0 = success, negative return value is error.
5.21.2.2 EXTERNRT int rtxStreamMemoryCreate (OSCTXT *pctxt, OSUINT16 flags)

Opens a memory stream.

A memory buffer will be created by this function. The 'flags' parameter specifies the access mode for the stream - input or output.

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.

flags Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

Returns:

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

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

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

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.
pMemBuf The pointer to the buffer
bufSize The size of the buffer

Returns:

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

5.21.2.4 EXTERNRT int rtxStreamMemoryCreateWriter (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)

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

If pMemBuf or bufSize is NULL then new buffer will be allocated.

Parameters:

pctxt Pointer to a context structure variable that has been initialized for stream operations.
pMemBuf The pointer to the buffer. Can be NULL - new buffer will be allocated in this case.
bufSize The size of the buffer. Can be 0 - new buffer will be allocated in this case.

Returns:

Completion status of operation: 0 = success, negative return value is error.
5.21.2.5 EXTERN RT OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT * pctxt, size_t * pSize)

This function returns the memory buffer and its size for the given memory stream.

Parameters:

- *pctxt* Pointer to a context structure variable that has been initialized for stream operations.
- *pSize* The pointer to size_t to receive the size of buffer.

Returns:

The pointer to memory buffer. NULL, if error occurred.
5.22 Socket stream functions.

Socket stream functions are used for socket stream operations.

Functions

- EXTERNRT int rtxStreamSocketAttach (OSCTXT *pctxt, OSRTSOCKET socket, OSUINT16 flags)
  
  Attaches the existing socket handle to the stream.

- EXTERNRT int rtxStreamSocketClose (OSCTXT *pctxt)
  
  This function closes a socket stream.

- EXTERNRT int rtxStreamSocketCreateWriter (OSCTXT *pctxt, const char *host, int port)
  
  This function opens a socket stream for writing.

- EXTERNRT int rtxStreamSocketSetOwnership (OSCTXT *pctxt, OSBOOL ownSocket)
  
  This function transfers ownership of the socket to or from the stream instance.

5.22.1 Detailed Description

Socket stream functions are used for socket stream operations.

5.22.2 Function Documentation

5.22.2.1 EXTERNRT int rtxStreamSocketAttach (OSCTXT *pctxt, OSRTSOCKET socket, OSUINT16 flags)

Attaches the existing socket handle to the stream.

The socket should be already opened and connected. The 'flags' parameter specifies the access mode for the stream - input or output.

Parameters:

- *pctxt Pointer to a context structure variable that has been initialized for stream operations.
- *socket The socket handle created by rtxSocketCreate.
- *flags Specifies the access mode for the stream:
  
  • OSRTSTRMF_INPUT = input (reading) stream;
  • OSRTSTRMF_OUTPUT = output (writing) stream.

Returns:

Completion status of operation: 0 = success, negative return value is error.
5.22.2.2  EXTERNRT int rtxStreamSocketClose (OSCTXT * pctxt)

This function closes a socket stream.

Parameters:

  pctxt  Pointer to a context structure variable that has been initialized for stream operations.

Returns:

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

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

This function opens a socket stream for writing.

Parameters:

  pctxt  Pointer to a context structure variable that has been initialized for stream operations.
  host   Name of host or IP address to which to connect.
  port   Port number to which to connect.

Returns:

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

5.22.2.4  EXTERNRT int rtxStreamSocketSetOwnership (OSCTXT * pctxt, OSBOOL ownSocket)

This function transfers ownership of the socket to or from the stream instance.

The socket will be closed and deleted when the stream is closed or goes out of scope. By default stream socket owns the socket.

Parameters:

  pctxt  Pointer to a context structure variable that has been initialized for stream operations.
  ownSocket  Boolean value.
5.23 UTF-8 String Functions

The UTF-8 string functions handle string operations on UTF-8 encoded strings.

Defines

- #define RTUTF8STRCMP(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)
  Compare UTF-8 string to a string literal.

Functions

- EXTERNRT long rtxUTF8ToUnicode (OSCTX *pctxt, const OSUTF8CHAR *inbuf, OSUNICHAR *outbuf, size_t outbufsz)
  This function converts a UTF-8 string to a Unicode string (UTF-16).

- EXTERNRT int rtxValidateUTF8 (OSCTX *pctxt, const OSUTF8CHAR *inbuf)
  This function will validate a UTF-8 encoded string to ensure that it is encoded correctly.

- EXTERNRT size_t rtxUTF8Len (const OSUTF8CHAR *inbuf)
  This function will return the length (in characters) of a null-terminated UTF-8 encoded string.

- EXTERNRT size_t rtxUTF8LenBytes (const OSUTF8CHAR *inbuf)
  This function will return the length (in bytes) of a null-terminated UTF-8 encoded string.

- EXTERNRT int rtxUTF8CharSize (OS32BITCHAR wc)
  This function will return the number of bytes needed to encode the given 32-bit universal character value as a UTF-8 character.

- EXTERNRT int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET *buf, size_t bufsiz)
  This function will convert a wide character into an encoded UTF-8 character byte string.

- EXTERNRT int rtxUTF8DecodeChar (OSCTX *pctxt, const OSUTF8CHAR *pinbuf, int *pInsize)
  This function will convert an encoded UTF-8 character byte string into a wide character value.

- EXTERNRT OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR *buf, OSUINT32 *len)
  This function will convert a UTF-8 encoded character value into a wide character.

- EXTERNRT OSUTF8CHAR * rtxUTF8StrChr (OSUTF8CHAR *utf8str, OS32BITCHAR utf8char)
  This function finds a character in the given UTF-8 character string.

- EXTERNRT OSUTF8CHAR * rtxUTF8Strdup (OSCTX *pctxt, const OSUTF8CHAR *utf8str)
  This function creates a duplicate copy of the given UTF-8 character string.

- EXTERNRT OSUTF8CHAR * rtxUTF8Strndup (OSCTX *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes)
  This function creates a duplicate copy of the given UTF-8 character string.

- EXTERNRT OSUTF8CHAR * rtxUTF8StrRefOrDup (OSCTX *pctxt, const OSUTF8CHAR *utf8str)
This function check to see if the given UTF8 string pointer exists on the memory heap.

• EXTERNRT OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
  This function compares two UTF-8 string values for equality.

• EXTERNRT OSBOOL rtxUTF8StrmEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
  This function compares two UTF-8 string values for equality.

• EXTERNRT int rtxUTF8Strcmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
  This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than).

• EXTERNRT int rtxUTF8Strncmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
  This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than).

• EXTERNRT OSUTF8CHAR * rtxUTF8Strcpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src)
  This function copies a null-terminated UTF-8 string to a target buffer.

• EXTERNRT OSUTF8CHAR * rtxUTF8Strncpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src, size_t nchars)
  This function copies the given number of characters from a UTF-8 string to a target buffer.

• EXTERNRT OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR *str)
  This function computes a hash code for the given string value.

• EXTERNRT const OSUTF8CHAR * rtxUTF8StrJoin (OSCTXT *pctxt, const OSUTF8CHAR *str1, const OSUTF8CHAR *str2, const OSUTF8CHAR *str3, const OSUTF8CHAR *str4, const OSUTF8CHAR *str5)
  This function concatenates up to five substrings together into a single string.

• EXTERNRT int rtxUTF8StrToBool (const OSUTF8CHAR *utf8str, OSBOOL *pvalue)
  This function converts the given null-terminated UTF-8 string to a boolean (true/false) value.

• EXTERNRT int rtxUTF8StrnToBool (const OSUTF8CHAR *utf8str, size_t nbytes, OSBOOL *pvalue)
  This function converts the given part of UTF-8 string to a boolean (true/false) value.

• EXTERNRT int rtxUTF8StrToDouble (const OSUTF8CHAR *utf8str, OSREAL *pvalue)
  This function converts the given null-terminated UTF-8 string to a floating point (C/C++ double) value.

• EXTERNRT int rtxUTF8StrnToDouble (const OSUTF8CHAR *utf8str, size_t nbytes, OSREAL *pvalue)
  This function converts the given part of UTF-8 string to a double value.

• EXTERNRT int rtxUTF8StrToInt (const OSUTF8CHAR *utf8str, OSINT32 *pvalue)
  This function converts the given null-terminated UTF-8 string to an integer value.

• EXTERNRT int rtxUTF8StrnToInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT32 *pvalue)
  This function converts the given part of UTF-8 string to an integer value.

• EXTERNRT int rtxUTF8StrToUInt (const OSUTF8CHAR *utf8str, OSUINT32 *pvalue)
  This function converts the given null-terminated UTF-8 string to an unsigned integer value.
• EXTERNRT int rtxUTF8StrnToUInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT32 *pvalue)
  This function converts the given part of UTF-8 string to an unsigned integer value.

• EXTERNRT int rtxUTF8StrToInt64 (const OSUTF8CHAR *utf8str, OSINT64 *pvalue)
  This function converts the given null-terminated UTF-8 string to a 64-bit integer value.

• EXTERNRT int rtxUTF8StrnToInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT64 *pvalue)
  This function converts the given part of UTF-8 string to a 64-bit integer value.

• EXTERNRT int rtxUTF8StrToUInt64 (const OSUTF8CHAR *utf8str, OSUINT64 *pvalue)
  This function converts the given null-terminated UTF-8 string to an unsigned 64-bit integer value.

• EXTERNRT int rtxUTF8StrnToUInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT64 *pvalue)
  This function converts the given part of UTF-8 string to an unsigned 64-bit integer value.

• EXTERNRT int rtxUTF8ToDynUniStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSUNICHAR **ppdata, OSUINT32 *pnchars)
  This function converts the given UTF-8 string to a Unicode string.

• EXTERNRT int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR *utf8instr, size_t nbytes, const OSUTF8CHAR **putf8outstr)
  This function removes leading and trailing whitespace from a string.

• EXTERNRT int rtxUTF8StrToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, OSDynOctStr *pvalue)
  This function converts the given null-terminated UTF-8 string to a octet string value.

• EXTERNRT int rtxUTF8StrnToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, OSDynOctStr *pvalue)
  This function converts the given part of UTF-8 string to a octet string value.

• EXTERNRT int rtxUTF8StrToNamedBits (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSBitMapItem *pBitMap, OSOCTET *pvalue, OSUINT32 *pnbits, OSUINT32 bufsize)
  This function converts the given null-terminated UTF-8 string to named bit items.

• EXTERNRT const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR *utf8str, OSUTF8CHAR **ppNext)
  This function returns the next whitespace-separated token from the input string.

5.23.1 Detailed Description

The UTF-8 string functions handle string operations on UTF-8 encoded strings.

This is the default character string data type used for encoded XML data. UTF-8 strings are represented in C as strings of unsigned characters (bytes) to cover the full range of possible single character encodings.
5.23.2 Define Documentation

5.23.2.1 #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)

Compare UTF-8 string to a string literal.

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

Definition at line 506 of file rtxUTF8.h.

5.23.3 Function Documentation

5.23.3.1 EXTERNRT int rtxUTF8CharSize (OS32BITCHAR wc)

This function will return the number of bytes needed to encode the given 32-bit universal character value as a UTF-8 character.

Parameters:
wc 32-bit wide character value.

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

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

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

Parameters:
buf Pointer to UTF-8 character value.
len Pointer to integer to receive decoded size (in bytes) of the UTF-8 character value sequence.

Returns:
Converted wide character value.

5.23.3.3 EXTERNRT int rtxUTF8DecodeChar (OSCTXT *pctxt, const OSUTF8CHAR *pinbuf, int *pInsize)

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

Parameters:
pctxt A pointer to a context structure.
pinbuf Pointer to UTF-8 byte sequence to be decoded.
pInsize Number of bytes that were consumed (i.e. size of the character).

Returns:
32-bit wide character value.
5.23.3.4  EXTERNRT int rtxUTF8EncodeChar (OS32BITCHAR *wc, OSOCTET * buf, size_t bufsiz)

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

**Parameters:**
- wc 32-bit wide character value.
- buf Buffer to receive encoded UTF-8 character value.
- bufsiz Size of the buffer to receive the encoded value.

**Returns:**
- Number of bytes consumed to encode character or negative status code if error.

5.23.3.5  EXTERNRT size_t rtxUTF8Len (const OSUTF8CHAR *inbuf)

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

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

**Returns:**
- Number of characters in string. Note that this may be different than the number of bytes as UTF-8 characters can span multiple-bytes.

5.23.3.6  EXTERNRT size_t rtxUTF8LenBytes (const OSUTF8CHAR *inbuf)

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

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

**Returns:**
- Number of bytes in the string.

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

This function removes leading and trailing whitespace from a string.

**Parameters:**
- utf8instr Input UTF-8 string from which to removed whitespace.
- nbytes Size in bytes of utf8instr.
- putf8outstr Pointer to receive result string.

**Returns:**
- Positive value = length of result string, negative value = error code.
This function finds a character in the given UTF-8 character string.
It is similar to the C `strchr` function.

**Parameters:**
- `utf8str` Null-terminated UTF-8 string to be searched.
- `utf8char` 32-bit Unicode character to find.

**Returns:**
Pointer to the first occurrence of character in string, or NULL if character is not found.

This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than).
It is similar to the C `strcmp` function.

**Parameters:**
- `utf8str1` UTF-8 string to be compared.
- `utf8str2` UTF-8 string to be compared.

**Returns:**
-1 if `utf8str1` is less than `utf8str2`, 0 if the two string are equal, and +1 if the `utf8str1` is greater than `utf8str2`.

This function copies a null-terminated UTF-8 string to a target buffer.
It is similar to the C `strcpy` function except more secure because it checks for buffer overrun.

**Parameters:**
- `dest` Pointer to destination buffer to receive string.
- `bufsiz` Size of the destination buffer.
- `src` Pointer to null-terminated string to copy.

**Returns:**
Pointer to destination buffer or NULL if copy failed.

This function creates a duplicate copy of the given UTF-8 character string.
It is similar to the C `strdup` function. Memory for the duplicated string is allocated using the `rtxMemAlloc` function.
Parameters:

\- \textit{pctxt}  A pointer to a context block structure.
\- \textit{utf8str} Null-terminated UTF-8 string to be duplicated.

Returns:

Pointer to duplicated string value.

5.23.3.12 \textbf{EXTERNRT OSBOOL rtxUTF8StrEqual} (\textit{const OSUTF8CHAR}∗ \textit{utf8str1}, \textit{const OSUTF8CHAR}∗ \textit{utf8str2})

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

Parameters:

\- \textit{utf8str1} UTF-8 string to be compared.
\- \textit{utf8str2} UTF-8 string to be compared.

Returns:

TRUE if equal, FALSE if not.

5.23.3.13 \textbf{EXTERNRT OSUINT32 rtxUTF8StrHash} (\textit{const OSUTF8CHAR}∗ \textit{str})

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

Parameters:

\- \textit{str} Pointer to string.

Returns:

Hash code value.

5.23.3.14 \textbf{EXTERNRT const OSUTF8CHAR∗ rtxUTF8StrJoin} (\textit{OSCTX}∗ \textit{pctxt}, \textit{const OSUTF8CHAR}∗ \textit{str1}, \textit{const OSUTF8CHAR}∗ \textit{str2}, \textit{const OSUTF8CHAR}∗ \textit{str3}, \textit{const OSUTF8CHAR}∗ \textit{str4}, \textit{const OSUTF8CHAR}∗ \textit{str5})

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

Parameters:

\- \textit{pctxt} Pointer to a context block structure.
\- \textit{str1} Pointer to substring to join.
\- \textit{str2} Pointer to substring to join.
\- \textit{str3} Pointer to substring to join.
\- \textit{str4} Pointer to substring to join.
\- \textit{str5} Pointer to substring to join.

Returns:

Composite string consisting of all parts. Memory is allocated for this string using rtxMemAlloc and must be freed using either rtxMemFreePtr or rtxMemFree. If memory allocation for the string fails, NULL is returned.
5.23.3.15 EXTERNRT int rtxUTF8Strncmp (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2, size_t count)

This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than).
In this case, a maximum count of the number of bytes to compare can be specified. It is similar to the C `strncmp` function.

Parameters:
- `utf8str1` UTF-8 string to be compared.
- `utf8str2` UTF-8 string to be compared.
- `count` Number of bytes to compare.

Returns:
- -1 if utf8str1 is less than utf8str2, 0 if the two string are equal, and +1 if the utf8str1 is greater than utf8str2.

5.23.3.16 EXTERNRT OSUTF8CHAR* rtxUTF8Strncpy (OSUTF8CHAR * dest, size_t bufsiz, const OSUTF8CHAR * src, size_t nchars)

This function copies the given number of characters from a UTF-8 string to a target buffer.
It is similar to the C `strncpy` function except more secure because it checks for buffer overrun and ensures a null-terminator is copied to the end of the target buffer.

Parameters:
- `dest` Pointer to destination buffer to receive string.
- `bufsiz` Size of the destination buffer.
- `src` Pointer to null-terminated string to copy.
- `nchars` Number of characters to copy.

Returns:
Pointer to destination buffer or NULL if copy failed.

5.23.3.17 EXTERNRT OSUTF8CHAR* rtxUTF8Strndup (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, size_t nbytes)

This function creates a duplicate copy of the given UTF-8 character string.
It is similar to the `rtxUTF8Strdup` function except that it allows the number of bytes to convert to be specified.
Memory for the duplicated string is allocated using the `rtxMemAlloc` function.

Parameters:
- `pctxt` A pointer to a context structure.
- `utf8str` UTF-8 string to be duplicated.
- `nbytes` Number of bytes from `utf8str` to duplicate.

Returns:
Pointer to duplicated string value.
5.23.3.18 EXTERNRT OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR * utf8str1, const OSUTF8CHAR * utf8str2, size_t count)

This function compares two UTF-8 string values for equality. It is similar to the rtxUTF8StrEqual function except that it allows the number of bytes to compare to be specified.

Parameters:

- utf8str1 UTF-8 string to be compared.
- utf8str2 UTF-8 string to be compared.
- count Number of bytes to compare.

Returns:

TRUE if equal, FALSE if not.

5.23.3.19 EXTERNRT const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR * utf8str, OSUTF8CHAR ** ppNext)

This function returns the next whitespace-separated token from the input string. It also returns a pointer to the first non-whitespace character after the parsed token. Note that the input string is altered in the operation as null-terminators are inserted to mark the token boundaries.

Parameters:

- utf8str Null-terminated UTF-8 string to parse. This string will be altered. Use rtxUTF8Strdup to make a copy of original string before calling this function if the original string cannot be altered.
- ppNext Pointer to receive next location in string after parsed token. This can be used as input to get the next token. If NULL returned, all tokens in in the string have been parsed.

Returns:

Pointer to next parsed token. NULL if no more tokens.

5.23.3.20 EXTERNRT int rtxUTF8StrnToBool (const OSUTF8CHAR * utf8str, size_t nbytes, OSBOOL * pvalue)

This function converts the given part of UTF-8 string to a boolean (true/false) value. It is assumed the string contains only the tokens 'true', 'false', ‘1’, or ‘0’.

Parameters:

- utf8str Null-terminated UTF-8 string to convert
- nbytes Size in bytes of utf8Str.
- pvalue Pointer to boolean value to receive result

Returns:

Status: 0 = OK, negative value = error
5.23.3.21  EXTERNRT int rtxUTF8StrnToDouble (const OSUTF8CHAR *utf8str, size_t nbytes, OSREAL *pvalue)

This function converts the given part of UTF-8 string to a double value.
It is assumed the string contains only numeric digits, whitespace, and other special floating point characters. It is similar to the C atof function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

Parameters:
- `utf8str`  UTF-8 string to convert. Not necessary to be null-terminated.
- `nbytes`  Size in bytes of utf8Str.
- `pvalue`  Pointer to double to receive result

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

5.23.3.22  EXTERNRT int rtxUTF8StrnToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, OSDynOctStr *pvalue)

This function converts the given part of UTF-8 string to a octet string value.
The string consists of a series of hex digits. This is the dynamic version in which memory is allocated for the returned octet string variable.

Parameters:
- `pctxt`  Pointer to context block structure.
- `utf8str`  Null-terminated UTF-8 string to convert
- `nbytes`  Size in bytes of utf8Str.
- `pvalue`  Pointer to a variable to receive the decoded octet string value.

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

5.23.3.23  EXTERNRT int rtxUTF8StrnToInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT32 *pvalue)

This function converts the given part of UTF-8 string to an integer value.
It is assumed the string contains only numeric digits and whitespace. It is similar to the C atoi function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

Parameters:
- `utf8str`  UTF-8 string to convert. Not necessary to be null-terminated.
\textbf{nbytes} Size in bytes of utf8Str.

\textbf{pvalue} Pointer to integer to receive result

\textbf{Returns:}
Status: 0 = OK, negative value = error

\textbf{5.23.3.24} \textbf{EXTERNRT int rtxUTF8StrnToInt64 (const OSUTF8CHAR} * \textit{utf8str}, \textit{size_t nbytes}, \textit{OSINT64} * \textit{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.

\textbf{Parameters:}
- \textit{utf8str} UTF-8 string to convert. Not necessary to be null-terminated.
- \textit{nbytes} Size in bytes of utf8Str.
- \textit{pvalue} Pointer to integer to receive result

\textbf{Returns:}
Status: 0 = OK, negative value = error

\textbf{5.23.3.25} \textbf{EXTERNRT int rtxUTF8StrnToUInt (const OSUTF8CHAR} * \textit{utf8str}, \textit{size_t nbytes}, \textit{OSUINT32} * \textit{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.

\textbf{Parameters:}
- \textit{utf8str} UTF-8 string to convert. Not necessary to be null-terminated.
- \textit{nbytes} Size in bytes of utf8Str.
- \textit{pvalue} Pointer to integer to receive result

\textbf{Returns:}
Status: 0 = OK, negative value = error

\textbf{5.23.3.26} \textbf{EXTERNRT int rtxUTF8StrnToUInt64 (const OSUTF8CHAR} * \textit{utf8str}, \textit{size_t nbytes}, \textit{OSUINT64} * \textit{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.

