Oracle® Database

XML C API Reference

23ai
F47014-03
May 2024

Oracle Database XML C API Reference, 23ai

F47014-03

Copyright © 1999, 2024, Oracle and/or its affiliates.

Primary Author: Maitreyee Chaliha

Contributing Authors: Apoorva Srinivas, Jayashree Sharma, Tulika Das, Roza Leyderman, Anguel Novoselsky, Ian Macky, Vijay Medi

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience xx
Documentation Accessibility xx
Related Documents xx
Conventions xxi

1 Datatypes for XML C APIs

oracheck 1-2
oraerr 1-2
oraprop_id 1-3
oramemctx 1-3
oraprop 1-3
oraprop_t 1-3
oraprop_v 1-3
orastream 1-4
orastreamhdl 1-4
xmlcmphow 1-4
xmlctx 1-4
xmldfoptype 1-5
xmldfsrct 1-5
xmlerr 1-5
xmlevctx 1-6
xmlevtype 1-6
xmlhasht 1-7
xmlistream 1-7
xmliter 1-7
xmlnodetype 1-8
xmlostream 1-8
xmlpoint 1-9
xmlrange 1-9
xmlsoapbind 1-9
xmlsoapcon 1-10

xmlsoapctx 1-10
xmlsoaprole 1-10
xmlshowbits 1-10
xmlurlacc 1-11
xmlurlhdl 1-11
xmlurlpart 1-11
xmlxptrloc 1-12
xmlxptrlocset 1-12
xmlxslobjtype 1-12
xmlxslomethod 1-12
xmlxvm 1-12
xmlxvmcomp 1-13
xmlxvmflags 1-13
xmlxvmobjtype 1-13
xpctx 1-13
xpexpr 1-14
xpobj 1-14
xsdctx 1-14
xslctx 1-14
xvmobj 1-14
JsonDomDoc 1-14
JsonDomNode 1-15
JsonDomObject 1-15
JsonDomArray 1-15
JsonDomScalar 1-15
jznnodetype 1-15
jznvaltype 1-15
jznScalarVal 1-16
JsonOCIVal 1-16
JsonDateTime 1-17
JsonDayInterval 1-17
JsonYearInterval 1-17
jsonerr 1-17

2 Package Callback for XML C APIs

XML_ACCESS_CLOSE_F() 2-1
XML_ACCESS_OPEN_F() 2-1
XML_ACCESS_READ_F() 2-1
XML_ALLOC_F() 2-1
XML_ERRMSG_F() 2-2

XML_FREE_F() 2-2
XML_STREAM_CLOSE_F() 2-2
XML_STREAM_OPEN_F() 2-2
XML_STREAM_READ_F() 2-2
XML_STREAM_WRITE_F() 2-3

3 Package DOM for XML C APIs

Attr Interface for DOM XML C APIs 3-1
- XmlDomGetAttrLocal() 3-2
- XmlDomGetAttrLocalLen() 3-2
- XmlDomGetAttrName() 3-3
- XmlDomGetAttrNameLen() 3-4
- XmlDomGetAttrPrefix() 3-5
- XmlDomGetAttrSpecified() 3-5
- XmlDomGetAttrURI() 3-6
- XmlDomGetAttrURILen() 3-7
- XmlDomGetAttrValue() 3-8
- XmlDomGetAttrValueLen() 3-8
- XmlDomGetAttrValueStream() 3-9
- XmlDomGetOwnerElem() 3-10
- XmlDomSetAttrValue() 3-10
- XmlDomSetAttrValueStream() 3-11

CharacterData Interface for DOM XML C APIs 3-11
- XmlDomAppendData() 3-12
- XmlDomDeleteData() 3-12
- XmlDomGetCharData() 3-13
- XmlDomGetCharDataLength() 3-14
- XmlDomInsertData() 3-14
- XmlDomReplaceData() 3-15
- XmlDomSetCharData() 3-16
- XmlDomSubstringData() 3-16