\textbf{Parameters:}
- \textit{utf8str} UTF-8 string to convert. Not necessary to be null-terminated.
- \textit{nbytes} Size in bytes of utf8Str.
**pvalue** Pointer to integer to receive result

**Returns:**

Status: 0 = OK, negative value = error

### 5.23.3.27 EXTERNRT OSUTF8CHAR\* rtxUTF8StrRefOrDup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str)

This function checks to see if the given UTF8 string pointer exists on the memory heap. If it does, its reference count is incremented; otherwise, a duplicate copy is made.

**Parameters:**

- **pctxt** A pointer to a context structure.
- **utf8str** Null-terminated UTF-8 string variable.

**Returns:**

Pointer to string value. This will either be the existing UTF-8 string pointer value (utf8str) or a new value.

### 5.23.3.28 EXTERNRT int rtxUTF8StrToBool (const OSUTF8CHAR *utf8str, OSBOOL *pvalue)

This function converts the given null-terminated UTF-8 string to a boolean (true/false) value. It is assumed the string contains only the tokens 'true', 'false', '1', or '0'.

**Parameters:**

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

**Returns:**

Status: 0 = OK, negative value = error

### 5.23.3.29 EXTERNRT int rtxUTF8StrToDouble (const OSUTF8CHAR *utf8str, OSREAL *pvalue)

This function converts the given null-terminated UTF-8 string to a floating point (C/C++ double) value. It is assumed the string contains only numeric digits, special floating point characters (+,-,.E..), and whitespace. It is similar to the C atof function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

**Parameters:**

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

**Returns:**

Status: 0 = OK, negative value = error
5.23.3.30  EXTERNRT int rtxUTF8StrToDynHexStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, OSDynOctStr * pvalue)

This function converts the given null-terminated UTF-8 string to a octet string value.
The string consists of a series of hex digits. This is the dynamic version in which memory is allocated for the returned octet string variable.

Parameters:
- pctxt  Pointer to context block structure.
- utf8str Null-terminated UTF-8 string to convert
- pvalue Pointer to a variable to receive the decoded octet string value.

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

5.23.3.31  EXTERNRT int rtxUTF8StrToInt (const OSUTF8CHAR * utf8str, OSINT32 * pvalue)

This function converts the given null-terminated UTF-8 string to an integer value.
It is assumed the string contains only numeric digits and whitespace. It is similar to the C atoi function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

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

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

5.23.3.32  EXTERNRT int rtxUTF8StrToInt64 (const OSUTF8CHAR * utf8str, OSINT64 * pvalue)

This function converts the given null-terminated UTF-8 string to a 64-bit integer value.
It is assumed the string contains only numeric digits and whitespace.

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

Returns:
Status: 0 = OK, negative value = error
5.23.3.33 EXTERNRT int rtxUTF8StrToNamedBits (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OSBiTMapItem * pBitMap, OSOCTET * pvalue, OSUINT32 * pnbits, OSUINT32 bufsize)

This function converts the given null-terminated UTF-8 string to named bit items.
The token-to-bit mappings are defined by a bit map table that is passed into the function. It is assumed the string contains a space-separated list of named bit token values.

Parameters:

- `pctxt` Context structure
- `utf8str` Null-terminated UTF-8 string to convert
- `pBitMap` Bit map defining bit to otken mappings
- `pvalue` Pointer to byte array to receive result.
- `pnbits` Pointer to integer to received number of bits.
- `bufsize` Size of byte array to received decoded bits.

Returns:

- Status: 0 = OK, negative value = error

5.23.3.34 EXTERNRT int rtxUTF8StrToUInt (const OSUTF8CHAR * utf8str, OSUINT32 * pvalue)

This function converts the given null-terminated UTF-8 string to an unsigned integer value.
It is assumed the string contains only numeric digits and whitespace.

Parameters:

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

Returns:

- Status: 0 = OK, negative value = error

5.23.3.35 EXTERNRT int rtxUTF8StrToUInt64 (const OSUTF8CHAR * utf8str, OSUINT64 * pvalue)

This function converts the given null-terminated UTF-8 string to an unsigned 64-bit integer value.
It is assumed the string contains only numeric digits and whitespace.

Parameters:

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

Returns:

- Status: 0 = OK, negative value = error
5.23.3.36 EXTERNRT int rtxUTF8ToDynUniStr (OSCTXT * pctxt, const OSUTF8CHAR * utf8str, const OSUNICHAR ** ppdata, OSUINT32 * pnchars)

This function converts the given UTF-8 string to a Unicode string. Memory is allocated for the Unicode string using the rtxMemAlloc function. This memory will be freed when the context is freed (rtxFreeContext) or it can be freed using rtxMemFreePtr.

Parameters:

- pctxt A pointer to a context structure.
- utf8str UTF-8 string to convert, null-terminated.
- ppdata Pointer to pointer to receive output string.
- pnchars Pointer to integer to receive number of chars decoded.

Returns:

Status: 0 = OK, negative value = error

5.23.3.37 EXTERNRT long rtxUTF8ToUnicode (OSCTXT * pctxt, const OSUTF8CHAR * inbuf, OSUNICHAR * outbuf, size_t outbufsiz)

This function converts a UTF-8 string to a Unicode string (UTF-16). The Unicode string is stored as an array of 16-bit characters (unsigned short integers).

Parameters:

- pctxt A pointer to a context structure.
- inbuf UTF-8 string to convert.
- outbuf Output buffer to receive converted Unicode data.
- outbufsiz Size of the output buffer in bytes.

Returns:

Completion status of operation:

- number of Unicode characters in the string
- negative return value is error.

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

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

Parameters:

- pctxt A pointer to a context structure.
- inbuf A pointer to the null-terminated UTF-8 encoded string.

Returns:

Completion status of operation:

- 0 = success,
- negative return value is error.
Chapter 6

Class Documentation

6.1 OSBiMapItem Struct Reference

Named bit in a bit map.
#include <osSysTypes.h>

6.1.1 Detailed Description

Named bit in a bit map.
This structure is used to equate a name with a bit in a bit map.
Definition at line 265 of file osSysTypes.h.
The documentation for this struct was generated from the following file:

• osSysTypes.h
6.2 OSCTX Struct Reference

Run-time context structure.
#include <rtxContext.h>

6.2.1 Detailed Description

Run-time context structure.
This structure is a container structure that holds all working variables involved in encoding or decoding a message.
Definition at line 170 of file rtxContext.h.
The documentation for this struct was generated from the following file:

- rtxContext.h
6.3  **OSDynOctStr Struct Reference**

Dynamic binary string structure.

#include <osSysTypes.h>

6.3.1  **Detailed Description**

Dynamic binary string structure.

This structure is used in generated code for XSD hexBinary and base64Binary types.

Definition at line 206 of file osSysTypes.h.

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

- osSysTypes.h
6.4  OSNumDateTime Struct Reference

Numeric date/time structure.
#include <osSysTypes.h>

6.4.1  Detailed Description

Numeric date/time structure.
Definition at line 107 of file osSysTypes.h.
The documentation for this struct was generated from the following file:

- osSysTypes.h
6.5 OSRTBuffer Struct Reference

Run-time message buffer structure.
#include <rtxContext.h>

6.5.1 Detailed Description

Run-time message buffer structure.

This structure holds encoded message data. For an encode operation, it is where the message being built is stored. For decode, it holds a copy of the message that is being decoded.

Definition at line 89 of file rtxContext.h.

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

• rtxContext.h
6.6 OSRTBufSave Struct Reference

Structure to save the current message buffer state.
#include <rtxContext.h>

6.6.1 Detailed Description

Structure to save the current message buffer state.
This structure is used to save the current state of the buffer.
Definition at line 106 of file rtxContext.h.
The documentation for this struct was generated from the following file:

- rtxContext.h
6.7 OSRTDLList Struct Reference

This is the main list structure.
#include <rtxDList.h>

Public Attributes

• OSUINT32 count
  Count of items in the list.

• OSRTDLListNode * head
  Pointer to first entry in list.

• OSRTDLListNode * tail
  Pointer to last entry in list.

6.7.1 Detailed Description

This is the main list structure.
It contains a count of the number of elements in the list and pointers to the list head and tail elements.
Definition at line 64 of file rtxDList.h.

6.7.2 Member Data Documentation

6.7.2.1 OSUINT32 OSRTDLList::count

Count of items in the list.
Definition at line 65 of file rtxDList.h.

6.7.2.2 OSRTDLListNode* OSRTDLList::head

Pointer to first entry in list.
Definition at line 66 of file rtxDList.h.

6.7.2.3 OSRTDLListNode* OSRTDLList::tail

Pointer to last entry in list.
Definition at line 67 of file rtxDList.h.

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

  • rtxDList.h
6.8 OSRTDListNode Struct Reference

This structure is used to hold a single data item within the list.
#include <rtxDList.h>

Public Attributes

• void * data
  Pointer to list data item.

• struct OSRTDListNode * next
  Pointer to next node in list.

• struct OSRTDListNode * prev
  Pointer to previous node in list.

6.8.1 Detailed Description

This structure is used to hold a single data item within the list.
It contains a void pointer to point at any type of data item and forward and backward pointers to the next and previous entries in the list.
Definition at line 52 of file rtxDList.h.

6.8.2 Member Data Documentation

6.8.2.1 void* OSRTDListNode::data

Pointer to list data item.
Definition at line 53 of file rtxDList.h.

6.8.2.2 struct OSRTDListNode* OSRTDListNode::next [read]

Pointer to next node in list.
Definition at line 54 of file rtxDList.h.

6.8.2.3 struct OSRTDListNode* OSRTDListNode::prev [read]

Pointer to previous node in list.
Definition at line 55 of file rtxDList.h.

The documentation for this struct was generated from the following file:
  • rtxDList.h
6.9 OSRTErrInfo Struct Reference

Run-time error information structure.
#include <rtxContext.h>

6.9.1 Detailed Description

Run-time error information structure.

This structure is a container structure that holds information on run-time errors. The stack variable holds the trace stack information that shows where the error occurred in the source code. The parms variable holds error parameters that are substituted into the message that is returned to the user.

Definition at line 66 of file rtxContext.h.

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

- rtxContext.h

154
6.10 OSRTErrLocn Struct Reference

Run-time error location structure.
#include <rtxContext.h>

6.10.1 Detailed Description

Run-time error location structure.
This structure is a container structure that holds information on the location within a C source file where a run-time error occurred.
Definition at line 51 of file rtxContext.h.
The documentation for this struct was generated from the following file:

- rtxContext.h
6.11 OSRTPrintStream Struct Reference

Structure to hold information about a global PrintStream.
#include <rtxPrintStream.h>

6.11.1 Detailed Description

Structure to hold information about a global PrintStream.
Definition at line 51 of file rtxPrintStream.h.
The documentation for this struct was generated from the following file:

- rtxPrintStream.h
6.12 OSRTScalarDList Struct Reference

This is the main list structure.
#include <rtxScalarDList.h>

Public Attributes

- **OSUINT32 count**
  Count of items in the list.

- **OSRTScalarDListNode * head**
  Pointer to first entry in list.

- **OSRTScalarDListNode * tail**
  Pointer to last entry in list.

6.12.1 Detailed Description

This is the main list structure.
It contains a count of the number of elements in the list and pointers to the list head and tail elements.
Definition at line 92 of file rtxScalarDList.h.
The documentation for this struct was generated from the following file:

- rtxScalarDList.h
6.13 OSRTScalarDListNode Struct Reference

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

#include <rtxScalarDList.h>

Public Attributes

- struct OSRTScalarDListNode * next
  Pointer to next node in list.

- struct OSRTScalarDListNode * prev
  Pointer to previous node in list.

- OSDOUBLE dfltval
  Double prec floating point value.

- OSFLOAT fltval
  Single prec floating point value.

- OSINT32 i32val
  32-bit signed integer

- OSUINT32 ui32val
  32-bit unsigned integer

- OSINT64 i64val
  64-bit signed integer

- OSUINT64 ui64val
  64-bit unsigned integer

6.13.1 Detailed Description

This structure is used to hold a single data item within the list.
The data item is a union of all of the possible scalar types it can hold. The node also contains forward and backward
pointers to the next and previous entries in the list.
Definition at line 68 of file rtxScalarDList.h.
The documentation for this struct was generated from the following file:

- rtxScalarDList.h
6.14 OSRTSTREAM Struct Reference

The stream control block.

#include <rtxStream.h>

Public Attributes

- OSRTMEMBUF * pCaptureBuf
  
  Buffer into which data read from stream can be captured for debugging purposes.

6.14.1 Detailed Description

The stream control block.

A user may implement a customized stream by defining read, skip, close functions for input streams and write, flush, close for output streams.

Definition at line 156 of file rtxStream.h.

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

- rtxStream.h
6.15 OSXMLFullQName Struct Reference

This version of QName contains complete namespace info (prefix + URI).
#include <rtxXmlQName.h>

6.15.1 Detailed Description

This version of QName contains complete namespace info (prefix + URI).
Definition at line 36 of file rtxXmlQName.h.
The documentation for this struct was generated from the following file:

• rtxXmlQName.h
6.16  OSXMLSTRING Struct Reference

XML UTF-8 character string structure.
#include <osSysTypes.h>

6.16.1  Detailed Description

XML UTF-8 character string structure.
This structure is used in generated code for XML string types.
Definition at line 244 of file osSysTypes.h.
The documentation for this struct was generated from the following file:

  • osSysTypes.h
6.17 OSXSDAny Struct Reference

Structure to hold xsd:any data in binary and XML text form.
#include <osSysTypes.h>

6.17.1 Detailed Description

Structure to hold xsd:any data in binary and XML text form.
Definition at line 218 of file osSysTypes.h.
The documentation for this struct was generated from the following file:

• osSysTypes.h
6.18 OSXSDDateTime Struct Reference

Numeric date/time structure.

#include <osSysTypes.h>

6.18.1 Detailed Description

Numeric date/time structure.

This structure is used in generated code for XSD date/time types when code generation is configured to use numeric date/time types (-numDateTime command-line option).

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

- osSysTypes.h
Chapter 7

File Documentation

7.1 rtxArrayList.h File Reference

ArrayList functions.
#include "rtxsrc/rtxContext.h"

Functions

- EXTERNRT void rtxArrayListInit (OSRTArrayList ∗pArrayList, size_t capacity)
  This function initializes an array list structure.

- EXTERNRT OSRTArrayList ∗rtxNewArrayList (OSCTXT ∗pctxt, size_t capacity)
  This function creates a new array list to hold the initial capacity of elements.

- EXTERNRT void rtxFreeArrayList (OSCTXT ∗pctxt, OSRTArrayList ∗pArrayList)
  This function frees all dynamic memory held by the array list.

- EXTERNRT int rtxArrayListAdd (OSCTXT ∗pctxt, OSRTArrayList ∗pArrayList, void ∗pdata, OSUINT32 ∗pindex)
  This function adds an element to an array list.

- EXTERNRT void rtxArrayListRemove (OSCTXT ∗pctxt, OSRTArrayList ∗pArrayList void ∗pdata)
  This function removes an element from an array list.

- EXTERNRT void rtxArrayListRemoveIndexed (OSCTXT ∗pctxt, OSRTArrayList ∗pArrayList, int index)
  This function removes the element at the given index from the array list.

- EXTERNRT int rtxArrayListInsert (OSCTXT ∗pctxt, OSRTArrayList ∗pArrayList, void ∗pdata, OSUINT32 index)
  This function inserts an element at the given position in the array list.

- EXTERNRT int rtxArrayListReplace (OSRTArrayList ∗pArrayList, void ∗pdata, OSUINT32 index)
  This function replaces (overwrites) the element at the given position in the array list with the new element.

- EXTERNRT void ∗rtxArrayListGetIndexed (const OSRTArrayList ∗pArrayList, OSUINT32 index)
This function gets the indexed data item from the array list.

- EXTERNRT int rtxArrayListIndexOf (OSRTArrayList ∗pArrayList, void ∗pdata)
  This function returns the index of the given data item in the list.

- EXTERNRT int rtxArrayListInitIter (OSRTArrayListIter ∗piter, OSRTArrayList ∗pArrayList, OSUINT32 startIndex)
  This function initializes an array list iterator with the given start index.

- EXTERNRT OSBOOL rtxArrayListHasNextItem (OSRTArrayListIter ∗piter)
  This function determines if another element exists at the next sequential position in the array list.

- EXTERNRT void ∗rtxArrayListNextItem (OSRTArrayListIter ∗piter)
  This function gets the next item from the array list.

7.1.1 Detailed Description

ArrayList functions.
Definition in file rtxArrayList.h.

7.1.2 Function Documentation

7.1.2.1 EXTERNRT int rtxArraylistAdd (OSCTXT ∗pctxt, OSRTArrayList ∗pArrayList, void ∗pdata, OSUINT32 ∗pindex)
This function adds an element to an array list.

Parameters:
  pctxt  Pointer to a context structure.
  pArrayList  Pointer to array list structure to initialize.
  pdata  Pointer to data item to add.
  pindex  Pointer to index variable to receive index at which entry was added.

Returns:
  Zero if item was successfully added; a negative status code if error.

7.1.2.2 EXTERNRT void ∗rtxArrayListGetIndexed (const OSRTArrayList ∗pArrayList, OSUINT32 index)
This function gets the indexed data item from the array list.

Parameters:
  pArrayList  Pointer to array list structure to initialize.
  index  Index of location where item should be inserted.

Returns:
  Pointer to indexed data item or NULL if index is greater than max index in list.
7.1.2.3 **EXTERNRT OSBOOL** rtxArrayListHasNextItem (OSRTArrayListIter *piter)

This function determines if another element exists at the next sequential position in the array list.

**Parameters:**

- *piter* Pointer to array list iterator structure.

**Returns:**

- True if another element exists; false otherwise.

7.1.2.4 **EXTERNRT int** rtxArrayListIndexOf (OSRTArrayList *pArrayList, void *pdata)

This function returns the index of the given data item in the list.

The 'equals' callback function is used to do comparisons.

**Parameters:**

- *pArrayList* Pointer to array list structure to initialize.
- *pdata* Pointer to data item to find in list.

**Returns:**

- Index of item in list or -1 if not found.

7.1.2.5 **EXTERNRT void** rtxArrayListInit (OSRTArrayList *pArrayList, size_t capacity)

This function initializes an array list structure.

**Parameters:**

- *pArrayList* Pointer to array list structure to initialize.
- *capacity* Initial capacity of the array or zero to use default.

7.1.2.6 **EXTERNRT int** rtxArrayListInitIter (OSRTArrayListIter *piter, OSRTArrayList *pArrayList, OSUINT32 startIndex)

This function initializes an array list iterator with the given start index.

**Parameters:**

- *piter* Pointer to array list iterator structure.
- *pArrayList* Pointer to array list structure.
- *startIndex* Index from which iteration is to start.

**Returns:**

- Zero if successfully initialized or RTERR_OUTOFBND if start index is beyond the current size of the array list.
7.1.2.7 EXTERNRT int rtxArrayListInsert (OSCTXT * pctxt, OSRTArrayList * pArrayList, void * pdata, OSUINT32 index)

This function inserts an element at the given position in the array list.

Parameters:
- \textit{pctxt} Pointer to a context structure.
- \textit{pArrayList} Pointer to array list structure to initialize.
- \textit{pdata} Pointer to data item to insert.
- \textit{index} Index of location where item should be inserted.

Returns:
Zero if item was successfully added; a negative status code if error.

7.1.2.8 EXTERNRT void* rtxArrayListNextItem (OSRTArrayListIter * piter)

This function gets the next item from the array list.

Parameters:
- \textit{piter} Pointer to array list iterator structure.

Returns:
Pointer to next item or null if beyond the end of the array.

7.1.2.9 EXTERNRT void rtxArrayListRemove (OSCTXT * pctxt, OSRTArrayList * pArrayList, void * pdata)

This function removes an element from an array list.

Parameters:
- \textit{pctxt} Pointer to a context structure.
- \textit{pArrayList} Pointer to array list structure to initialize.
- \textit{pdata} Pointer to data item to remove.

7.1.2.10 EXTERNRT void rtxArrayListRemoveIndexed (OSCTXT * pctxt, OSRTArrayList * pArrayList, int index)

This function removes the element at the given index from the array list.

Parameters:
- \textit{pctxt} Pointer to a context structure.
- \textit{pArrayList} Pointer to array list structure to initialize.
- \textit{index} Index of item to remove. -1 indicates tail item should be removed.
7.1.2.11 EXTERNRT int rtxArrayListReplace (OSRTArrayList *pArrayList, void *pdata, OSUINT32 index)

This function replaces (overwrites) the element at the given position in the array list with the new element.

Parameters:
- pArrayList  Pointer to array list structure to initialize.
- pdata  Pointer to data item to insert.
- index  Index of location where item should be inserted.

Returns:
Zero if item was successfully added; a negative status code if error.

7.1.2.12 EXTERNRT void rtxFreeArrayList (OSCTXT *pctxt, OSRTArrayList *pArrayList)

This function frees all dynamic memory held by the array list.
It does not free the array list structure itself, just the internal data array.

Parameters:
- pctxt  Pointer to a context structure.
- pArrayList  Pointer to array list structure.

7.1.2.13 EXTERNRT OSRTArrayList* rtxNewArrayList (OSCTXT *pctxt, size_t capacity)

This function creates a new array list to hold the initial capacity of elements.

Parameters:
- pctxt  Pointer to a context structure.
- capacity  Initial capacity of the array or zero to use default.

Returns:
Allocated array list structure or NULL if insufficient dynamic memory is available to hold the structure.
7.2 rtxBase64.h File Reference

#include "rtxsig/rtxContext.h"

Functions

• EXTERNRT long rtxBase64EncodeData (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET **ppDstData)
  
  *Encode binary data into base64 string form to a dynamic buffer.*

• EXTERNRT long rtxBase64DecodeData (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET **ppDstData)
  
  *Decode base64 string to binary form into a dynamic buffer.*

• EXTERNRT long rtxBase64DecodeDataToFSB (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET *buf, size_t bufsiz)
  
  *Decode base64 string to binary form into a fixed-size buffer.*

• EXTERNRT long rtxBase64GetBinDataLen (const char *pSrcData, size_t srcDataSize)
  
  *Calculate number of byte required to hold a decoded base64 string in binary form.*

7.2.1 Detailed Description

Definition in file rtxBase64.h.

7.2.2 Function Documentation

7.2.2.1 EXTERNRT long rtxBase64DecodeData (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET **ppDstData)

Decode base64 string to binary form into a dynamic buffer.

Parameters:

  *pctxt*  Pointer to context structure.
  
  *pSrcData*  Pointer to base64 string to decode.
  
  *srcDataSize*  Length of the base64 string.
  
  *ppDstData*  Pointer to pointer variable to hold address of dynamically allocated buffer to hold data.

Returns:

  Completion status of operation:
  
  • number of binary bytes written
  
  • negative return value is error.
7.2.2.2 EXTERNRT long rtxBase64DecodeDataToFSB (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET * buf, size_t bufsiz)

Decode base64 string to binary form into a fixed-size buffer.

Parameters:

- *pctxt* Pointer to context structure.
- *pSrcData* Pointer to base64 string to decode.
- *srcDataSize* Length of the base64 string.
- *buf* Address of buffer to receive decoded binary data.
- *bufsiz* Size of output buffer.

Returns:

Completion status of operation:

- number of binary bytes written
- negative return value is error.

7.2.2.3 EXTERNRT long rtxBase64EncodeData (OSCTXT * pctxt, const char * pSrcData, size_t srcDataSize, OSOCTET ** ppDstData)

Encode binary data into base64 string form to a dynamic buffer.

Parameters:

- *pctxt* Pointer to context structure.
- *pSrcData* Pointer to binary data to encode.
- *srcDataSize* Length of the binary data in octets.
- * ppDstData* Pointer to pointer variable to hold address of dynamically allocated buffer the encoded base64 string.

Returns:

Completion status of operation:

- number of binary bytes written
- negative return value is error.

7.2.2.4 EXTERNRT long rtxBase64GetBinDataLen (const char * pSrcData, size_t srcDataSize)

Calculate number of byte required to hold a decoded base64 string in binary form.

Parameters:

- *pSrcData* Pointer to base64 string to decode.
- *srcDataSize* Length of the base64 string.

Returns:

Completion status of operation: If success, positive value is number of bytes, If failure, negative status code.
7.3  rtxBigInt.h File Reference

#include "rtxsrclrtxContext.h"

7.3.1  Detailed Description

Definition in file rtxBigInt.h.
7.4 rtxBitDecode.h File Reference

Bit decode functions.

#include "rtxsrt/rtxContext.h"

Functions

- EXTERNRT int rtxDecBits (OSCTXT *pctxt, OSUINT32 *pvalue, OSUINT32 nbits)

  This function decodes up to sizeof(unsigned) bits and returns the result in an unsigned integer value.

7.4.1 Detailed Description

Bit decode functions.
Definition in file rtxBitDecode.h.

7.4.2 Function Documentation

7.4.2.1 EXTERNRT int rtxDecBits (OSCTXT *pctxt, OSUINT32 *pvalue, OSUINT32 nbits)

This function decodes up to sizeof(unsigned) bits and returns the result in an unsigned integer value.

Parameters:

  pctxt Pointer to a context structure. This provides a storage area for the function to store all working variables
          that must be maintained between function calls.
  pvalue Pointer to value to be receive decoded result.
  nbits Number of bits to read from decode buffer.

Returns:

  Status of the operation. Zero if successful; a negative status code if failed.
7.5  rtxBitEncode.h File Reference

Bit encode functions.
#include "rtxsrc/rtxContext.h"

Functions

• EXTERNRT int rtxEncBit (OSCTX ∗ pctxt, OSBOOL value)
  This function will set the bit at the current encode bit cursor position to 1 or 0 and advance the cursor pointer.

• EXTERNRT int rtxEncBits (OSCTX ∗ pctxt, OSUINT32 value, OSUINT32 nbits)
  This function will encode a series of bits (up to 32) from an unsigned integer value/.

7.5.1 Detailed Description

Bit encode functions.
Definition in file rtxBitEncode.h.

7.5.2 Function Documentation

7.5.2.1 EXTERNRT int rtxEncBit (OSCTX ∗ pctxt, OSBOOL value)

This function will set the bit at the current encode bit cursor position to 1 or 0 and advance the cursor pointer.

Parameters:

  pctxt Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
  value The value to be encoded.

7.5.2.2 EXTERNRT int rtxEncBits (OSCTX ∗ pctxt, OSUINT32 value, OSUINT32 nbits)

This function will encode a series of bits (up to 32) from an unsigned integer value/.

Parameters:

  pctxt Pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
  value The value to be encoded.
  nbits Number of bits from the value to encode.

Returns:

  Status of the operation. Zero if successful; a negative status code if failed.
7.6  rt(getString.h File Reference

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

#include "rtxs/rtxContext.h"

Defines

- #define OSRTBYTEARRAYSIZE(numbits) (((numbits-1)/8)+1)
  This macro is used to calculate the byte array size required to hold the given number of bits.

Functions

- EXTERNRT int rtxSetBit (OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)
  This function sets the specified bit in the bit string.

- EXTERNRT OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)
  This function sets one or more bits to TRUE or FALSE in a 32-bit unsigned bit flag set.

- EXTERNRT int rtxClearBit (OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)
  This function clears the specified bit in the bit string.

- EXTERNRT OSBOOL rtxTestBit (const OSOCTET *pBits, OSUINT32 numbits, OSUINT32 bitIndex)
  This function tests the specified bit in the bit string.

7.6.1 Detailed Description

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

Definition in file rtBitString.h.
Common runtime functions for reading from or writing to the message buffer defined within the context structure.

#include "rtxsrg/rtxContext.h"

7.7.1 Detailed Description

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

Definition in file rtxBuffer.h.
7.8 rtxCharStr.h File Reference

#include "rtxs src/rtxContext.h"

Functions

- EXTERNRT char * rtxStrcat (char *dest, size_t bufsiz, const char *src)
  This function concatenates the given string onto the string buffer.

- EXTERNRT char * rtxStrncat (char *dest, size_t bufsiz, const char *src, size_t nchars)
  This function concatenates the given number of characters from the given string onto the string buffer.

- EXTERNRT char * rtxStrcpy (char *dest, size_t bufsiz, const char *src)
  This function copies a null-terminated string to a target buffer.

- EXTERNRT char * rtxStrncpy (char *dest, size_t bufsiz, const char *src, size_t nchars)
  This function copies the given number of characters from a string to a target buffer.

- EXTERNRT int rtxIntToCharStr (OSINT32 value, char *dest, size_t bufsiz, char padchar)
  This function converts a signed 32-bit integer into a character string.

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

- EXTERNRT int rtxInt64ToCharStr (OSINT64 value, char *dest, size_t bufsiz, char padchar)
  This function converts a signed 64-bit integer into a character string.

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

- EXTERNRT int rtxHexCharsToBinCount (const char *hexstr, size_t nchars)
  This function returns a count of the number of bytes the would result from the conversion of a hexadecimal character string to binary.

- EXTERNRT int rtxHexCharsToBin (const char *hexstr, size_t nchars, OSOCTET *binbuf, size_t bufsize)
  This function converts the given hex string to binary.

7.8.1 Detailed Description

Definition in file rtxCharStr.h.
7.9  rtCommon.h File Reference

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

```c
#include "rtxsrt/osSysTypes.h"
#include "rtxsrt/osMacros.h"
#include "rtxsrt/rtxExternDefs.h"
#include "rtxsrt/rtxBigInt.h"
#include "rtxsrt/rtxBitString.h"
#include "rtxsrt/rtxBuffer.h"
#include "rtxsrt/rtxCharStr.h"
#include "rtxsrt/rtxCommonDefs.h"
#include "rtxsrt/rtxDateTime.h"
#include "rtxsrt/rtxDiag.h"
#include "rtxsrt/rtxEnum.h"
#include "rtxsrt/rtxError.h"
#include "rtxsrt/rtxFile.h"
#include "rtxsrt/rtxMemory.h"
#include "rtxsrt/rtxPattern.h"
#include "rtxsrt/rtxReal.h"
#include "rtxsrt/rtxUTF8.h"
#include "rtxsrt/rtxUtil.h"
```

7.9.1 Detailed Description

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

Definition in file rtCommon.h.
7.10  rtxContext.h File Reference

Common run-time context definitions.
#include "rtxsrc/rtxDList.h"

Classes

• struct OSRTErrLocn
  Run-time error location structure.

• struct OSRTErrInfo
  Run-time error information structure.

• struct OSRTBuffer
  Run-time message buffer structure.

• struct OSRTBufSave
  Structure to save the current message buffer state.

• struct OSCTXT
  Run-time context structure.

Defines

• #define rtxCtxtGetMsgPtr(pctxt) (pctxt) \rightarrow buffer.data
  This macro returns the start address of an encoded message.

• #define rtxCtxtGetMsgLen(pctxt) (pctxt) \rightarrow buffer.byteIndex
  This macro returns the length of an encoded message.

• #define rtxCtxtTestFlag(pctxt, mask) ((pctxt) \rightarrow flags & mask) != 0
  This macro tests if the given bit flag is set in the context.

• #define rtxByteAlign(pctxt)
  This macro will byte-align the context buffer.

Functions

• EXTERNRT int rtxInitContextUsingKey (OSCTXT *pctxt, const OSOCTET *key, size_t keylen)
  This function initializes a context using a run-time key.

• EXTERNRT int rtxInitContext (OSCTXT *pctxt)
  This function initializes an OSCTXT block.

• EXTERNRT int rtxInitContextExt (OSCTXT *pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
This function initializes an OSCTXT block.

- EXTERNRT int rtxInitThreadContext (OSCTXT *pctxt, const OSCTXT *pSrcCtxt)
  This function initializes a context for use in a thread.

- EXTERNRT int rtxInitContextBuffer (OSCTXT *pctxt, OSOCTET *bufaddr, size_t bufsiz)
  This function assigns a message buffer to a context block.

- EXTERNRT int rtxCtxtSetBufPtr (OSCTXT *pctxt, OSOCTET *bufaddr, size_t bufsiz)
  This function is used to set the internal buffer pointer for in-memory encoding or decoding.

- EXTERNRT size_t rtxCtxtGetIOByteCount (OSCTXT *pctxt)
  This function returns the count of bytes either written to a stream or memory buffer.

- EXTERNRT int rtxCheckContext (OSCTXT *pctxt)
  This function verifies that the given context structure is initialized and ready for use.

- EXTERNRT void rtxFreeContext (OSCTXT *pctxt)
  This function frees all dynamic memory associated with a context.

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

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

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

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

### 7.10.1 Detailed Description

Common run-time context definitions.

Definition in file rtxContext.h.
7.11  rtxCtype.h File Reference

7.11.1  Detailed Description

Definition in file rtxCtype.h.
7.12  rtxDateTime.h File Reference

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

#include <time.h>
#include "rtxsrc/rtxContext.h"

Functions

- EXTERNRT int rtxDateToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  This function formats a numeric date value consisting of individual date components (year, month, day) into XML schema standard format (CCYY-MM-DD).

- EXTERNRT int rtxTimeToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  This function formats a numeric time value consisting of individual time components (hour, minute, second, fraction-of-second, time zone) into XML schema standard format (HH:MM:SS[.frac][TZ]).

- EXTERNRT int rtxDateTimeToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
  This function formats a numeric date/time value of all components in the OSNumDateTime structure into XML schema standard format (CCYY-MM-DDTHH:MM:SS[.frac][TZ]).

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

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

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

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

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

- EXTERNRT int rtxGetCurrDateTime (OSNumDateTime *pvalue)
  This function reads the system date and time and stores the value in the given OSNumDateTime structure variable.

- EXTERNRT int rtxCmpDate (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
  This function compares the date part of two OSNumDateTime structures and returns the result of the comparison.

- EXTERNRT int rtxCmpDate2 (const OSNumDateTime *pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSBOOL tzflag, OSINT32 tzo)
  This function compares the date part of OSNumDateTime structure and date components, specified as parameters.

- EXTERNRT int rtxCmpTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
This function compares the time part of two \texttt{OSNumDateTime} structures and returns the result of the comparison.

- \textbf{EXTERNRT int rtxCmpTime2} (const \texttt{OSNumDateTime \*pvalue}, \texttt{OSUINT8} hour, \texttt{OSUINT8} min, \texttt{OSREAL} sec, \texttt{OSBOOL} tzflag, \texttt{OSINT32} tzo)
  
  This function compares the time part of \texttt{OSNumDateTime} structure and time components, specified as parameters.

- \textbf{EXTERNRT int rtxCmpDateTime} (const \texttt{OSNumDateTime \*pvalue1}, const \texttt{OSNumDateTime \*pvalue2})
  
  This function compares two \texttt{OSNumDateTime} structures and returns the result of the comparison.

- \textbf{EXTERNRT int rtxCmpDateTime2} (const \texttt{OSNumDateTime \*pvalue}, \texttt{OSINT32} year, \texttt{OSUINT8} mon, \texttt{OSUINT8} day, \texttt{OSUINT8} hour, \texttt{OSUINT8} min, \texttt{OSREAL} sec, \texttt{OSBOOL} tzflag, \texttt{OSINT32} tzo)
  
  This function compares the \texttt{OSNumDateTime} structure and dateTime components, specified as parameters.

- \textbf{EXTERNRT int rtxParseDateString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{OSNumDateTime \*pvalue})
  
  This function decodes a date value from a supplied string and sets the given \texttt{OSNumDateTime} argument to the decoded date value.

- \textbf{EXTERNRT int rtxParseTimeString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{OSNumDateTime \*pvalue})
  
  This function decodes a time value from a supplied string and sets the given \texttt{OSNumDateTime} structure to the decoded time value.

- \textbf{EXTERNRT int rtxParseDateTimeString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{OSNumDateTime \*pvalue})
  
  This function decodes a datetime value from a supplied string and sets the given \texttt{OSNumDateTime} to the decoded date and time value.

- \textbf{EXTERNRT int rtxParseGYearString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{OSNumDateTime \*pvalue})
  
  This function decodes a gregorian year value from a supplied string and sets the year in the given \texttt{OSNumDateTime} to the decoded value.

- \textbf{EXTERNRT int rtxParseGYearMonthString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{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 \texttt{OSNumDateTime} to the decoded values.

- \textbf{EXTERNRT int rtxParseGMonthString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{OSNumDateTime \*pvalue})
  
  This function decodes a gregorian month value from a supplied string and sets the month field in the given \texttt{OSNumDateTime} to the decoded value.

- \textbf{EXTERNRT int rtxParseGMonthDayString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{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 \texttt{OSNumDateTime} to the decoded values.

- \textbf{EXTERNRT int rtxParseGDayString} (const \texttt{OSUTF8CHAR \*inpdata}, size\_t \texttt{inpdatalen}, \texttt{OSNumDateTime \*pvalue})
  
  This function decodes a gregorian day value from a supplied string and sets the day field in the given \texttt{OSNumDateTime} to the decoded value.
• EXTERNRT int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR ∗buf, OSUINT32 bufsize)
  This function converts milliseconds to a duration string with format "PnYnMnDTnHnMnS".

• EXTERNRT int rtxDurationToMSecs (OSUTF8CHAR ∗buf, OSUINT32 bufsize, OSINT32 ∗msecs)
  This function converts a duration string to milliseconds.

• EXTERNRT int rtxSetDateTime (OSNumDateTime ∗pvalue, struct tm ∗timeStruct)
  This function converts a structure of type 'struct tm' to an OSNumDateTime structure.

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

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

• EXTERNRT int rtxGetDateTime (const OSNumDateTime ∗pvalue, time_t ∗timeMs)
  This function converts an OSNumDateTime structure to a calendar time encoded as a value of type time_t.

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

• EXTERNRT OSBOOL rtxTimeIsValid (const OSNumDateTime ∗pvalue)
  This function verifies that time members (hour, minute, second, timezone) of the OSNumDateTime structure contains valid values.

• EXTERNRT OSBOOL rtxDateTimeIsValid (const OSNumDateTime ∗pvalue)
  This function verifies that all members of the OSNumDateTime structure contains valid values.

7.12.1 Detailed Description

Common runtime functions for converting to and from various standard date/time formats.
Definition in file rtxDateTime.h.
7.13 **rtxDecimal.h File Reference**

Common runtime functions for working with xsd:decimal numbers.

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

### 7.13.1 Detailed Description

Common runtime functions for working with xsd:decimal numbers.

Definition in file `rtxDecimal.h`. 
Common runtime functions for diagnostic tracing and debugging.

`#include <stdarg.h>`

`#include "rtxsrc/rtxContext.h"`

`#include "rtxsrc/rtxPrintToStream.h"`

**Functions**

- **EXTERNRT OSBOOL rtxDiagEnabled (OSCTXT *pctxt)**
  
  *This function is used to determine if diagnostic tracing is currently enabled for the specified context.*

- **EXTERNRT OSBOOL rtxSetDiag (OSCTXT *pctxt, OSBOOL value)**
  
  *This function is used to turn diagnostic tracing on or off at run-time on a per-context basis.*

- **EXTERNRT OSBOOL rtxSetGlobalDiag (OSBOOL value)**
  
  *This function is used to turn diagnostic tracing on or off at run-time on a global basis.*

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

- **EXTERNRT void rtxDiagStream (OSCTXT *pctxt, const char *fmtspec, ...)**
  
  *This function conditionally outputs diagnostic trace messages to an output stream defined within the context.*

- **EXTERNRT void rtxDiagHexDump (OSCTXT *pctxt, const OSOCTET *data, OSUINT32 numocts)**
  
  *This function is used to print a diagnostics hex dump of a section of memory.*

- **EXTERNRT void rtxDiagStreamHexDump (OSCTXT *pctxt, const OSOCTET *data, OSUINT32 numocts)**
  
  *This function is used to print a diagnostics hex dump of a section of memory to a print stream.*

- **EXTERNRT void rtxDiagPrintChars (OSCTXT *pctxt, const char *data, OSUINT32 nchars)**
  
  *This function is used to print a given number of characters to standard output.*

- **EXTERNRT void rtxDiagStreamPrintChars (OSCTXT *pctxt, const char *data, OSUINT32 nchars)**
  
  *This function is used to print a given number of characters to a print stream.*

- **EXTERNRT void rtxDiagSetTraceLevel (OSCTXT *pctxt, OSRTDiagTraceLevel level)**
  
  *This function is used to set the maximum trace level for diagnostic trace messages.*

### 7.14.1 Detailed Description

Common runtime functions for diagnostic tracing and debugging.

Definition in file `rtxDiag.h`. 
7.15 rtxDiagBitTrace.h File Reference

Common runtime functions for tracing bit patterns written to or read from a stream.

```c
#include <stdarg.h>
#include "rtxsrc/rtxMemBuf.h"
#include "rtxsrc/rtxSList.h"
#include "rtxsrc/rtxPrintToStream.h"
```

Defines

- `#define RTDIAG_GETCTXTBITOFFSET(pctxt) (((pctxt) buffer.byteIndex * 8) + (8 - (pctxt) buffer.bitOffset))`
  This macro calculates the relative bit offset to the current buffer position.

Functions

- `EXTERNRT OSBOOL rtxDiagSetBitTraceEnabled (OSCTXT *pctxt, OSBOOL value)`
  This function enables bit field tracing.

- `EXTERNRT OSBOOL rtxDiagBitTraceEnabled (OSCTXT *pctxt)`
  This function check to see if bit field tracing is enabled.

- `EXTERNRT void rtxDiagBitFieldListInit (OSCTXT *pctxt, OSRTDiagBitFieldList *pBFList)`
  This function initializes a bit field list structure.

- `EXTERNRT void rtxDiagInsBitFieldLen (OSRTDiagBitFieldList *pBFList)`
  This function inserts a special length field before the current record in the bit field list.

- `EXTERNRT OSRTDiagBitField * rtxDiagNewBitField (OSRTDiagBitFieldList *pBFList, const char *nameSuffix)`
  This function allocates a new bit field structure and adds it to the bit field list.

- `EXTERNRT void rtxDiagSetBitFldOffset (OSRTDiagBitFieldList *pBFList)`
  This function is used to set the bit offset in the current bit field structure.

- `EXTERNRT void rtxDiagSetBitFldCount (OSRTDiagBitFieldList *pBFList)`
  This function is used to set the bit count in the current bit field structure.

- `EXTERNRT void rtxDiagSetBitFldNameSuffix (OSRTDiagBitFieldList *pBFList, const char *nameSuffix)`
  This function is used to set the name suffix in the current bit field structure.

- `EXTERNRT OSBOOL rtxDiagSetBitFldDisabled (OSRTDiagBitFieldList *pBFList, OSBOOL value)`
  This function increments or decrements the disabled count.

- `EXTERNRT void rtxDiagBitTracePrint (OSRTDiagBitFieldList *pBFList, const char *varname)`
  This function prints the bit field list to a an output stream.
• EXTERNRT void rtxDiagBitTracePrintHTML (const char *filename, OSRTDiagBitFieldList *pBFList, const char *varname)
   
   This function prints the bit field list to a HTML document.

• EXTERNRT void rtxDiagBitFldAppendNamePart (OSRTDiagBitFieldList *pBFList, const char *namePart)
   
   This function appends the given name part to the element name in the bit field.

7.15.1 Detailed Description

Common runtime functions for tracing bit patterns written to or read from a stream. 
Definition in file rtxDiagBitTrace.h.

7.15.2 Function Documentation

7.15.2.1 EXTERNRT void rtxDiagBitFieldListInit (OSCTXT *pctxt, OSRTDiagBitFieldList *pBFList)

This function initializes a bit field list structure.

Parameters:

   pctxt  Pointer to a context structure.
   pBFList  Pointer to bit field list structure.

7.15.2.2 EXTERNRT void rtxDiagBitF1dAppendNamePart (OSRTDiagBitFieldList *pBFList, const char *namePart)

This function appends the given name part to the element name in the bit field.
A dot (.) separator character is added after the existing name and before the name part.

Parameters:

   pBFList  Pointer to bit field list structure.
   namePart  A name part that is appended to the field.

7.15.2.3 EXTERNRT OSBOOL rtxDiagBitTraceEnabled (OSCTXT *pctxt)

This function check to see if bit field tracing is enabled.

Parameters:

   pctxt  Pointer to a context structure.

Returns:

   True if enabled; false otherwise.
7.15.2.4 EXTERNRT void rtxDiagBitTracePrint (OSRTDiagBitFieldList * pBFList, const char * varname)

This function prints the bit field list to a an output stream.

By default, the output goes to stdout; but this can be changed by creating a print output stream within the context (see rtxPrintStream).

Parameters:

    pBFList  Pointer to bit field list structure.
    varname  A variable name that is prepended to each field.

7.15.2.5 EXTERNRT void rtxDiagBitTracePrintHTML (const char * filename, OSRTDiagBitFieldList * pBFList, const char * varname)

This function prints the bit field list to a an HTML document.

Parameters:

    filename  Name of HTML file to be written.
    pBFList   Pointer to bit field list structure.
    varname   A variable name that is prepended to each field.

7.15.2.6 EXTERNRT void rtxDiagInsBitFieldLen (OSRTDiagBitFieldList * pBFList)

This function inserts a special length field before the current record in the bit field list.

Parameters:

    pBFList  Pointer to bit field list structure.

7.15.2.7 EXTERNRT OSRTDiagBitField* rtxDiagNewBitField (OSRTDiagBitFieldList * pBFList, const char * nameSuffix)

This function allocates a new bit field structure and adds it to the bit field list.

Parameters:

    pBFList  Pointer to bit field list structure.
    nameSuffix  Suffix to append to the bit field name.

Returns:

    Allocated bit field structure.

7.15.2.8 EXTERNRT void rtxDiagSetBitFldCount (OSRTDiagBitFieldList * pBFList)

This function is used to set the bit count in the current bit field structure.

Parameters:

    pBFList  Pointer to bit field list structure.
7.15.2.9  EXTERNRT OSBOOL rtxDiagSetBitFldDisabled (OSRTDiagBitFieldList * pBFList, OSBOOL value)

This function increments or decrements the disabled count.
This allows the list to be temporarily disabled to allow collection of more bits to form larger, aggregate fields.

Parameters:
  pBFList  Pointer to bit field list structure.
  value    Indicates if disabled count should be incremented (TRUE) or decremented (FALSE).

Returns:
  TRUE if field operations are still disabled.

7.15.2.10 EXTERNRT void rtxDiagSetBitFldNameSuffix (OSRTDiagBitFieldList * pBFList, const char * nameSuffix)

This function is used to set the name suffix in the current bit field structure.
This text is printed after the element name when the field is displayed.

Parameters:
  pBFList  Pointer to bit field list structure.
  nameSuffix  Suffix to append to the bit field name.

7.15.2.11 EXTERNRT void rtxDiagSetBitFldOffset (OSRTDiagBitFieldList * pBFList)

This function is used to set the bit offset in the current bit field structure.

Parameters:
  pBFList  Pointer to bit field list structure.

7.15.2.12 EXTERNRT OSBOOL rtxDiagSetBitTraceEnabled (OSCTXT * pctxt, OSBOOL value)

This function enables bit field tracing.

Parameters:
  pctxt    Pointer to a context structure.
  value    Boolean value to turn trace on or off.

Returns:
  Previous state of bit trace enabled setting.
Doubly-Linked List Utility Functions.
#include "rtxs src/osSysTypes.h"
#include "rtxs src/rtxExternDefs.h"
#include "rtxs src/rtxCommonDefs.h"

Classes

• struct OSRTDListNode
  This structure is used to hold a single data item within the list.

• struct OSRTDList
  This is the main list structure.

Functions

• EXTERNRT void rtxDListInit (OSRTDList *pList)
  This function initializes a doubly linked list structure.

• EXTERNRT OSRTDListNode * rtxDListAppend (struct OSCTXT *pctxt, OSRTDList *pList, void *pData)
  This function appends an item to the linked list structure.

• EXTERNRT OSRTDListNode * rtxDListInsert (struct OSCTXT *pctxt, OSRTDList *pList, OSUINT32 index, void *pData)
  This function inserts an item into the linked list structure.

• EXTERNRT OSRTDListNode * rtxDListInsertBefore (struct OSCTXT *pctxt, OSRTDList *pList, OSRTDListNode *node, void *pData)
  This function inserts an item into the linked list structure before the specified element.

• EXTERNRT OSRTDListNode * rtxDListInsertAfter (struct OSCTXT *pctxt, OSRTDList *pList, OSRTDListNode *node, void *pData)
  This function inserts an item into the linked list structure after the specified element.

• EXTERNRT OSRTDListNode * rtxDListFindByIndex (const OSRTDList *pList, OSUINT32 index)
  This function will return the node pointer of the indexed entry in the list.

• EXTERNRT OSRTDListNode * rtxDListFindByData (const OSRTDList *pList, void *data)
  This function will return the node pointer of the given data item within the list or NULL if the item is not found.

• EXTERNRT int rtxDListFindIndexByData (const OSRTDList *pList, void *data)
  This function will return the index of the given data item within the list or -1 if the item is not found.

• EXTERNRT void rtxDListFreeNode (struct OSCTXT *pctxt, OSRTDList *pList, OSRTDListNode *node)
  This function will remove the given node from the list and free memory.
• EXTERNRT void rtxDListRemove (OSRTDList *pList, OSRTDListNode *node)
   This function will remove the given node from the list.

• EXTERNRT void rtxDListFreeNodes (struct OSCTXT *pctxt, OSRTDList *pList)
   This function will free all of the dynamic memory used to hold the list node pointers.

• EXTERNRT void rtxDListFreeAll (struct OSCTXT *pctxt, OSRTDList *pList)
   This function will free all of the dynamic memory used to hold the list node pointers and the data items.

• EXTERNRT int rtxDListToArray (struct OSCTXT *pctxt, OSRTDList *pList, void **ppArray, OSUINT32 *pElemCount, size_t elemSize)
   This function converts a doubly linked list to an array.

• EXTERNRT int rtxDListAppendArray (struct OSCTXT *pctxt, OSRTDList *pList, void *pArray, OSUINT32 numElements, size_t elemSize)
   This function appends pointers to items in the given array to a doubly linked list structure.

• EXTERNRT int rtxDListAppendArrayCopy (struct OSCTXT *pctxt, OSRTDList *pList, const void *pArray, OSUINT32 numElements, size_t elemSize)
   This function appends a copy of each item in the given array to a doubly linked list structure.

• EXTERNRT int rtxDListToUTF8Str (struct OSCTXT *pctxt, OSRTDList *pList, OSUTF8CHAR **ppstr, char sep)
   This function concatenates all of the components in the given list to form a UTF-8 string.

7.16.1 Detailed Description

Doubly-Linked List Utility Functions.
Definition in file rtxDList.h.
7.17  rtxDynBitSet.h File Reference

- Implementation of a dynamic bit set similar to the Java BitSet class.

```
#include "rtxsrcreference/rtxBitString.h"
```

Functions

- EXTERNRT int rtxDynBitSetInit (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT16 segNBytes)
  
  This function initializes a dynamic bit set structure.

- EXTERNRT void rtxDynBitSetFree (OSCTXT *pctxt, OSRTDynBitSet *pbitset)
  
  This function frees dynamic memory held by the bit set.

- EXTERNRT int rtxDynBitSetCopy (OSCTXT *pctxt, const OSRTDynBitSet *pSrcBitSet, OSRTDynBitSet *pDestBitSet)
  
  This function creates a deep copy of the given bit set.

- EXTERNRT int rtxDynBitSetSetBit (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT32 index)
  
  This function sets the bit at the given index.

- EXTERNRT int rtxDynBitSetClearBit (OSRTDynBitSet *pbitset, OSUINT32 index)
  
  This function clears the bit at the given index.

- EXTERNRT OSBOOL rtxDynBitSetTestBit (const OSRTDynBitSet *pbitset, OSUINT32 index)
  
  This function tests the bit at the given index.

- EXTERNRT OSBOOL rtxDynBitSetSetBitToValue (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT32 index, OSBOOL value)
  
  This function sets the bit at the given index to the give value.

- EXTERNRT int rtxDynBitSetInsertBit (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT32 index, OSBOOL value)
  
  This function inserts a bit with the given value at the given index.

7.17.1 Detailed Description

- Implementation of a dynamic bit set similar to the Java BitSet class.

Definition in file rtxDynBitSet.h.

7.17.2 Function Documentation

7.17.2.1 EXTERNRT int rtxDynBitSetClearBit (OSRTDynBitSet * pbitset, OSUINT32 index)

This function clears the bit at the given index.

The bit set will be not be expanded if the given index is outside the currently allocated range. The bit will be assumed to already be clear since it is undefined.
Parameters:

\textit{pbitset} Pointer to bit set structure.
\textit{index} Index of bit to be clear.

Returns:

Status of operation: zero if success or a negative error code if failure.

7.17.2.2 \textbf{EXTERNRT int rtxDynBitSetCopy (OSCTXT \textit{*pctxt}, const OSRTDynBitSet \textit{*pSrcBitSet}, OSRTDynBitSet \textit{*pDestBitSet})}

This function creates a deep copy of the given bit set.

Parameters:

\textit{pctxt} Pointer to a context structure.
\textit{pSrcBitSet} Pointer to bit set structure to be copied.
\textit{pDestBitSet} Pointer to bit set structure to recieve copied data.

Returns:

Status of operation: zero if success or a negative error code if failure.

7.17.2.3 \textbf{EXTERNRT void rtxDynBitSetFree (OSCTXT \textit{*pctxt}, OSRTDynBitSet \textit{*pbitset})}

This function frees dynamic memory held by the bit set.

Parameters:

\textit{pctxt} Pointer to a context structure.
\textit{pbitset} Pointer to bit set structure to be freed.

7.17.2.4 \textbf{EXTERNRT int rtxDynBitSetInit (OSCTXT \textit{*pctxt}, OSRTDynBitSet \textit{*pbitset}, OSUINT16 \textit{segNBytes})}

This function initializes a dynamic bit set structure. Memory is allocated for the initial segment.

Parameters:

\textit{pctxt} Pointer to a context structure.
\textit{pbitset} Pointer to bit set structure to be initialized.
\textit{segNBytes} Number of bytes per segment expansion. If zero, the default value is used.

Returns:

Status of operation: zero if success or a negative error code if failure.
7.17.2.5 EXTERNRT int rtxDynBitSetInsertBit (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT32 index, OSBOOL value)

This function inserts a bit with the given value at the given index. All other bits are shifted to the right one position. If the maximum set bit number is at the end of the allocated range, the set is expanded.

Parameters:
- *pctxt Pointer to a context structure.
- *pbitset Pointer to bit set structure.
- index Index of position where bit is to be inserted.
- value Boolean value of the bit.

Returns:
- Status of operation: zero if success or a negative error code if failure.

7.17.2.6 EXTERNRT int rtxDynBitSetSetBit (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT32 index)

This function sets the bit at the given index. The bit set will be expanded if the given index is outside the currently allocated range.

Parameters:
- *pctxt Pointer to a context structure.
- *pbitset Pointer to bit set structure.
- index Index of bit to be set.

Returns:
- Status of operation: zero if success or a negative error code if failure.

7.17.2.7 EXTERNRT int rtxDynBitSetSetBitToValue (OSCTXT *pctxt, OSRTDynBitSet *pbitset, OSUINT32 index, OSBOOL value)

This function sets the bit at the given index to the give value. The bit set will be expanded if the given index is outside the currently allocated range.

Parameters:
- *pctxt Pointer to a context structure.
- *pbitset Pointer to bit set structure.
- index Index of bit to be set.
- value Boolean value to which bit is to be set.

Returns:
- Status of operation: zero if success or a negative error code if failure.
7.17.2.8 EXTERN RT OSBOOL rtxDynBitSetTestBit (const OSRTDynBitSet * pbitset, OSUINT32 index)

This function tests the bit at the given index.

If the index is outside the range of the currently allocated set, the bit is assumed to be clear; otherwise, the state of the bit in the set is tested.

**Parameters:**

- *pbitset* Pointer to bit set structure.
- *index* Index of bit to be tested.

**Returns:**

Boolean result: true if set; false if clear.
7.18  rtxDynPtrArray.h File Reference

- Implementation of a dynamic pointer array.

#include "rtxsrc/rtxContext.h"

Functions

- EXTERNRT int rtxDynPtrArrayInit (OSCTXT *pctxt, OSRTDynPtrArray *pArray, OSUINT16 initialSize)
  This function initializes a new dynamic pointer array structure.

- EXTERNRT int rtxDynPtrArrayAppend (OSCTXT *pctxt, OSRTDynPtrArray *pArray, void *ptr)
  This function adds a pointer to the end of the array and expands the array if necessary.

7.18.1  Detailed Description

- Implementation of a dynamic pointer array.

Definition in file rtxDynPtrArray.h.

7.18.2  Function Documentation

7.18.2.1  EXTERNRT int rtxDynPtrArrayAppend (OSCTXT *pctxt, OSRTDynPtrArray *pArray, void *ptr)

This function adds a pointer to the end of the array and expands the array if necessary.

Parameters:

  pctxt  Pointer to a context structure.
  pArray  Pointer to dynamic pointer array structure.
  ptr  Pointer to be added to the array.

Returns:

  Status of operation: zero if success or a negative error code if failure.

7.18.2.2  EXTERNRT int rtxDynPtrArrayInit (OSCTXT *pctxt, OSRTDynPtrArray *pArray, OSUINT16 initialSize)

This function initializes a new dynamic pointer array structure.

Memory is allocated for the initial capacity of pointers.

Parameters:

  pctxt  Pointer to a context structure.
  pArray  Pointer to dynamic pointer array structure.
**initialSize**  Initial capacity of the array. The size will doubled on each expansion. If zero is provided, a default size will be used.

**Returns:**

Status of operation: zero if success or a negative error code if failure.
7.19  rtxEnum.h File Reference

Common runtime types and functions for performing operations on enumerated data items.

#include "rtxsrc/rtxContext.h"

Functions

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

- EXTERNRT OSINT32 rtxLookupEnumByValue (OSINT32 value, const OSEnumItem enumTable[], size_t enumTableSize)
  
  Lookup enum by integer value.

- EXTERNRT int rtxTestNumericEnum (OSINT32 ivalue, const OSNumericEnumItem enumTable[], OSUINT16 enumTableSize)
  
  This function determines if the given numeric enumerated value is within the defined numeration set.

7.19.1 Detailed Description

Common runtime types and functions for performing operations on enumerated data items.

Definition in file rtxEnum.h.
7.20 **rtxErrCodes.h File Reference**

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

**Defines**

- `#define RT_OK 0
  Normal completion status.
- `#define RT_OK_FRAG 2
  Message fragment return status.
- `#define RTERR_BUFOVFLW -1
  Encode buffer overflow.
- `#define RTERR_ENDOFBUF -2
  Unexpected end-of-buffer.
- `#define RTERR_IDNOTFOU -3
  Expected identifier not found.
- `#define RTERR_INVENUM -4
  Invalid enumerated identifier.
- `#define RTERR_SETDUPL -5
  Duplicate element in set.
- `#define RTERR_SETMISRQ -6
  Missing required element in set.
- `#define RTERR_NOTINSET -7
  Element not in set.
- `#define RTERR_SEQOVFLW -8
  Sequence overflow.
- `#define RTERR_INVOPT -9
  Invalid option in choice.
- `#define RTERR_NOMEM -10
  No dynamic memory available.
- `#define RTERR_INVHEXS -11
  Invalid hexadecimal string.
- `#define RTERR_INVREAL -12
  Invalid real number value.
- `#define RTERR_STROVFLW -13
String overflow.

- #define RTERR_BADVALUE -14
  Bad value.

- #define RTERR_TOODEEP -15
  Nesting level too deep.

- #define RTERR_CONSVIO -16
  Constraint violation.

- #define RTERR_ENDOFFILE -17
  Unexpected end-of-file error.

- #define RTERR_INVUTF8 -18
  Invalid UTF-8 character encoding.

- #define RTERR_OUTOFBND -19
  Array index out-of-bounds.

- #define RTERR_INVPARAM -20
  Invalid parameter passed to a function of method.

- #define RTERR_INVFORMAT -21
  Invalid value format.

- #define RTERR_NOTINIT -22
  Context not initialized.

- #define RTERR_TOOBIG -23
  Value will not fit in target variable.

- #define RTERR_INVCHAR -24
  Invalid character.

- #define RTERR_XMLSTATE -25
  XML state error.

- #define RTERR_XMLPARSE -26
  XML parser error.

- #define RTERR_SEQORDER -27
  Sequence order error.

- #define RTERR_FILNOTFOU -28
  File not found.

- #define RTERR_READERR -29
  Read error.
• #define RTERR_WRITEERR -30
  Write error.

• #define RTERR_INVBASE64 -31
  Invalid Base64 encoding.

• #define RTERR_INVSOCKET -32
  Invalid socket.

• #define RTERR_INVATTR -33
  Invalid attribute.

• #define RTERR_REGEXP -34
  Invalid regular expression.

• #define RTERR_PATMATCH -35
  Pattern match error.

• #define RTERR_ATTRMISRQ -36
  Missing required attribute.

• #define RTERR_HOSTNOTFOU -37
  Host name could not be resolved.

• #define RTERR_HTTPERR -38
  HTTP protocol error.

• #define RTERR_SOAPERR -39
  SOAP error.

• #define RTERR_EXPIRED -40
  Evaluation license expired.

• #define RTERR_UNEXPELEM -41
  Unexpected element encountered.

• #define RTERR_INVOCCUR -42
  Invalid number of occurrences.

• #define RTERR_INVMSGBUF -43
  Invalid message buffer has been passed to decode or validate method.

• #define RTERR_DECELEMFAIL -44
  Element decode failed.

• #define RTERR_DECATTRFAIL -45
  Attribute decode failed.

• #define RTERR_STRMINUSE -46
  Stream in-use.
• #define RTERR_NULLPTR -47
  Null pointer.

• #define RTERR_FAILED -48
  General failure.

• #define RTERR_ATTRFIXEDVAL -49
  Attribute fixed value mismatch.

• #define RTERR_MULTIPLE -50
  Multiple errors occurred during an encode or decode operation.

• #define RTERR_NOTYPEINFO -51
  This error is returned when decoding a derived type definition and no information exists as to what type of data is in the element content.

• #define RTERR_ADDRINUSE -52
  Address already in use.

• #define RTERR_CONNRESET -53
  Remote connection was reset.

• #define RTERR_UNREACHABLE -54
  Network failure.

• #define RTERR_NOCONN -55
  Not connected.

• #define RTERR_CONNREFUSED -56
  Connection refused.

• #define RTERR_INVSOCKOPT -57
  Invalid option.

• #define RTERR_SOAPFAULT -58
  This error is returned when decoded SOAP envelope is fault message.

• #define RTERR_MARKNOTSUP -59
  This error is returned when an attempt is made to mark a stream position on a stream type that does not support it.

• #define RTERR_NOTSUPP -60
  Feature is not supported.

7.20.1 Detailed Description

List of numeric status codes that can be returned by common run-time functions and generated code. Definition in file rtxErrCodes.h.
7.21  rtxError.h File Reference

Error handling function and macro definitions.
#include "rtxs/rtxContext.h"
#include "rtxs/rtxErrCodes.h"

Defines

• #define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat, __FILE__, __LINE__)
  This macro is used to log a run-time error in the context.

• #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition, __LINE__, __FILE__); }
  This macro is used to check an assertion.

• #define OSRTCHECKPARAM(condition) if (condition) { /* do nothing */ } 
  This macro check a condition but takes no action.

Functions

• EXTERNRT OSBOOL rtxErrAddCtxtBufParm (OSCTXT *pctxt)
  This function adds the contents of the context buffer to the error information structure in the context.

• EXTERNRT OSBOOL rtxErrAddDoubleParm (OSCTXT *pctxt, double errParm)
  This function adds a double parameter to an error information structure.

• EXTERNRT OSBOOL rtxErrAddErrorTableEntry (const char *const ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)
  This function adds a set of error codes to the global error table.

• EXTERNRT OSBOOL rtxErrAddElemNameParm (OSCTXT *pctxt)
  This function adds an element name parameter to the context error information structure.

• EXTERNRT OSBOOL rtxErrAddIntParm (OSCTXT *pctxt, int errParm)
  This function adds an integer parameter to an error information structure.

• EXTERNRT OSBOOL rtxErrAddInt64Parm (OSCTXT *pctxt, OSINT64 errParm)
  This function adds a 64-bit integer parameter to an error information structure.

• EXTERNRT OSBOOL rtxErrAddStrParm (OSCTXT *pctxt, const char *pErrParm)
  This function adds a character string parameter to an error information structure.

• EXTERNRT OSBOOL rtxErrAddStrnParm (OSCTXT *pctxt, const char *pErrParm, size_t nchars)
  This function adds a given number of characters from a character string parameter to an error information structure.

• EXTERNRT OSBOOL rtxErrAddUniStrParm (OSCTXT *pctxt, const OSUNICHAR *pErrParm)
  This function adds a Unicode string parameter to an error information structure.
• EXTERNRT OSBOOL rtxErrAddUIntParm (OSCTX *pctxt, unsigned int errParm)

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

• EXTERNRT OSBOOL rtxErrAddUInt64Parm (OSCTX *pctxt, OSUINT64 errParm)

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

• EXTERNRT void rtxErrAssertionFailed (const char *conditionText, int lineNo, const char *fileName)

This function is used to record an assertion failure.

• EXTERNRT const char * rtxErrFmtMsg (OSRTErrInfo *pErrInfo, char *bufp, size_t bufsiz)

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

• EXTERNRT void rtxErrFreeParms (OSCTX *pctxt)

This function is used to free dynamic memory that was used in the recording of error parameters.

• EXTERNRT char * rtxErrGetText (OSCTX *pctxt, char *pBuf, size_t *pBufSize)

This function returns error text in a memory buffer.

• EXTERNRT char * rtxErrGetTextBuf (OSCTX *pctxt, char *pbuf, size_t bufsiz)

This function returns error text in the given fixed-size memory buffer.

• EXTERNRT OSRTErrInfo * rtxErrNewNode (OSCTX *pctxt)

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

• EXTERNRT void rtxErrInit ()

This function is a one-time initialization function that must be called before any other error processing functions can be called.

• EXTERNRT int rtxErrReset (OSCTX *pctxt)

This function is used to reset the error state recorded in the context to successful.

• EXTERNRT void rtxErrLogUsingCB (OSCTX *pctxt, OSErrCbFunc cb, void *cbArg_p)

This function allows error information to be logged using a user-defined callback routine.

• EXTERNRT void rtxErrPrint (OSCTX *pctxt)

This function is used to print the error information stored in the context to the standard output device.

• EXTERNRT void rtxErrPrintElement (OSRTErrInfo *pErrInfo)

This function is used to print the error information stored in the error information element to the standard output device.

• EXTERNRT int rtxErrSetData (OSCTX *pctxt, int status, const char *module, int lineno)

This function is used to record an error in the context structure.

• EXTERNRT int rtxErrSetNewData (OSCTX *pctxt, int status, const char *module, int lineno)

This function is used to record an error in the context structure.

• EXTERNRT int rtxErrGetFirstError (const OSCTX *pctxt)

This function returns the error code, stored in the first error record.
• EXTERNRT int rtxErrGetLastError (const OSCTXT *pctxt)
  "This function returns the error code, stored in the last error record."

• EXTERNRT OSUINT32 rtxErrGetErrorCnt (const OSCTXT *pctxt)
  "This function returns the total number of error records."

• EXTERNRT int rtxErrGetStatus (const OSCTXT *pctxt)
  "This function returns the status value from the context."

• EXTERNRT int rtxErrResetLastErrors (OSCTXT *pctxt, int errorsToReset)
  "This function resets last 'errorsToReset' errors in the context."

7.21.1 Detailed Description
Error handling function and macro definitions.
Definition in file rtxError.h.
7.22 rtxExternDefs.h File Reference

Common definitions of external function modifiers used to define the scope of functions used in DLL’s (Windows only).

7.22.1 Detailed Description

Common definitions of external function modifiers used to define the scope of functions used in DLL’s (Windows only).

Definition in file rtxExternDefs.h.
7.23  rtxFile.h File Reference

Common runtime functions for reading from or writing to files.

#include "rtxsrctxtxContext.h"

Functions

• EXTERNRT int rtxFileReadBinary (OSCTXT *pctxt, const char *filePath, OSOCTET **ppMsgBuf, size_t *pLength)

  This function reads the entire contents of a binary file into memory.

• EXTERNRT int rtxFileReadText (OSCTXT *pctxt, const char *filePath, OSOCTET **ppMsgBuf, size_t *pLength)

  This function reads the entire contents of an ASCII text file into memory.

• EXTERNRT int rtxFileWriteBinary (const char *filePath, const OSOCTET *pMsgBuf, size_t length)

  This function writes binary data from memory to the given file.

• EXTERNRT int rtxFileWriteText (const char *filePath, const char *pMsgBuf)

  This function writes text data from memory to the given file.

7.23.1  Detailed Description

Common runtime functions for reading from or writing to files.

Definition in file rtxFile.h.

7.23.2  Function Documentation

7.23.2.1  EXTERNRT int rtxFileReadBinary (OSCTXT * pctxt, const char * filePath, OSOCTET ** ppMsgBuf, size_t * pLength)

This function reads the entire contents of a binary file into memory.

A memory buffer is allocated for the file contents using the run-time memory management functions.

Parameters:

  pctxt  Pointer to context block structure.
  filePath  Complete file path name of file to read.
  ppMsgBuf  Pointer to message buffer to receive allocated memory pointer.
  pLength  Pointer to integer to receive length of data read.

Returns:

  Completion status of operation:
  • 0 (ASN_OK) = success,
  • RTERR_FILNOTFOU = file not found
  • RTERR_FILEREAD = file read error (see errno)
7.23.2.2 EXTERNRT int rtxFileReadText (OSCTXT * pctxt, const char * filePath, OSOCTET ** ppMsgBuf, size_t * pLength)

This function reads the entire contents of an ASCII text file into memory. A memory buffer is allocated for the file contents using the run-time memory management functions. This function is identical to rtxReadFileBinary except that a) the file is opened in text mode, and b) and extra byte is allocated at the end for a null-terminator character.

Parameters:

- pctxt Pointer to context block structure.
- filePath Complete file path name of file to read.
- ppMsgBuf Pointer to message buffer to receive allocated memory pointer.
- pLength Pointer to integer to receive length of data read.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- RTERR_FILENOTFOU = file not found
- RTERR_FILEREAD = file read error (see errno)

7.23.2.3 EXTERNRT int rtxFileWriteBinary (const char * filePath, const OSOCTET * pMsgBuf, size_t length)

This function writes binary data from memory to the given file.

Parameters:

- filePath Complete file path name of file to be written to.
- pMsgBuf Pointer to buffer containing data to be written.
- length Size (in bytes) of data to be written

Returns:

Completion status of operation:

- 0 = success,
- negative status code if error

7.23.2.4 EXTERNRT int rtxFileWriteText (const char * filePath, const char * pMsgBuf)

This function writes text data from memory to the given file. The text is expected to be terminated by a null terminator character. This function will work with standard ASCII or UTF-8 encoded text.

Parameters:

- filePath Complete file path name of file to be written to.
- pMsgBuf Pointer to buffer containing data to be written.
Returns:
Completion status of operation:
• 0 = success,
• negative status code if error
7.24 rtxFloat.h File Reference

#include "rtxsrsrc/rtxCommon.h"

7.24.1 Detailed Description

Definition in file rtxFloat.h.
7.25  rtxHashMap.h File Reference

Generic hash map interface.

#include "rtxsctx/rtxContext.h"

Functions

- EXTERNRT void HASHMAPINITFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap, size_t capacity, OSUINT32(*hashFunc)(HASHMAPKEYTYPE), OSBOOL(*keyEqualsFunc)(HASHMAPKEYTYPE, HASHMAPKEYTYPE))
  
  This function initializes the hash map.

- EXTERNRT HASHMAPTYPE NAME * HASHMAPNEWFUNC (OSCTXT *ctxt, size_t capacity, OSUINT32(*hashFunc)(HASHMAPKEYTYPE), OSBOOL(*keyEqualsFunc)(HASHMAPKEYTYPE, HASHMAPKEYTYPE))
  
  This function creates a new hash map.

- EXTERNRT HASHMAPTYPE NAME * HASHMAPCOPYFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap)
  
  This function creates a copy of an existing hash map.

- EXTERNRT void HASHMAPFREEFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap)
  
  This function frees all entries within an existing hash map.

- EXTERNRT int HASHMAPINSERTFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE value)
  
  This function inserts an entry into the hash map.

- EXTERNRT OSBOOL HASHMAPSEARCHFUNC (HASHMAPTYPE NAME *pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE *pvalue)
  
  This function searches for an entry in the hash map.

- EXTERNRT OSBOOL HASHMAPREMOVEFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE *pvalue)
  
  This function removes an entry from the hash map.

- EXTERNRT int HASHMAPPUTFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE value)
  
  This function inserts/replaces an entry into the hash map.

- EXTERNRT int HASHMAPSORTFUNC (OSCTXT *ctxt, HASHMAPTYPE NAME *pHashMap, OSRTDList *pSortedList, int(*compareFunc)(HASHMAPKEYTYPE key1, HASHMAPKEYTYPE key2))
  
  This function sorts the hash map in ascending order using the given key compare function.

7.25.1  Detailed Description

Generic hash map interface.
This relates a generic key structure (void*) to a generic value (void*). Based on "C Hash Table" public domain code (http://www.cl.cam.ac.uk/~cwc22/hashtable/).

Definition in file rtxHashMap.h.

7.25.2 Function Documentation

7.25.2.1 EXTERNRT HASHMAPTYPENAME* HASHMAPCOPYFUNC (OSCTXT * pctxt, HASHMAPTYPENAME * pHashMap)

This function creates a copy of an existing hash map.

Parameters:

    pctxt  Pointer to a context structure.
    pHashMap  Pointer to hash map structure to copy.

Returns:

    Allocated and copied hash map structure or NULL if insufficient dynamic memory is available to hold the structure.

7.25.2.2 EXTERNRT void HASHMAPFREEFUNC (OSCTXT * pctxt, HASHMAPTYPENAME * pHashMap)

This function frees all entries within an existing hash map.

It does not free the structure itself.

Parameters:

    pctxt  Pointer to a context structure.
    pHashMap  Pointer to hash map structure to free.

7.25.2.3 EXTERNRT void HASHMAPINITFUNC (OSCTXT * pctxt, HASHMAPTYPENAME * pHashMap, size_t capacity, OSUINT32(*)(HASHMAPKEYTYPE) hashFunc, OSBOOL(*)(HASHMAPKEYTYPE, HASHMAPKEYTYPE) keyEqualsFunc)

This function initializes the hash map.

Parameters:

    pctxt  Pointer to a context structure.
    pHashMap  Pointer to hash map structure.
    capacity  Capacity of the hash map or zero to use default.
    hashFunc  Hash callback function.
    keyEqualsFunc  Key equals callback function.
7.25.2.4 EXTERNRT int HASHMAPINSERTFUNC (OSCTXT * pctxt, HASHMAPTYPENAME * pHashMap, HASHMAPKEYTYPE key, HASHMAPVALETYPE value)

This function inserts an entry into the hash map.

The table will be expanded if the insertion would take the ratio of entries to table size over the maximum load factor.

This function does not check for repeated insertions with a duplicate key. The value returned when using a duplicate key is undefined – when the hashtable changes size, the order of retrieval of duplicate key entries is reversed. If in doubt, remove before insert.

Parameters:
- `pctxt` Pointer to a context structure.
- `pHashMap` Pointer to hash map structure.
- `key` Key value. Memory is owned by caller.
- `value` Value to insert. Memory is owned by caller.

Returns:
- Zero if insertion was successful, a negative status code otherwise.

7.25.2.5 EXTERNRT HASHMAPTYPENAME * HASHMAPNEWFUNC (OSCTXT * pctxt, size_t capacity, OSUINT32(*)(HASHMAPKEYTYPE) hashFunc, OSBOOL(*)(HASHMAPKEYTYPE, HASHMAPKEYTYPE) keyEqualsFunc)

This function creates a new hash map.

Parameters:
- `pctxt` Pointer to a context structure.
- `capacity` Capacity of the map or zero to use default.
- `hashFunc` Hash callback function.
- `keyEqualsFunc` Key equals callback function.

Returns:
- Allocated hash map structure or NULL if insufficient dynamic memory is available to hold the structure.

7.25.2.6 EXTERNRT int HASHMAPPUTFUNC (OSCTXT * pctxt, HASHMAPTYPENAME * pHashMap, HASHMAPKEYTYPE key, HASHMAPVALETYPE value)

This function inserts/replaces an entry into the hash map.

If the key already exists in the map, its value is updated. Otherwise, the key/value pair is inserted.

Parameters:
- `pctxt` Pointer to a context structure.
- `pHashMap` Pointer to hash map structure.
- `key` Key value. Memory is owned by caller.
- `value` Value to insert/replace. Memory is owned by caller.

Returns:
- Zero if operation was successful, a negative status code otherwise.
7.25.2.7 EXTERNRT OSBOOL HASHMAPREMOVEFUNC (OSCTXT ∗ pctxt, HASHMAPTYPENAME ∗ pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE ∗ pvalue)

This function removes an entry from the hash map.

Parameters:
- `pctxt` Pointer to a context structure.
- `pHashMap` Pointer to hash map structure.
- `key` Key value. Memory is owned by caller.
- `pvalue` Pointer to value to receive search result value.

Returns:
Boolean result: true if found and removed.

7.25.2.8 EXTERNRT OSBOOL HASHMAPSEARCHFUNC (HASHMAPTYPENAME ∗ pHashMap, HASHMAPKEYTYPE key, HASHMAPVALUETYPE ∗ pvalue)

This function searches for an entry in the hash map.

Parameters:
- `pHashMap` Pointer to hash map structure.
- `key` Key value. Memory is owned by caller.
- `pvalue` Pointer to value to receive search result value.

Returns:
Boolean search result: true if found; false if not.

7.25.2.9 EXTERNRT int HASHMAPSORTFUNC (OSCTXT ∗ pctxt, HASHMAPTYPENAME ∗ pHashMap, OSRTDList ∗ pSortedList, int(*)(HASHMAPKEYTYPE key1, HASHMAPKEYTYPE key2) compareFunc)

This function sorts the hash map in ascending order using the given key compare function.

Parameters:
- `pctxt` Pointer to a context structure.
- `pHashMap` Pointer to hash map structure.
- `compareFunc` Comparison function for key values.
- `pSortedList` Pointer to linked list structure to receive sorted values. Entries within the list are items from the hash map themselves, not copies. List memory may be freed by calling rtxDListFreeNodes.

Returns:
Status of operation: 0 = success or negative status code.
7.26  rtxHashMapStr2Int.h File Reference

String-to-integer hash map interface.

#include "rtxs/rtxHashMap.h"

7.26.1  Detailed Description

String-to-integer hash map interface.

This relates a STRING key structure (const OSUTF8CHAR*) to a 32-bit signed integer value (OSINT32). It uses the rtxHashMap.h/.c file as a template.

Definition in file rtxHashMapStr2Int.h.
7.27 rtxHashMapStr2UInt.h File Reference

String-to-unsigned integer hash map interface.

```
#include "rtxsrl/rtxHashMap.h"
```

7.27.1 Detailed Description

String-to-unsigned integer hash map interface.

This relates a string key structure (const OSUTF8CHAR*) to a 32-bit unsigned integer value (OSUINT32). It uses the rtxHashMap .h/.c file as a template.

Definition in file rtxHashMapStr2UInt.h.
7.28  rtxHashMapUndef.h File Reference

Undefine all hash map symbols to allow reuse of the basic definitions in a different of the map.

7.28.1  Detailed Description

Undefine all hash map symbols to allow reuse of the basic definitions in a different of the map. 
Definition in file rtxHashMapUndef.h.
7.29  rtxMemBuf.h File Reference

#include "rtxsrc/rtxContext.h"

Functions

- EXTERNRT int rtxMemBufAppend (OSRTMEMBUF *pMemBuf, const OSOCTET *pdata, size_t nbytes)
  
  This function appends the data to the end of a memory buffer.

- EXTERNRT int rtxMemBufCut (OSRTMEMBUF *pMemBuf, size_t fromOffset, size_t nbytes)
  
  This function cuts off the part of memory buffer.

- EXTERNRT void rtxMemBufFree (OSRTMEMBUF *pMemBuf)
  
  This function frees the memory buffer.

- EXTERNRT OSOCTET * rtxMemBufGetData (OSRTMEMBUF *pMemBuf, int *length)
  
  This function returns the pointer to the used part of a memory buffer.

- EXTERNRT int rtxMemBufGetDataLen (OSRTMEMBUF *pMemBuf)
  
  This function returns the length of the used part of a memory buffer.

- EXTERNRT void rtxMemBufInit (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, size_t segsize)
  
  This function initializes a memory buffer structure.

- EXTERNRT void rtxMemBufInitBuffer (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, OSOCTET *buf, size_t bufsize, size_t segsize)
  
  This function assigns a static buffer to the memory buffer structure.

- EXTERNRT int rtxMemBufPreAllocate (OSRTMEMBUF *pMemBuf, size_t nbytes)
  
  This function allocates a buffer with a predetermined amount of space.

- EXTERNRT void rtxMemBufReset (OSRTMEMBUF *pMemBuf)
  
  This function resets the memory buffer structure.

- EXTERNRT int rtxMemBufSet (OSRTMEMBUF *pMemBuf, OSOCTET value, size_t nbytes)
  
  This function sets part of a memory buffer to a specified octet value.

- EXTERNRT OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF *pMemBuf, OSBOOL isExpandable)
  
  This function sets "isExpandable" flag for the memory buffer object.

- EXTERNRT int rtxMemBufTrimW (OSRTMEMBUF *pMemBuf)
  
  This function trims white space of the memory buffer.

7.29.1  Detailed Description

Definition in file rtxMemBuf.h.
7.30 rtxMemory.h File Reference

Memory management function and macro definitions.

#include "rtxsrc/rtxContext.h"

Defines

• #define RT_MH_DONTKEEPFREE 0x1
  Uncomment this definition before building the C or C++ run-time libraries to enable compact memory management.

• #define OSRTALLOCTYPE(pctxt, type) (type*) rtxMemHeapAlloc (&(pctxt)→pMemHeap, sizeof(type))
  This macro allocates a single element of the given type.

• #define OSRTALLOCTYPEZ(pctxt, type) (type*) rtxMemHeapAllocZ (&(pctxt)→pMemHeap, sizeof(type))
  This macro allocates and zeros a single element of the given type.

• #define rtxMemAllocType(pctxt, ctype) (ctype*) rtxMemHeapAlloc (&(pctxt)→pMemHeap, sizeof(ctype))
  Allocate type.

• #define rtxMemAllocTypeZ(pctxt, ctype) (ctype*) rtxMemHeapAllocZ (&(pctxt)→pMemHeap, sizeof(ctype))
  Allocate type and zero memory.

• #define rtxMemAllocArray(pctxt, n, type) (type*) rtxMemHeapAlloc (&(pctxt)→pMemHeap, sizeof(type)*n)
  Allocate a dynamic array.

• #define rtxMemNewAutoPtr(pctxt, nbytes) rtxMemHeapAlloc (&(pctxt)→pMemHeap, nbytes)
  This function allocates a new block of memory and creates an auto-pointer with reference count set to one.

• #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef (&(pctxt)→pMemHeap, (void*)ptr)
  This function increments the auto-pointer reference count.

• #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref (&(pctxt)→pMemHeap, (void*)ptr)
  This function decrements the auto-pointer reference count.

• #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount (&(pctxt)→pMemHeap, (void*)ptr)
  This function returns the reference count of the given pointer.

Functions

• EXTERNRT void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
  This function sets the pointers to standard allocation functions.

• EXTERNRT void rtxMemSetDefBlkSize (OSUINT32 blkSize)
This function sets the minimum size and the granularity of memory blocks for newly created memory heaps.

- EXTERNRT OSUINT32 rtxMemGetDefBlkSize ()
  
  This function returns the actual granularity of memory blocks.

- EXTERNRT OSBOOL rtxMemIsZero (const void *pmem, size_t memsiz)
  
  This helper function determines if an arbitrarily sized block of memory is set to zero.

7.30.1 Detailed Description

Memory management function and macro definitions.
Definition in file rtxMemory.h.

7.30.2 Define Documentation

7.30.2.1 #define RT_MH_DONTKEEPFREE 0x1

Uncomment this definition before building the C or C++ run-time libraries to enable compact memory management. This will have a smaller code footprint than the standard memory management, however, the performance may not be as good.
Definition at line 41 of file rtxMemory.h.
Pattern matching functions.

```c
#include "rtxs/rtxContext.h"
```

### Functions

- EXTERNRT OSBOOL rtxMatchPattern (OSCTX pctxt, const OSUTF8CHAR *text, const OSUTF8CHAR *pattern)

  *This function compares the given string to the given pattern.*

### 7.31.1 Detailed Description

Pattern matching functions.

Definition in file `rtxPattern.h`.

#include <stdio.h>
#include "rtxsrs/osSysTypes.h"
#include "rtxsrs/rtxExternDefs.h"
#include "rtxsrs/rtxDList.h"

## Functions

- **EXTERNRT int rtxByteToHexChar** (OSOCTET byte, char *buf, size_t bufsize)
  
  This function converts a byte value into its hex string equivalent.

- **EXTERNRT void rtxPrintBoolean** (const char *name, OSBOOL value)
  
  Prints a boolean value to stdout.

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

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

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

- **EXTERNRT void rtxPrintInteger** (const char *name, OSINT32 value)
  
  Prints an integer value to stdout.

- **EXTERNRT void rtxPrintInt64** (const char *name, OSINT64 value)
  
  Prints a 64-bit integer value to stdout.

- **EXTERNRT void rtxPrintUnsigned** (const char *name, OSUINT32 value)
  
  Prints an unsigned integer value to stdout.

- **EXTERNRT void rtxPrintUInt64** (const char *name, OSUINT64 value)
  
  Prints an unsigned 64-bit integer value to stdout.

- **EXTERNRT void rtxPrintHexStr** (const char *name, OSUINT32 numocts, const OSOCTET *data)
  
  This function prints the value of a binary string in hex format to standard output.

- **EXTERNRT void rtxPrintHexBinary** (const char *name, OSUINT32 numocts, const OSOCTET *data)
  
  Prints an octet string value in hex binary format to stdout.

- **EXTERNRT void rtxPrintCharStr** (const char *name, const char *cstring)
  
  Prints an ASCII character string value to stdout.

- **EXTERNRT void rtxPrintUTF8CharStr** (const char *name, const OSUTF8CHAR *cstring)
  
  Prints a UTF-8 encoded character string value to stdout.

- **EXTERNRT void rtxPrintUnicodeCharStr** (const char *name, const OSUNICHAR *str, int nchars)
This function prints a Unicode string to standard output.

- EXTERNRT void rtxPrintReal (const char *name, OSREAL value)
  
  Prints a REAL (float, double, decimal) value to stdout.

- EXTERNRT void rtxPrintNull (const char *name)

  Prints a NULL value to stdout.

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

  Prints a name-value pair to stdout.

- EXTERNRT int rtxPrintFile (const char *filename)

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

- EXTERNRT void rtxPrintIndent (void)

  This function prints indentation spaces to stdout.

- EXTERNRT void rtxPrintIncrIndent (void)

  This function increments the current indentation level.

- EXTERNRT void rtxPrintDecrIndent (void)

  This function decrements the current indentation level.

- EXTERNRT void rtxPrintCloseBrace (void)

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

- EXTERNRT void rtxPrintOpenBrace (const char *)

  This function opens a braced region by printing indent spaces, printing the name and opening brace, and increasing the indent level.

- EXTERNRT void rtxHexDumpToNamedFile (const char *filename, const OSOCTET *data, OSUINT32 numocts)

  This function outputs a hexadecimal dump of the current buffer contents to the file with the given name.

- EXTERNRT void rtxHexDumpToFile (FILE *fp, const OSOCTET *data, OSUINT32 numocts)

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

- EXTERNRT void rtxHexDumpToFileEx (FILE *fp, const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)

  This function outputs a hexadecimal dump of the current buffer to a file, but it may output the dump as an array of bytes, words, or double words.

- EXTERNRT void rtxHexDump (const OSOCTET *data, OSUINT32 numocts)

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

- EXTERNRT void rtxHexDumpEx (const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)

  This function outputs a hexadecimal dump of the current buffer contents to stdout, but it may display the dump as an array or bytes, words, or double words.
• EXTERNRT int rtxHexDumpToString (const OSOCTET *data, OSUINT32 numocts, char *buffer, int bufferIndex, int bufferSize)
  This function formats a hexadecimal dump of the current buffer contents to a string.

• EXTERNRT int rtxHexDumpToStringEx (const OSOCTET *data, OSUINT32 numocts, char *buffer, int bufferIndex, int bufferSize, int bytesPerUnit)
  This function formats a hexadecimal dump of the current buffer contents to a string. But it may output the dump as an array of bytes, words, or double words.

7.32.1 Detailed Description

Definition in file rtxPrint.h.
7.33  rtxPrintStream.h File Reference

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

```c
#include <stdarg.h>
#include "rtxsoc/rtxContext.h"
```

Classes

- struct OSRTPrintStream
  
  Structure to hold information about a global PrintStream.

Typedefs

- typedef void(*rtxPrintCallback)(void*pPrntStrmInfo, const char*fmtspec, va_list arglist)
  
  Callback function definition for print stream.

Functions

- EXTERNRT int rtxSetPrintStream(OSCTXT*pctxt, rtxPrintCallback myCallback, void*pStrmInfo)
  
  This function is for setting the callback function for a PrintStream.

- EXTERNRT int rtxSetGlobalPrintStream(rtxPrintCallback myCallback, void*pStrmInfo)
  
  This function is for setting the callback function for a PrintStream.

- EXTERNRT int rtxPrintToStream(OSCTXT*pctxt, const char*fmtspec,...)
  
  Print-to-stream function which in turn calls the user registered callback function of the context for printing.

- EXTERNRT int rtxDiagToStream(OSCTXT*pctxt, const char*fmtspec, va_list arglist)
  
  Diagnostics print-to-stream function.

- EXTERNRT int rtxPrintStreamRelease(OSCTXT*pctxt)
  
  This function releases the memory held by PrintStream in the context.

Variables

- OSRTPrintStream g_PrintStream
  
  Global PrintStream.

7.33.1 Detailed Description

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

Definition in file rtxPrintStream.h.
# include <stdio.h>
#include "rtxs/rtxContext.h"

## Functions

- **EXTERNRT void rtxPrintToStreamBoolean (OSCTXT *pctxt, const char *name, OSBOOL value)**
  
  Prints a boolean value to a print stream.

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

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

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

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

- **EXTERNRT void rtxPrintToStreamInt64 (OSCTXT *pctxt, const char *name, OSINT64 value)**
  
  Prints a 64-bit integer value to a print stream.

- **EXTERNRT void rtxPrintToStreamUnsigned (OSCTXT *pctxt, const char *name, OSUINT32 value)**
  
  Prints an unsigned integer value to a print stream.

- **EXTERNRT void rtxPrintToStreamUInt64 (OSCTXT *pctxt, const char *name, OSUINT64 value)**
  
  Prints an unsigned 64-bit integer value to a print stream.

- **EXTERNRT void rtxPrintToStreamHexStr (OSCTXT *pctxt, const char *name, OSUINT32 numocts, const OSOCTET *data)**
  
  This function prints the value of a binary string in hex format to standard output.

- **EXTERNRT void rtxPrintToStreamHexBinary (OSCTXT *pctxt, const char *name, OSUINT32 numocts, const OSOCTET *data)**
  
  Prints an octet string value in hex binary format to a print stream.

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

- **EXTERNRT void rtxPrintToStreamUTF8CharStr (OSCTXT *pctxt, const char *name, const OSUTF8CHAR *cstring)**
  
  Prints a UTF-8 encoded character string value to a print stream.
• EXTERNRT void rtxPrintToStreamUnicodeCharStr (OSCTX† *pctxt, const char *name, const OSUNICHAR *str, int nchars)
  This function prints a Unicode string to standard output.

• EXTERNRT void rtxPrintToStreamReal (OSCTX† *pctxt, const char *name, OSREAL value)
  Prints a REAL (float, double, decimal) value to a print stream.

• EXTERNRT void rtxPrintToStreamNull (OSCTX† *pctxt, const char *name)
  Prints a NULL value to a print stream.

• EXTERNRT void rtxPrintToStreamNVP (OSCTX† *pctxt, const char *name, const OSUTF8NVP *value)
  Prints a name-value pair to a print stream.

• EXTERNRT int rtxPrintToStreamFile (OSCTX† *pctxt, const char *filename)
  This function prints the contents of a text file to a print stream.

• EXTERNRT void rtxPrintToStreamIndent (OSCTX† *pctxt)
  This function prints indentation spaces to a print stream.

• EXTERNRT void rtxPrintToStreamIncrIndent (void)
  This function increments the current indentation level.

• EXTERNRT void rtxPrintToStreamDecrIndent (void)
  This function decrements the current indentation level.

• EXTERNRT void rtxPrintToStreamCloseBrace (OSCTX† *pctxt)
  This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.

• EXTERNRT void rtxPrintToStreamOpenBrace (OSCTX† *pctxt, const char *)
  This function opens a braced region by printing indent spaces, printing the name and opening brace, and increasing the indent level.

• EXTERNRT void rtxHexDumpToStream (OSCTX† *pctxt, const OSOCTET *data, OSUINT32 numocts)
  This function outputs a hexadecimal dump of the current buffer contents to a print stream.

• EXTERNRT void rtxHexDumpToStreamEx (OSCTX† *pctxt, const OSOCTET *data, OSUINT32 numocts, int bytesPerUnit)
  This function outputs a hexadecimal dump of the current buffer to a print stream, but it may output the dump as an array of bytes, words, or double words.

### 7.34.1 Detailed Description

Definition in file rtxPrintToStream.h.
7.35 rtxReal.h File Reference

Common runtime functions for working with floating-point numbers.
#include "rtxs src/osSysTypes.h"
#include "rtxs src/rtxExternDefs.h"

Functions

- EXTERNRT OSREAL rtxGetMinusInfinity (void)
  Returns the IEEE negative infinity value.

- EXTERNRT OSREAL rtxGetMinusZero (void)
  Returns the IEEE minus zero value.

- EXTERNRT OSREAL rtxGetNaN (void)
  Returns the IEEE Not-A-Number (NaN) value.

- EXTERNRT OSREAL rtxGetPlusInfinity (void)
  Returns the IEEE positive infinity value.

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

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

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

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

7.35.1 Detailed Description

Common runtime functions for working with floating-point numbers.
Definition in file rtxReal.h.

228
7.36 rtxScalarDList.h File Reference

Doubly-linked list utility functions to hold scalar data variables.

```
#include "rtxs src/osSysTypes.h"
#include "rtxs rc/rtxExternDefs.h"
```

Classes

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

- `struct OSRTScalarDList`
  
  This is the main list structure.

Functions

- `EXTERNRT void rtxScalarDListInit (OSRTScalarDList *pList)`
  
  This function initializes a doubly linked list structure.

- `EXTERNRT OSRTScalarDListNode * rtxScalarDListAppendDouble (struct OSCTXT *pctxt, OSRTScalarDList *pList, OSDOUBLE value)`
  
  This set of functions appends an item of the given scalar type to the linked list structure.

- `EXTERNRT OSRTScalarDListNode * rtxScalarDListAppendNode (OSRTScalarDList *pList, OSRTScalarDListNode *pListNode)`
  
  This function is used to append a node to the linked list.

- `EXTERNRT OSRTScalarDListNode * rtxScalarDListInsertNode (OSRTScalarDList *pList, OSUINT32 index, OSRTScalarDListNode *pListNode)`
  
  This function is used to insert a node into the linked list.

- `EXTERNRT OSRTScalarDListNode * rtxScalarDListFindByIndex (const OSRTScalarDList *pList, OSUINT32 index)`
  
  This function will return the node pointer of the indexed entry in the list.

- `EXTERNRT void rtxScalarDListFreeNode (struct OSCTXT *pctxt, OSRTScalarDList *pList, OSRTScalarDListNode *node)`
  
  This function will remove the given node from the list and free memory.

- `EXTERNRT void rtxScalarDListRemove (OSRTScalarDList *pList, OSRTScalarDListNode *node)`
  
  This function will remove the given node from the list.

- `EXTERNRT void rtxScalarDListFreeNodes (struct OSCTXT *pctxt, OSRTScalarDList *pList)`
  
  This function will free all of the dynamic memory used to hold the list node pointers.
7.36.1 Detailed Description

Doubly-linked list utility functions to hold scalar data variables.
Definition in file rtxScalarDList.h.
7.37 rtxSOAP.h File Reference

: common SOAP socket communications functions
#include "rtxsSrc/rtxCommon.h"
#include "rtxsSrc/rtxSocket.h"

Functions

- EXTERNRT int rtxSoapInitConn (OSSOAPCONN *pSoapConn, OSCTXT *pctxt, OSSoapVersion soapv, const char *url)
  This function initializes a connection to a SOAP endpoint.

- EXTERNRT int rtxSoapAcceptConn (OSRTSOCKET listenSocket, OSSOAPCONN *pSoapConn)
  This function accepts an incoming connection request and sets up a stream on which to receive messages.

- EXTERNRT int rtxSoapConnect (OSSOAPCONN *pSoapConn)
  This function creates a connection to a SOAP endpoint.

- EXTERNRT int rtxSoapRecvHttp (OSSOAPCONN *pSoapConn)
  This function receives the initial header returned from an HTTP request.

- EXTERNRT int rtxSoapRecvHttpContent (OSSOAPCONN *pSoapConn, OSOCTET **ppbuf)
  This function receives a complete HTTP response from a SOAP connection.

- EXTERNRT int rtxSoapSendHttp (OSSOAPCONN *pSoapConn, const OSUTF8CHAR *soapMsg)
  This function sends a complete HTTP request to a SOAP connection.

7.37.1 Detailed Description

: common SOAP socket communications functions
Definition in file rtxSOAP.h.

7.37.2 Function Documentation

7.37.2.1 EXTERNRT int rtxSoapAcceptConn (OSRTSOCKET listenSocket, OSSOAPCONN *pSoapConn)

This function accepts an incoming connection request and sets up a stream on which to receive messages.

Parameters:

  listenSocket - Listener socket
  pSoapConn - Pointer to SOAP connection structure.

Returns:

  - Operation status: 0 if success, negative code if error.
7.37.2.2 EXTERNRT int rtxSoapConnect (OSSOAPCONN * pSoapConn)

This function creates a connection to a SOAP endpoint.

The endpoint is described by a SOAP connection structure which must have been initialized using the rtxSoapInitConn function.

Parameters:
   pSoapConn - Pointer to SOAP connection structure.

Returns:
   - Operation status: 0 if success, negative code if error.

7.37.2.3 EXTERNRT int rtxSoapInitConn (OSSOAPCONN * pSoapConn, OSCTXT * pctxt, OSSoapVersion soapv, const char * url)

This function initializes a connection to a SOAP endpoint.

Parameters:
   pSoapConn - Pointer to SOAP connection structure.
   pctxt - Pointer to an XBinder run-time context structure.
   soapv - SOAP version that is to be used.
   url - URL to which to connect.

Returns:
   - Operation status: 0 if success, negative code if error.

7.37.2.4 EXTERNRT int rtxSoapRecvHttp (OSSOAPCONN * pSoapConn)

This function receives the initial header returned from an HTTP request.

The header response information including content length and whether the response is 'chunked' is stored in the connection structure.

Parameters:
   pSoapConn - Pointer to SOAP connection structure.

Returns:
   - Operation status: 0 if success, negative code if error.

7.37.2.5 EXTERNRT int rtxSoapRecvHttpContent (OSSOAPCONN * pSoapConn, OSOCTET ** ppbuf)

This function receives a complete HTTP response from a SOAP connection.

The response if stored in a dynamic memory buffer which is returned via the buffer pointer argument. Memory is allocated for the response using XBinder memory management, so it will be freed when the context is freed or the rtxMemFree function is called. This buffer can now be used in a decode function call to parse the received XML message into a program structure.
Parameters:

- `pSoapConn` - Pointer to SOAP connection structure.
- `ppbuf` - Pointer to pointer to receive content buffer.

Returns:

- Operation status: 0 if success, negative code if error.

7.37.2.6 `EXTERNRT int rtxSoapSendHttp (OSSOAPCONN * pSoapConn, const OSUTF8CHAR * soapMsg)`

This function sends a complete HTTP request to a SOAP connection.

The request is stored in the XBinder context buffer. If an XML encode operation was just completed, the calling this function will send the encoded XML message to the SOAP endpoint.

Parameters:

- `pSoapConn` - Pointer to SOAP connection structure.
- `soapMsg` - SOAP XML message to be sent.

Returns:

- Operation status: 0 if success, negative code if error.
7.38  rtxSocket.h File Reference

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

Typedefs

- typedef int OSRTSOCKET
  socket handle

- typedef unsigned long OSIPADDR
  The IP address represented as unsigned long value.

Functions

- EXTERNRT int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET *pNewSocket, OSIPADDR *destAddr, int *destPort)
  This function permits an incoming connection attempt on a socket.

- EXTERNRT int rtxSocketAddrToStr (OSIPADDR ipAddr, char *pbuf, size_t bufsize)
  This function converts an IP address to its string representation.

- EXTERNRT int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)
  This function associates a local address with a socket.

- EXTERNRT int rtxSocketClose (OSRTSOCKET socket)
  This function closes an existing socket.

- EXTERNRT int rtxSocketConnect (OSRTSOCKET socket, const char *host, int port)
  This function establishes a connection to a specified socket.

- EXTERNRT int rtxSocketCreate (OSRTSOCKET *psocket)
  This function creates a TCP socket.

- EXTERNRT int rtxSocketGetHost (const char *host, struct in_addr *inaddr)
  This function resolves the given host name to an IP address.

- EXTERNRT int rtxSocketsInit ()
  This function initiates use of sockets by an application.

- EXTERNRT int rtxSocketListen (OSRTSOCKET socket, int maxConnection)
  This function places a socket a state where it is listening for an incoming connection.

- EXTERNRT int rtxSocketParseURL (char *url, char **protocol, char **address, int *port)
  This function parses a simple URL of the form <protocol>://<address>:<port> into its individual components.

- EXTERNRT int rtxSocketRecv (OSRTSOCKET socket, OSOCTET *pbuf, int bufsize)
  This function receives data from a connected socket.
• EXTERNRT int rtxSocketSend (OSRTSOCKET socket, const OSOCTET *pdata, int size)
  This function sends data on a connected socket.

• EXTERNRT int rtxSocketStrToAddr (const char *pIPAddrStr, OSIPADDR *pIPAddr)
  This function converts the string with IP address to a double word representation.

7.38.1 Detailed Description

Definition in file rtxSocket.h.
7.39 rtxStream.h File Reference

Input/output data stream type definitions and function prototypes.

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

Classes

- struct OSRTSTREAM
  
  *The stream control block.*

Typedefs

- typedef long(∗ OSRTStreamReadProc )(struct OSRTSTREAM ∗pStream, OSOCTET ∗pbuffer, size_t bufSize)
  
  *Stream read function pointer type.*

- typedef long(∗ OSRTStreamBlockingReadProc )(struct OSRTSTREAM ∗pStream, OSOCTET ∗pbuffer, size_t toReadBytes)
  
  *Stream blockingRead function pointer type.*

- typedef long(∗ OSRTStreamWriteProc )(struct OSRTSTREAM ∗pStream, const OSOCTET ∗data, size_t numocts)
  
  *Stream write function pointer type.*

- typedef int(∗ OSRTStreamFlushProc )(struct OSRTSTREAM ∗pStream)
  
  *Stream flush function pointer type.*

- typedef int(∗ OSRTStreamCloseProc )(struct OSRTSTREAM ∗pStream)
  
  *Stream close function pointer type.*

- typedef int(∗ OSRTStreamSkipProc )(struct OSRTSTREAM ∗pStream, size_t skipBytes)
  
  *Stream skip function pointer type.*

- typedef int(∗ OSRTStreamMarkProc )(struct OSRTSTREAM ∗pStream, size_t readAheadLimit)
  
  *Stream mark function pointer type.*

- typedef int(∗ OSRTStreamResetProc )(struct OSRTSTREAM ∗pStream)
  
  *Stream reset function pointer type.*

Functions

- EXTERNRT int rtxStreamClose (OSCTX ∗pctxt)
  
  *This function closes the input or output stream and releases any system resources associated with the stream.*

- EXTERNRT int rtxStreamFlush (OSCTX ∗pctxt)
This function flushes the output stream and forces any buffered output octets to be written out.

- EXTERNRT int rtxStreamInit (OSCTXT *pctxt)
  This function initializes a stream part of the context block.

- EXTERNRT long rtxStreamRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t bufSize)
  This function reads up to 'bufsize' bytes of data from the input stream into an array of octets.

- EXTERNRT long rtxStreamBlockingRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t readBytes)
  This function reads up to 'bufsize' bytes of data from the input stream into an array of octets.

- EXTERNRT int rtxStreamSkip (OSCTXT *pctxt, size_t skipBytes)
  This function skips over and discards the specified amount of data octets from this input stream.

- EXTERNRT long rtxStreamWrite (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
  This function writes the specified amount of octets from the specified array to the output stream.

- EXTERNRT int rtxStreamGetIOBytes (OSCTXT *pctxt, size_t *pPos)
  This function returns the number of processed octets.

- EXTERNRT int rtxStreamMark (OSCTXT *pctxt, size_t readAheadLimit)
  Marks the current position in this input stream.

- EXTERNRT int rtxStreamReset (OSCTXT *pctxt)
  Repositions this stream to the position recorded by the last call to the rtxStreamMark function.

- EXTERNRT OSBOOL rtxStreamMarkSupported (OSCTXT *pctxt)
  Tests if this input stream supports the mark and reset methods.

- EXTERNRT OSBOOL rtxStreamIsOpened (OSCTXT *pctxt)
  Tests if this stream opened (for reading or writing).

- EXTERNRT OSBOOL rtxStreamIsReadable (OSCTXT *pctxt)
  Tests if this stream opened for reading.

- EXTERNRT OSBOOL rtxStreamIsWritable (OSCTXT *pctxt)
  Tests if this stream opened for writing.

- EXTERNRT int rtxStreamRelease (OSCTXT *pctxt)
  This function releases the stream’s resources.

- EXTERNRT void rtxStreamSetCapture (OSCTXT *pctxt, OSRTMEMBUF *pmembuf)
  This function sets a capture buffer for the stream.

- EXTERNRT OSRTMEMBUF * rtxStreamGetCapture (OSCTXT *pctxt)
  This function returns the capture buffer currently assigned to the stream.
7.39.1 Detailed Description

Input/output data stream type definitions and function prototypes.
Definition in file rtxStream.h.
7.40  rtxStreamBuffered.h File Reference

#include "rtxsrtc/rtxStream.h"

7.40.1  Detailed Description

Definition in file rtxStreamBuffered.h.
7.41 rtxStreamFile.h File Reference

#include <stdio.h>
#include "rtxsrc/rtxStream.h"

Functions

- EXTERNRT int rtxStreamFileAttach (OSCTX T *pctxt, FILE *pFile, OSUINT16 flags)
  
  Attaches the existing file structure pointer to the stream.

- EXTERNRT int rtxStreamFileOpen (OSCTX T *pctxt, const char *pFilename, OSUINT16 flags)

  Opens a file stream.

- EXTERNRT int rtxStreamFileCreateReader (OSCTX T *pctxt, const char *pFilename)

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

- EXTERNRT int rtxStreamFileCreateWriter (OSCTX T *pctxt, const char *pFilename)

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

7.41.1 Detailed Description

Definition in file rtxStreamFile.h.
7.42 rtxStreamMemory.h File Reference

#include "rtxsrcretxStream.h"

Functions

- EXTERNRT int rtxStreamMemoryCreate (OSCTXT *pctxt, OSUINT16 flags)
  
  Opens a memory stream.

- EXTERNRT int rtxStreamMemoryAttach (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize, OSUINT16 flags)
  
  Opens a memory stream using the specified memory buffer.

- EXERNRT OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT *pctxt, size_t *pSize)
  
  This function returns the memory buffer and its size for the given memory stream.

- EXTERNRT int rtxStreamMemoryCreateReader (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
  
  This function creates an input memory stream using the specified buffer.

- EXTERNRT int rtxStreamMemoryCreateWriter (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
  
  This function creates an output memory stream using the specified buffer.

7.42.1 Detailed Description

Definition in file rtxStreamMemory.h.
7.43  rtxStreamSocket.h File Reference

#include "rtxsrt/rtxStream.h"
#include "rtxsrt/rtxSocket.h"

Functions

• EXTERNRT int rtxStreamSocketAttach (OSCTXT *pctxt, OSRTSOCKET socket, OSUINT16 flags)
  Attaches the existing socket handle to the stream.

• EXTERNRT int rtxStreamSocketClose (OSCTXT *pctxt)
  This function closes a socket stream.

• EXTERNRT int rtxStreamSocketCreateWriter (OSCTXT *pctxt, const char *host, int port)
  This function opens a socket stream for writing.

• EXTERNRT int rtxStreamSocketSetOwnership (OSCTXT *pctxt, OSBOOL ownSocket)
  This function transfers ownership of the socket to or from the stream instance.

7.43.1  Detailed Description

Definition in file rtxStreamSocket.h.
7.44 rtxUnicode.h File Reference

This is an open source header file derived from the libxml2 project.

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

Functions

- **EXTERNRT long rtxUCSToUTF8 (OSCTXT *pctxt, const OSUNICHAR *inbuf, size_t inlen, OSOCTET *outbuf, size_t outbufsz)**
  
  *This function converts a Unicode string into a UTF-8 string.*

- **EXTERNRT const OSUTF8CHAR * rtxUCSToDynUTF8 (OSCTXT *pctxt, const OSUNICHAR *inbuf)**
  
  *This function converts a null-terminated Unicode string into a UTF-8 string.*

- **EXTERNRT OSBOOL rtxUCSIsChar (OS32BITCHAR c)**
  
  *rtxUCSIsChar:

- **EXTERNRT OSBOOL rtxUCSIsBlank (OS32BITCHAR c)**
  
  *rtxUCSIsBlank:

- **EXTERNRT OSBOOL rtxUCSIsBaseChar (OS32BITCHAR c)**
  
  *rtxUCSIsBaseChar:

- **EXTERNRT OSBOOL rtxUCSIsDigit (OS32BITCHAR c)**
  
  *rtxUCSIsDigit:

- **EXTERNRT OSBOOL rtxUCSIsCombining (OS32BITCHAR c)**
  
  *rtxUCSIsCombining:

- **EXTERNRT OSBOOL rtxUCSIsExtender (OS32BITCHAR c)**
  
  *rtxUCSIsExtender:

- **EXTERNRT OSBOOL rtxUCSIsIdeographic (OS32BITCHAR c)**
  
  *rtxUCSIsIdeographic:

- **EXTERNRT OSBOOL rtxUCSIsLetter (OS32BITCHAR c)**
  
  *rtxUCSIsLetter:

- **EXTERNRT OSBOOL rtxUCSIsPubidChar (OS32BITCHAR c)**
  
  *rtxUCSIsPubidChar:

7.44.1 Detailed Description

This is an open source header file derived from the libxml2 project.

It defines UNICODE data types and macros. See the header file for further details.

Definition in file rtxUnicode.h.
7.44.2 Function Documentation

7.44.2.1 EXTERNRT OSBOOL rtxUCSIsBaseChar (OS32BITCHAR c)

rtxUCSIsBaseChar:

Parameters:

\( c \): an unicode character (int)

Check whether the character is allowed by the production [85] BaseChar ::= ... long list see REC ...
Returns 0 if not, non-zero otherwise

7.44.2.2 EXTERNRT OSBOOL rtxUCSIsBlank (OS32BITCHAR c)

rtxUCSIsBlank:

Parameters:

\( c \): a UNICODE character (int)

Check whether the character is allowed by the production [3] S ::= (#x20 | #x9 | #xD | #xA)+ Also available as a macro IS_BLANK()
Returns 0 if not, non-zero otherwise

7.44.2.3 EXTERNRT OSBOOL rtxUCSIsChar (OS32BITCHAR c)

rtxUCSIsChar:

Parameters:

\( c \): an unicode character (int)

Check whether the character is allowed by the production [2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#x10FFFF] any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. Also available as a macro IS_CHAR()
Returns 0 if not, non-zero otherwise

7.44.2.4 EXTERNRT OSBOOL rtxUCSIsCombining (OS32BITCHAR c)

rtxUCSIsCombining:

Parameters:

\( c \): an unicode character (int)

Check whether the character is allowed by the production [87] CombiningChar ::= ... long list see REC ...
Returns 0 if not, non-zero otherwise
7.44.2.5  EXTERNRT OSBOOL rtxUCSIsDigit (OS32BITCHAR c)

rtxUCSIsDigit:

Parameters:
  c: an unicode character (int)

Check whether the character is allowed by the production [88] Digit ::= ... long list see REC ...
Returns 0 if not, non-zero otherwise

7.44.2.6  EXTERNRT OSBOOL rtxUCSIsExtender (OS32BITCHAR c)

rtxUCSIsExtender:

Parameters:
  c: an unicode character (int)

Check whether the character is allowed by the production [89] Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 |
#x0640 | #x0E46 | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]
Returns 0 if not, non-zero otherwise

7.44.2.7  EXTERNRT OSBOOL rtxUCSIsIdeographic (OS32BITCHAR c)

rtxUCSIsIdeographic:

Parameters:
  c: an unicode character (int)

Check whether the character is allowed by the production [86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-|
#x3029]
Returns 0 if not, non-zero otherwise

7.44.2.8  EXTERNRT OSBOOL rtxUCSIsLetter (OS32BITCHAR c)

rtxUCSIsLetter:

Parameters:
  c: an unicode character (int)

Check whether the character is allowed by the production [84] Letter ::= BaseChar | Ideographic
Returns 0 if not, non-zero otherwise

7.44.2.9  EXTERNRT OSBOOL rtxUCSIsPubidChar (OS32BITCHAR c)

rtxUCSIsPubidChar:
Parameters:

\( c \): an unicode character (int)

Check whether the character is allowed by the production [13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [\-`()+,./?:=?;!∗#@$_%]

Returns 0 if not, non-zero otherwise

**7.44.2.10** EXTERNRT const OSUTF8CHAR* rtxUCSToDynUTF8 (OSCTXT * pctxt, const OSUNICHAR * inbuf)

This function converts a null-terminated Unicode string into a UTF-8 string.
Memory is allocated for the output string using the built-in memory management functions.

Parameters:

* pctxt  Pointer to context structure.
* inbuf  Null-terminated Unicode string to convert.

Returns:

Converted UTF-8 character string.

**7.44.2.11** EXTERNRT long rtxUCSToUTF8 (OSCTXT * pctxt, const OSUNICHAR * inbuf, size_t inlen, OSOCTET * outbuf, size_t outbufsz)

This function converts a Unicode string into a UTF-8 string. A buffer large enough to hold the converted UTF-8 characters must be provided. A buffer providing 4 bytes-per-character should be large enough to hold the largest possible UTF-8 conversion. The output UTF-8 string is null-terminated.

Parameters:

* pctxt  Pointer to context structure.
* inbuf  Unicode string to convert. Does not need to be null-terminated.
* inlen  Number of characters in inbuf.
* outbuf  Buffer to hold converted string.
* outbufsz  Size of output buffer.

Returns:

Total number of bytes in converted string or a negative status code if error.
7.45  rtxUTF8.h File Reference

Utility functions for handling UTF-8 strings.
#include "rtxs src/rtxContext.h"

Defines

- #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR ∗)lstr)
  Compare UTF-8 string to a string literal.

Functions

- EXTERN RT 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).

- EXTERN RT int rtxValidateUTF8 (OSCTXT ∗pctxt, const OSUTF8CHAR ∗inbuf)
  This function will validate a UTF-8 encoded string to ensure that it is encoded correctly.

- EXTERN RT size_t rtxUTF8Len (const OSUTF8CHAR ∗inbuf)
  This function will return the length (in characters) of a null-terminated UTF-8 encoded string.

- EXTERN RT size_t rtxUTF8LenBytes (const OSUTF8CHAR ∗inbuf)
  This function will return the length (in bytes) of a null-terminated UTF-8 encoded string.

- EXTERN RT 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.

- EXTERN RT int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET ∗buf, size_t bufsiz)
  This function will convert a wide character into an encoded UTF-8 character byte string.

- EXTERN RT int rtxUTF8DecodeChar (OSCTXT ∗pctxt, const OSUTF8CHAR ∗pinbuf, int ∗pInsize)
  This function will convert an encoded UTF-8 character byte string into a wide character value.

- EXTERN RT OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR ∗buf, OSUINT32 ∗len)
  This function will convert a UTF-8 encoded character value into a wide character.

- EXTERN RT OSUTF8CHAR ∗ rtxUTF8StrChr (OSUTF8CHAR ∗utf8str, OS32BITCHAR utf8char)
  This function finds a character in the given UTF-8 character string.

- EXTERN RT OSUTF8CHAR ∗ rtxUTF8Strdup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str)
  This function creates a duplicate copy of the given UTF-8 character string.

- EXTERN RT OSUTF8CHAR ∗ rtxUTF8Strndup (OSCTXT ∗pctxt, const OSUTF8CHAR ∗utf8str, size_t nbytes)
  This function creates a duplicate copy of the given UTF-8 character string.
• EXTERNRT OSUTF8CHAR * rtxUTF8StrRefOrDup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str)
  * This function checks to see if the given UTF8 string pointer exists on the memory heap.

• EXTERNRT OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
  * This function compares two UTF-8 string values for equality.

• EXTERNRT OSBOOL rtxUTF8StrmEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
  * This function compares two UTF-8 string values for equality.

• EXTERNRT int rtxUTF8Strcmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
  * This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than).

• EXTERNRT int rtxUTF8Strncmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
  * This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than).

• EXTERNRT OSUTF8CHAR * rtxUTF8Strcpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src)
  * This function copies a null-terminated UTF-8 string to a target buffer.

• EXTERNRT OSUTF8CHAR * rtxUTF8Strncpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src, size_t nchars)
  * This function copies the given number of characters from a UTF-8 string to a target buffer.

• EXTERNRT OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR *str)
  * This function computes a hash code for the given string value.

• EXTERNRT const OSUTF8CHAR * rtxUTF8StrJoin (OSCTXT *pctxt, const OSUTF8CHAR *str1, const OSUTF8CHAR *str2, const OSUTF8CHAR *str3, const OSUTF8CHAR *str4, const OSUTF8CHAR *str5)
  * This function concatenates up to five substrings together into a single string.

• EXTERNRT int rtxUTF8StrToBool (const OSUTF8CHAR *utf8str, OSBOOL *pvalue)
  * This function converts the given null-terminated UTF-8 string to a boolean (true/false) value.

• EXTERNRT int rtxUTF8StrmToBool (const OSUTF8CHAR *utf8str, size_t nbytes, OSBOOL *pvalue)
  * This function converts the given part of UTF-8 string to a boolean (true/false) value.

• EXTERNRT int rtxUTF8StrToDouble (const OSUTF8CHAR *utf8str, OSREAL *pvalue)
  * This function converts the given null-terminated UTF-8 string to a floating point (C/C++ double) value.

• EXTERNRT int rtxUTF8StrmToDouble (const OSUTF8CHAR *utf8str, size_t nbytes, OSREAL *pvalue)
  * This function converts the given part of UTF-8 string to a double value.

• EXTERNRT int rtxUTF8StrToInt (const OSUTF8CHAR *utf8str, OSINT32 *pvalue)
  * This function converts the given null-terminated UTF-8 string to an integer value.

• EXTERNRT int rtxUTF8StrmToInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT32 *pvalue)
  * This function converts the given part of UTF-8 string to an integer value.

• EXTERNRT int rtxUTF8StrtUInt (const OSUTF8CHAR *utf8str, OSUINT32 *pvalue)
This function converts the given null-terminated UTF-8 string to an unsigned integer value.

- EXTERNRT int rtxUTF8StrToUInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT32 *pvalue)
  
  This function converts the given part of UTF-8 string to an unsigned integer value.

- EXTERNRT int rtxUTF8StrToInt64 (const OSUTF8CHAR *utf8str, OSINT64 *pvalue)
  
  This function converts the given null-terminated UTF-8 string to a 64-bit integer value.

- EXTERNRT int rtxUTF8StrnToUInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT32 *pvalue)
  
  This function converts the given part of UTF-8 string to an unsigned integer value.

- EXTERNRT int rtxUTF8StrnToInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT64 *pvalue)
  
  This function converts the given part of UTF-8 string to a 64-bit integer value.

- EXTERNRT int rtxUTF8StrToUInt64 (const OSUTF8CHAR *utf8str, OSUINT64 *pvalue)
  
  This function converts the given null-terminated UTF-8 string to an unsigned 64-bit integer value.

- EXTERNRT int rtxUTF8StrnToUInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT64 *pvalue)
  
  This function converts the given part of UTF-8 string to an unsigned 64-bit integer value.

- EXTERNRT int rtxUTF8ToDynUniStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSUNICHAR **ppdata, OSUINT32 *pnchars)
  
  This function converts the given UTF-8 string to a Unicode string.

- EXTERNRT int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR *utf8instr, size_t nbytes, const OSUTF8CHAR **utf8outstr)
  
  This function removes leading and trailing whitespace from a string.

- EXTERNRT int rtxUTF8StrToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, OSDynOctStr *pvalue)
  
  This function converts the given null-terminated UTF-8 string to a octet string value.

- EXTERNRT int rtxUTF8StrnToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, OSDynOctStr *pvalue)
  
  This function converts the given part of UTF-8 string to a octet string value.

- EXTERNRT int rtxUTF8StrToNamedBits (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSBitMapItem *pBitmap, OSOCTET *pvalue, OSUINT32 *pnbits, OSUINT32 bufsize)
  
  This function converts the given null-terminated UTF-8 string to named bit items.

- EXTERNRT const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR *utf8str, OSUTF8CHAR **ppNext)
  
  This function returns the next whitespace-separated token from the input string.

7.45.1 Detailed Description

Utility functions for handling UTF-8 strings.

Definition in file rtxUTF8.h.
7.46 rtxXmlQName.h File Reference

XML QName type definition and associated utility functions.

#include "rtxsrc/rtxContext.h"

Classes

- struct OSXMLFullQName
  
  *This version of QName contains complete namespace info (prefix + URI).*

Functions

- EXTERNRT OSXMLFullQName * rtxNewFullQName (OSCTXT *pctxt, const OSUTF8CHAR *localName, const OSUTF8CHAR *prefix, const OSUTF8CHAR *nsuri)
  
  *This function creates a new full QName structure given the parts.*

- EXTERNRT OSXMLFullQName * rtxNewFullQNameDeepCopy (OSCTXT *pctxt, const OSXMLFullQName *pqname)
  
  *This function allocates a new QName instance and makes a deep copy of the given QName including the strings inside.*

- EXTERNRT void rtxQNameDeepCopy (OSCTXT *pctxt, OSXMLFullQName *pdest, const OSXMLFullQName *psrc)
  
  *This function makes a deep copy of the given QName including the strings inside.*

- EXTERNRT void rtxQNameFreeMem (OSCTXT *pctxt, OSXMLFullQName *pqname, OSBOOL dynamic)
  
  *This function frees all memory within a QName structure.*

- EXTERNRT OSUINT32 rtxQNameHash (const OSXMLFullQName *pqname)
  
  *This function computes a hash code for the given QName.*

- EXTERNRT OSBOOL rtxQNamesEqual (const OSXMLFullQName *pqname1, const OSXMLFullQName *pqname2)
  
  *This function tests 2 QNames for equality.*

- EXTERNRT const OSUTF8CHAR * rtxQNameToString (const OSXMLFullQName *pqname, OSUTF8CHAR *buffer, OSUINT32 bufsiz)
  
  *This function returns the QName in the following stringified format: [uri]/localName.*

7.46.1 Detailed Description

XML QName type definition and associated utility functions.

Definition in file rtxXmlQName.h.
7.46.2 Function Documentation

7.46.2.1 EXTERNRT OSXMLFullQName* rtxNewFullQName (OSCTX * pctxt, const OSUTF8CHAR * localName, const OSUTF8CHAR * prefix, const OSUTF8CHAR * nsuri)

This function creates a new full QName structure given the parts.
Memory is allocated for the structure using rtxMemAlloc. Copies are not made of the string variables - the pointers are stored.

Parameters:

   pctxt   Pointer to a context structure.
   localName Element local name.
   prefix   Namespace prefix.
   nsuri    Namespace URI.

Returns:

   QName value. Memory for the value will have been allocated by rtxMemAlloc and thus must be freed using one of the rtxMemFree functions. The value will be NULL if no dynamic memory was available.

7.46.2.2 EXTERNRT OSXMLFullQName* rtxNewFullQNameDeepCopy (OSCTX * pctxt, const OSXMLFullQName * pqname)

This function allocates a new QName instance and makes a deep copy of the given QName including the strings inside.

Parameters:

   pctxt   Pointer to a context structure.
   pqname  Pointer to QName to be copied.

Returns:

   Deep copy of QName structure.

7.46.2.3 EXTERNRT void rtxQNameDeepCopy (OSCTX * pctxt, OSXMLFullQName * pdest, const OSXMLFullQName * psrc)

This function makes a deep copy of the given QName including the strings inside.

Parameters:

   pctxt   Pointer to a context structure.
   pdest   Pointer to QName to receive copied data.
   psrc    Pointer to QName to be copied.
7.46.2.4 EXTERNRT void rtxQNameFreeMem (OSCTXT * pctxt, OSXMLFullQName * pqname, OSBOOL dynamic)

This function frees all memory within a QName structure.

Parameters:

  pctxt  Pointer to a context structure.
  pqname Pointer to QName in which memory will be freed.
  dynamic Boolean indicating if pqname is dynamic. If true, the memory for pqname is freed.

7.46.2.5 EXTERNRT OSUINT32 rtxQNameHash (const OSXMLFullQName * pqname)

This function computes a hash code for the given QName.

Parameters:

  pqname  Pointer to QName structure.

Returns:

  Computed hash code.

7.46.2.6 EXTERNRT OSBOOL rtxQNamesEqual (const OSXMLFullQName * pqname1, const OSXMLFullQName * pqname2)

This function tests 2 QNames for equality.

Parameters:

  pqname1  Pointer to QName structure.
  pqname2  Pointer to QName structure.

Returns:

  True if names equal; false, otherwise.

7.46.2.7 EXTERNRT const OSUTF8CHAR* rtxQNameToString (const OSXMLFullQName * pqname, OSUTF8CHAR * buffer, OSUINT32 bufsiz)

This function returns the QName in the following stringified format: {uri}/localName.

Parameters:

  pqname  Pointer to QName structure.
  buffer  Buffer into which to return name.
  bufsiz  Size of buffer into which name is to be returned. If name will not fit in buffer, it is truncated.

Returns:

  Pointer to string (address of 'buffer' argument).
7.47 rtxXmlStr.h File Reference

#include "rtxsrc/rtxContext.h"

Functions

- EXTERNRT OSXMLSTRING * rtxCreateXmlStr (OSCTXT *pctxt, const OSUTF8CHAR *pStr, OSBOOL cdata)

  This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

- EXTERNRT OSXMLSTRING * rtxCreateCopyXmlStr (OSCTXT *pctxt, const OSUTF8CHAR *pStr, OSBOOL cdata)

  This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

7.47.1 Detailed Description

Definition in file rtxXmlStr.h.

7.47.2 Function Documentation

7.47.2.1 EXTERNRT OSXMLSTRING* rtxCreateCopyXmlStr (OSCTXT *pctxt, const OSUTF8CHAR *pStr, OSBOOL cdata)

This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

In contrary to rtxCreateXmlStr function, the string value is copied. This function uses rtxMemAlloc to allocate the memory for the OSXMLSTRING structure and for the string value being copied. To free memory, rtxMemFreePtr function may be used for both value and structure itself:

```
OSXMLSTRING* pStr = rtxCreateCopyXmlStr (....);
```

```
....
rtxMemFreePtr (pctxt, pStr->value);
rtxMemFreePtr (pctxt, pStr);
```

Parameters:

- **pctxt** Pointer to a context block
- **pStr** Pointer to a character string to be copied.
- **cdata** This indicates if this string should be encoded as a CDATA section in an XML document.

Returns:

The allocated and initialized instance of OSXMLSTRING type.
7.47.2.2  EXTERNRT OSXMLSTRING* rtxCreateXmlStr (OSCTXT * pctxt, const OSUTF8CHAR * pStr, OSBOOL cdata)

This function creates an instance of XML UTF-8 character string structure (OSXMLSTRING type) and initializes it by the passed values.

This function uses rtxMemAlloc to allocate the memory for the OSXMLSTRING structure. String pStr is not copied: the pointer will be assigned to "value" member of OSXMLSTRING structure. To free memory, rtxMemFreePtr function may be used:

OSXMLSTRING* pStr = rtxCreateXmlStr (....);

....

rtxMemFreePtr (pctxt, pStr);

Note, user is responsible to free pStr->value if it is not static and was allocated dynamically.

Parameters:

  pctxt  Pointer to a context block
  pStr   Pointer to a character string to be assigned.
  cdata  This indicates if this string should be encoded as a CDATA section in an XML document.

Returns:

  The allocated and initialized instance of OSXMLSTRING type.
Index

Bit String Functions, 6
bitstrhelpers
   rtxClearBit, 6
   rtxSetBit, 7
   rtxSetBitFlags, 7
   rtxTestBit, 7
buffermanfun
   rtxMemBufAppend, 78
   rtxMemBufCut, 78
   rtxMemBufFree, 78
   rtxMemBufGetData, 79
   rtxMemBufGetDataLen, 79
   rtxMemBufInit, 79
   rtxMemBufInitBuffer, 79
   rtxMemBufPreAllocate, 80
   rtxMemBufReset, 80
   rtxMemBufSet, 80
   rtxMemBufSetExpandable, 81
   rtxMemBufTrimW, 81
ccfDateTime
   rtxCmpDate, 23
   rtxCmpDate2, 23
   rtxCmpDateTime, 24
   rtxCmpDateTime2, 24
   rtxCmpTime, 25
   rtxCmpTime2, 25
   rtxDateIsValid, 25
   rtxDateTimeIsValid, 26
   rtxDateTimeToString, 26
   rtxDateToString, 26
   rtxDurationToMSecs, 27
   rtxGDayToString, 27
   rtxGetCurrDateTime, 27
   rtxGetDateTime, 28
ccfDiag
   rtxDiagEnabled, 38
   rtxDiagHexDump, 38
   rtxDiagPrint, 39
   rtxDiagPrintChars, 39
   rtxDiagSetTraceLevel, 39
   rtxDiagStream, 39
   rtxDiagStreamHexDump, 40
   rtxDiagStreamPrintChars, 40
   rtxDiagToStream, 40
   rtxPrintStreamRelease, 40
   rtxPrintToStream, 41
   rtxSetDiag, 41
   rtxSetGlobalDiag, 41
   rtxSetGlobalPrintStream, 42
   rtxSetPrintStream, 42
ccfDList
   rtxDListAppend, 44
   rtxDListAppendArray, 44
   rtxDListAppendArrayCopy, 45
   rtxDListFindByData, 45
   rtxDListFindByIndex, 45
   rtxDListFindIndexByData, 46
   rtxDListFreeAll, 46
   rtxDListFreeNode, 46
   rtxDListFreeNodes, 46
   rtxDListInit, 47
   rtxDListInsert, 47
   rtxDListInsertAfter, 47
   rtxDListInsertBefore, 47
   rtxDListRemove, 48
   rtxDListToArray, 48
   rtxDListToUTF8Str, 49
ccfErr
   LOG_RTERR, 68
   OSRTASSERT, 68
   OSRTCHECKPARAM, 68
rtxHashMap.h, 212
HASHMAPINSERTFUNC
rtxHashMap.h, 212
HASHMAPNEWFUNC
rtxHashMap.h, 213
HASHMAPPUTFUNC
rtxHashMap.h, 213
HASHMAPREMOVEFUNC
rtxHashMap.h, 213
HASHMAPSEARCHFUNC
rtxHashMap.h, 214
HASHMAPSORTFUNC
rtxHashMap.h, 214
head
OSRTDLList, 152

Input/Output Data Stream Utility Functions, 115

LOG_RTERR
ccfErr, 68

Memory Allocation Macros and Functions, 82
Memory Buffer Management Functions, 77
Memory stream functions., 126

next
OSRTDLListNode, 153

OSBitMapItem, 146
OSCTX, 147
OSDynOctStr, 148
OSIPADDR
ccfSocket, 110
OSNumDateTime, 149
OSRTALLOCTYPE
rtxmem, 83
OSRTALLOCTYPEZ
rtxmem, 83
OSRTASSERT
ccfErr, 68
OSRTBuffer, 150
OSRTBufSave, 151
OSRTCHECKPARAM
ccfErr, 68
OSRTDLList, 152
count, 152
head, 152
tail, 152
OSRTDLListNode, 153
data, 153
next, 153
prev, 153
OSRTERrInfo, 154
OSRTERrLocn, 155
OSRTPrintStream, 156

OSRTScalarDLList, 157
OSRTScalarDLListNode, 158
OSRTSTREAM, 159
OSRTStreamBlockingReadProc
rtxStream, 117
OSRTStreamCloseProc
rtxStream, 117
OSRTStreamFlushProc
rtxStream, 117
OSRTStreamMarkProc
rtxStream, 117
OSRTStreamReadProc
rtxStream, 117
OSRTStreamResetProc
rtxStream, 118
OSRTStreamSkipProc
rtxStream, 118
OSRTStreamWriteProc
rtxStream, 118
OSXMLFullQName, 160
OSXMLSTRING, 161
OSXSDAny, 162
OSXSDDateTime, 163

Pattern matching functions, 87

prev
OSRTDLListNode, 153
Print Functions, 88
Print-To-Stream Functions, 96
prtToStrm

rtxHexDumpToStream, 98
rtxHexDumpToStreamEx, 98
rtxPrintToStreamBoolean, 98
rtxPrintToStreamCharStr, 98
rtxPrintToStreamDate, 98
rtxPrintToStreamDateTime, 99
rtxPrintToStreamFile, 99
rtxPrintToStreamHexBinary, 99
rtxPrintToStreamHexStr, 99
rtxPrintToStreamInt64, 100
rtxPrintToStreamInteger, 100
rtxPrintToStreamNull, 100
rtxPrintToStreamNVP, 100
rtxPrintToStreamReal, 101
rtxPrintToStreamTime, 101
rtxPrintToStreamUInt64, 101
rtxPrintToStreamUnicodeCharStr, 101
rtxPrintToStreamUnsigned, 102
rtxPrintToStreamUTF8CharStr, 102

RT_MH_DONTKEEPFREE
rtxMemory.h, 220
RT_OK_FRAG
rtxErrCodes, 56
rtxDateTimeIsValid
 ccfdDateTime, 25
 rtxDateTime
 ccfdDateTime, 25
 rtxDateTimeIsValid
 ccfdDateTime, 25
 rtxDateTimeToString
 ccfdDateTime, 25
 rtxDecBits
 rtxBitDecode.h, 172
 rtxDecimal.h, 184
 rtxDiag.h, 185
 rtxDiagBitFieldListInit
 rtxDiagBitTrace.h, 187
 rtxDiagBitFldAppendNamePart
 rtxDiagBitTrace.h, 187
 rtxDiagBitTrace.h, 186
 rtxDiagBitFieldListInit, 187
 rtxDiagBitFldAppendNamePart, 187
 rtxDiagBitTraceEnabled, 187
 rtxDiagBitTracePrint, 187
 rtxDiagBitTracePrintHTML, 188
 rtxDiagInsBitFieldLen, 188
 rtxDiagNewBitField, 188
 rtxDiagSetBitFldCount, 188
 rtxDiagSetBitFldDisabled, 188
 rtxDiagSetBitFldNameSuffix, 189
 rtxDiagSetBitFldOffset, 189
 rtxDiagSetBitTraceEnabled, 189
 rtxDiagBitTraceEnabled
 rtxDiagBitTrace.h, 187
 rtxDiagBitTracePrint
 rtxDiagBitTrace.h, 187
 rtxDiagBitTracePrintHTML
 rtxDiagBitTrace.h, 188
 rtxDiagEnabled
 ccfdDiag, 38
 rtxDiagHexDump
 ccfdDiag, 38
 rtxDiagInsBitFieldLen
 rtxDiagBitTrace.h, 188
 rtxDiagNewBitField
 rtxDiagBitTrace.h, 188
 rtxDiagPrint
 ccfdDiag, 39
 rtxDiagPrintChars
 ccfdDiag, 39
 rtxDiagSetBitFldCount
 ccfdDiag, 39
 rtxDiagSetBitField
 rtxDiagBitTrace.h, 188
 rtxDiagSetBitFldDisabled
 rtxDiagBitTrace.h, 188
 rtxDiagSetBitFldNameSuffix
 rtxDiagBitTrace.h, 189
 rtxDiagSetBitFldOffset
 rtxDiagBitTrace.h, 189
 rtxDiagSetTraceLevel
 ccfdDiag, 39
 rtxDiagStream
 ccfdDiag, 39
 rtxDiagStreamHexDump
RTERR_ATTRFIXEDVAL, 56
RTERR_ATTRMISQ, 56
RTERR_BADVALUE, 56
RTERR_BUFOVFLOW, 56
RTERR_CONNREFUSED, 56
RTERR_CONNRESET, 57
RTERR_CONNVIO, 57
RTERR_DECATTRFAIL, 57
RTERR_DECLEMEFAIL, 57
RTERR_ENDOFBUF, 57
RTERR_ENDOFFILE, 57
RTERR_EXPIRED, 58
RTERR_FAILED, 58
RTERR_FILENOTFOUND, 58
RTERR_HOSTNOTFOUND, 58
RTERR_HTTPERR, 58
RTERR_INVATTR, 58
RTERR_INVBASE64, 59
RTERR_INVCHAR, 59
RTERR_INVREAL, 59
RTERR_INVFORMAT, 59
RTERR_INVHEXS, 59
RTERR_INVMSGBUF, 59
RTERR_INVOCUR, 60
RTERR_INVOPT, 60
RTERR_INVPARAM, 60
RTERR_INVREAL, 60
RTERR_INVSOCKET, 60
RTERR_INVSOCKOPT, 60
RTERR_INVUTF8, 61
RTERR_MULTIPLE, 61
RTERR_NOCONN, 61
RTERR_NOMEM, 61
RTERR_NOTINIT, 61
RTERR_NOTINSET, 61
RTERR_NOTSUPP, 62
RTERR_NOTYPEINFO, 62
RTERR_NULLPTR, 62
RTERR_OUTOFBOUND, 62
RTERR_PATMATCH, 62
RTERR_READERR, 62
RTERR_REGEXP, 63
RTERR_SEQORDER, 63
RTERR_SEQOVFLOW, 63
RTERR_SETDUPL, 63
RTERR_SETMISQ, 63
RTERR_SOAPERR, 63
RTERR_STRMINUSE, 63
RTERR_STROVFLOW, 64
RTERR_TOOBIG, 64
RTERR_TOODEEP, 64
RTERR_UNEXPELEM, 64
RTERR_UNREACHABLE, 64
RTERR_WRITEERR, 64
RTERR_XMLPARSE, 65
RTERR_XMLSTATE, 65

rtxErrCodes.h, 199
rtxErrFmtMsg
ccfErr, 72
rtxErrFreeParms
ccfErr, 72
rtxErrGetErrorCnt
ccfErr, 72
rtxErrGetFirstError
ccfErr, 73
rtxErrGetLastError
ccfErr, 73
rtxErrGetStatus
ccfErr, 73
rtxErrGetText
ccfErr, 73
rtxErrGetTextBuf
ccfErr, 74
rtxErrInit
ccfErr, 74
rtxErrLogUsingCB
ccfErr, 74
rtxErrNewNode
ccfErr, 75
rtxError.h, 203
rtxErrPrint
ccfErr, 75
rtxErrPrintElement
ccfErr, 75
rtxErrReset
ccfErr, 75
rtxErrResetLastErrors
ccfErr, 75
rtxErrSetData
ccfErr, 76
rtxErrSetNewData
ccfErr, 76
rtxExternDefs.h, 206
rtxFile.h, 207
rtxFileReadBinary, 207
rtxFileReadText, 207
rtxFileWriteBinary, 208
rtxFileWriteText, 208
rtxFile.h, 208
rtxFile.h, 208
rtxFile.h, 207
rtxFile.h, 210
<table>
<thead>
<tr>
<th>Function</th>
<th>Header</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtxFreeArrayList</td>
<td>rtxArrayList.h</td>
<td>168</td>
</tr>
<tr>
<td>rtxFreeContext</td>
<td>rtxCtxt</td>
<td>18</td>
</tr>
<tr>
<td>rtxGDayToString</td>
<td>ccfDateTime</td>
<td>27</td>
</tr>
<tr>
<td>rtxGetCurrDateTime</td>
<td>ccfDateTime</td>
<td>27</td>
</tr>
<tr>
<td>rtxGetDateTime</td>
<td>ccfDateTime</td>
<td>28</td>
</tr>
<tr>
<td>rtxGetMinusInfinity</td>
<td>rtxReal</td>
<td>103</td>
</tr>
<tr>
<td>rtxGetMinusZero</td>
<td>rtxReal</td>
<td>103</td>
</tr>
<tr>
<td>rtxGetNaN</td>
<td>rtxReal</td>
<td>104</td>
</tr>
<tr>
<td>rtxGetPlusInfinity</td>
<td>rtxReal</td>
<td>104</td>
</tr>
<tr>
<td>rtxGMonthDayToString</td>
<td>ccfDateTime</td>
<td>28</td>
</tr>
<tr>
<td>rtxGMonthToString</td>
<td>ccfDateTime</td>
<td>28</td>
</tr>
<tr>
<td>rtxGYearMonthToString</td>
<td>ccfDateTime</td>
<td>29</td>
</tr>
<tr>
<td>rtxGYearToString</td>
<td>ccfDateTime</td>
<td>29</td>
</tr>
<tr>
<td>rtxHashMap.h</td>
<td></td>
<td>211</td>
</tr>
<tr>
<td>HASHMAPCOPYFUNC</td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>HASHMAPFREEFUNC</td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>HASHMAPINITFUNC</td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>HASHMAPINSERTFUNC</td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>HASHMAPNEWFUNC</td>
<td></td>
<td>213</td>
</tr>
<tr>
<td>HASHMAPPUTFUNC</td>
<td></td>
<td>213</td>
</tr>
<tr>
<td>HASHMAPREMOVEFUNC</td>
<td></td>
<td>213</td>
</tr>
<tr>
<td>HASHMAPSEARCHFUNC</td>
<td></td>
<td>214</td>
</tr>
<tr>
<td>HASHMAPSORTFUNC</td>
<td></td>
<td>214</td>
</tr>
<tr>
<td>rtxHashMapStr2Int.h</td>
<td></td>
<td>215</td>
</tr>
<tr>
<td>rtxHashMapStr2UInt.h</td>
<td></td>
<td>216</td>
</tr>
<tr>
<td>rtxHashMapUndef.h</td>
<td></td>
<td>217</td>
</tr>
<tr>
<td>rtxHexCharsToBin</td>
<td>rtxCharStr</td>
<td>9</td>
</tr>
<tr>
<td>rtxHexCharsToBinCount</td>
<td>rtxCharStr</td>
<td>10</td>
</tr>
<tr>
<td>rtxHexDump</td>
<td>rtxCharStr</td>
<td>90</td>
</tr>
<tr>
<td>rtxHexDumpEx</td>
<td>rtxCharStr</td>
<td>90</td>
</tr>
<tr>
<td>rtxHexDumpToStdout</td>
<td>valsToStdout</td>
<td>90</td>
</tr>
<tr>
<td>rtxHexDumpEx</td>
<td>valsToStdout</td>
<td>90</td>
</tr>
<tr>
<td>rtxHexDumpToFile</td>
<td>valsToStdout</td>
<td>90</td>
</tr>
<tr>
<td>rtxHexDumpToFileEx</td>
<td>valsToStdout</td>
<td>91</td>
</tr>
<tr>
<td>rtxHexDumpToNamedFile</td>
<td>valsToStdout</td>
<td>91</td>
</tr>
<tr>
<td>rtxHexDumpToStream</td>
<td>prtToStrm</td>
<td>98</td>
</tr>
<tr>
<td>rtxHexDumpToString</td>
<td>valsToStdout</td>
<td>91</td>
</tr>
<tr>
<td>rtxHexDumpToStringEx</td>
<td>valsToStdout</td>
<td>91</td>
</tr>
<tr>
<td>rtxInitContext</td>
<td>rtxCtxt</td>
<td>18</td>
</tr>
<tr>
<td>rtxInitContextBuffer</td>
<td>rtxCtxt</td>
<td>18</td>
</tr>
<tr>
<td>rtxInitContextExt</td>
<td>rtxCtxt</td>
<td>19</td>
</tr>
<tr>
<td>rtxInitContextUsingKey</td>
<td>rtxCtxt</td>
<td>19</td>
</tr>
<tr>
<td>rtxInitThreadContext</td>
<td>rtxCtxt</td>
<td>20</td>
</tr>
<tr>
<td>rtxInt64ToCharStr</td>
<td>rtxCharStr</td>
<td>10</td>
</tr>
<tr>
<td>rtxIntToCharStr</td>
<td>rtxCharStr</td>
<td>10</td>
</tr>
<tr>
<td>rtxIsMinusInfinity</td>
<td>rtxReal</td>
<td>104</td>
</tr>
<tr>
<td>rtxIsMinusZero</td>
<td>rtxReal</td>
<td>104</td>
</tr>
<tr>
<td>rtxIsNaN</td>
<td>rtxReal</td>
<td>104</td>
</tr>
<tr>
<td>rtxIsPlusInfinity</td>
<td>rtxReal</td>
<td>104</td>
</tr>
<tr>
<td>rtxLookupEnum</td>
<td>rtxEnum</td>
<td>50</td>
</tr>
<tr>
<td>rtxLookupEnumByValue</td>
<td>rtxEnum</td>
<td>50</td>
</tr>
<tr>
<td>rtxMatchPattern</td>
<td>ccfPattern</td>
<td>87</td>
</tr>
<tr>
<td>rtxMemAllocArray</td>
<td>rtmem</td>
<td>83</td>
</tr>
<tr>
<td>rtxMemAllocType</td>
<td>rtmem</td>
<td>83</td>
</tr>
<tr>
<td>rtxMemAllocTypeZ</td>
<td>rtmem</td>
<td>84</td>
</tr>
<tr>
<td>rtxMemAutoPtrGetRefCount</td>
<td>rtmem</td>
<td>84</td>
</tr>
<tr>
<td>rtxMemAutoPtrRef</td>
<td>rtmem</td>
<td>84</td>
</tr>
<tr>
<td>rtxMemAutoPtrUnref</td>
<td>rtmem</td>
<td>85</td>
</tr>
<tr>
<td>rtxMemBuf.h</td>
<td></td>
<td>218</td>
</tr>
<tr>
<td>rtxMemBufAppend</td>
<td>buffermanfun</td>
<td>78</td>
</tr>
<tr>
<td>rtxMemBufCut</td>
<td>buffermanfun</td>
<td>78</td>
</tr>
<tr>
<td>rtxMemBufFree</td>
<td>buffermanfun</td>
<td>78</td>
</tr>
</tbody>
</table>
rtxStreamMemory, 127
rtxStreamRead
  rtxStream, 121
rtxStreamRelease
  rtxStream, 121
rtxStreamReset
  rtxStream, 122
rtxStreamSetCapture
  rtxStream, 122
rtxStreamSkip
  rtxStream, 122
rtxStreamSocket
  rtxStreamSocketAttach, 129
  rtxStreamSocketClose, 129
  rtxStreamSocketCreateWriter, 130
  rtxStreamSocketSetOwnership, 130
rtxStreamSocket.h, 242
rtxStreamSocketAttach
  rtxStreamSocket, 129
rtxStreamSocketClose
  rtxStreamSocket, 129
rtxStreamSocketCreateWriter
  rtxStreamSocket, 130
rtxStreamSocketSetOwnership
  rtxStreamSocket, 130
rtxStreamWrite
  rtxStream, 122
rtxStrncat
  rtxCharStr, 11
rtxStrncpy
  rtxCharStr, 12
rtxTestBit
  bitstrhelpers, 7
rtxTestNumericEnum
  rtxEnum, 51
rtxTimeIsValid
  ccfDateTime, 34
rtxTimeToString
  ccfDateTime, 34
rtxUCSIsBaseChar
  rtxUnicode.h, 244
rtxUCSIsBlank
  rtxUnicode.h, 244
rtxUCSIsChar
  rtxUnicode.h, 244
rtxUCSIsCombining
  rtxUnicode.h, 244
rtxUCSIsDigit
  rtxUnicode.h, 244
rtxUCSIsExtender
  rtxUnicode.h, 245
rtxUCSIsIdeographic
  rtxUnicode.h, 245
rtxUCSISIsLetter
  rtxUnicode.h, 245
rtxUCSISIsPubidChar
  rtxUnicode.h, 245
rtxUCSISToDynUTF8
  rtxUnicode.h, 246
rtxUCSToUTF8
  rtxUnicode.h, 246
rtxUInt64ToCharStr
  rtxCharStr, 12
rtxUIntToCharStr
  rtxCharStr, 12
rtxUnicode.h, 243
rtxUCSISBaseChar, 244
rtxUCSISBlank, 244
rtxUCSISChar, 244
rtxUCSISCombining, 244
rtxUCSISDigit, 244
rtxUCSISExtender, 245
rtxUCSISIdeographic, 245
rtxUCSISLetter, 245
rtxUCSISPubidChar, 245
rtxUCSToDynUTF8, 246
rtxUCSToUTF8, 246
rtxUTF8.h, 247
rtxUTF8CharSize
  ccfUTF8, 134
rtxUTF8CharToWC
  ccfUTF8, 134
rtxUTF8DecodeChar
  ccfUTF8, 134
rtxUTF8EncodeChar
  ccfUTF8, 134
rtxUTF8Len
  ccfUTF8, 135
rtxUTF8LenBytes
  ccfUTF8, 135
rtxUTF8RemoveWhiteSpace
  ccfUTF8, 135
rtxUTF8StrChr
  ccfUTF8, 135
rtxUTF8Strcmp
  ccfUTF8, 136
rtxUTF8Strcprop
  ccfUTF8, 136
rtxUTF8Strdup
  ccfUTF8, 136
rtxUTF8StrEqual
  ccfUTF8, 137
rtxUTF8StrHash
  ccfUTF8, 137
rtxUTF8StrJoin
  ccfUTF8, 137
rtxUTF8Strncmp
  ccfUTF8, 137
267