Document Interface for DOM XML C APIs 3-17
- XmlDomCreateAttr() 3-18
- XmlDomCreateAttrNS() 3-19
- XmlDomCreateCDATA() 3-20
- XmlDomCreateComment() 3-21
- XmlDomCreateElem() 3-22
- XmlDomCreateElemNS() 3-22
- XmlDomCreateEntityRef() 3-23
- XmlDomCreateFragment() 3-24

XmlDomCreatePI() 3-24
XmlDomCreateText() 3-25
XmlDomFreeString() 3-26
XmlDomGetBaseURI() 3-27
XmlDomGetDTD() 3-27
XmlDomGetDecl() 3-28
XmlDomGetDocElem() 3-29
XmlDomGetDocElemByID() 3-29
XmlDomGetDocElemsByTag() 3-30
XmlDomGetDocElemsByTagNS() 3-31
XmlDomGetLastError() 3-32
XmlDomGetSchema() 3-32
XmlDomImportNode() 3-33
XmlDomIsSchemaBased() 3-34
XmlDomSaveString() 3-34
XmlDomSaveString2() 3-35
XmlDomSetBaseURI() 3-35
XmlDomSetDTD() 3-36
XmlDomSetDocOrder() 3-37
XmlDomSetLastError() 3-37
XmlDomSync() 3-38
DocumentType Interface for DOM XML C APIs 3-38
XmlDomGetDTDEntities() 3-38
XmlDomGetDTDInternalSubset() 3-39
XmlDomGetDTDName() 3-40
XmlDomGetDTDNotations() 3-40
XmlDomGetDTDPubID() 3-41
XmlDomGetDTDSysID() 3-41
Element Interface for DOM XML C APIs 3-42
XmlDomGetAttr() 3-43
XmlDomGetAttrNS() 3-43
XmlDomGetAttrNode() 3-44
XmlDomGetAttrNodeNS() 3-45
XmlDomGetChildrenByTag() 3-45
XmlDomGetChildrenByTagNS() 3-46
XmlDomGetElemsByTag() 3-47
XmlDomGetElemsByTagNS() 3-47
XmlDomGetTag() 3-48
XmlDomHasAttr() 3-49
XmlDomHasAttrNS() 3-49
XmlDomRemoveAttr() 3-50

XmlDomRemoveAttrNS() 3-51
XmlDomRemoveAttrNode() 3-51
XmlDomSetAttr() 3-52
XmlDomSetAttrNS() 3-53
XmlDomSetAttrNode() 3-53
XmlDomSetAttrNodeNS() 3-54
Entity Interface for DOM XML C APIs 3-55
XmlDomGetEntityNotation() 3-55
XmlDomGetEntityPubID() 3-55
XmlDomGetEntitySysID() 3-56
XmlDomGetEntityType() 3-57
NamedNodeMap Interface for DOM XML C APIs 3-57
XmlDomGetNamedItem() 3-58
XmlDomGetNamedItemNS() 3-58
XmlDomGetNodeMapItem() 3-59
XmlDomGetNodeMapLength() 3-60
XmlDomRemoveNamedItem() 3-60
XmlDomRemoveNamedItemNS() 3-61
XmlDomSetNamedItem() 3-62
XmlDomSetNamedItemNS() 3-62
Node Interface for DOM XML C APIs 3-63
XmlDomAppendChild() 3-65
XmlDomCleanNode() 3-66
XmlDomCloneNode() 3-66
XmlDomFreeNode() 3-67
XmlDomGetAttrs() 3-68
XmlDomGetChildNodes() 3-68
XmlDomGetDefaultNS() 3-69
XmlDomGetFirstChild() 3-69
XmlDomGetFirstPfnsPair() 3-70
XmlDomGetLastChild() 3-70
XmlDomGetNextPfnsPair() 3-71
XmlDomGetNextSibling() 3-72
XmlDomGetNodeLocal() 3-72
XmlDomGetNodeLocalLen() 3-73
XmlDomGetNodeName() 3-74
XmlDomGetNodeNameLen() 3-74
XmlDomGetNodePrefix() 3-75
XmlDomGetNodeType() 3-76
XmlDomGetNodeURI() 3-77
XmlDomGetNodeURILen() 3-78

XmlDomGetNodeValue() 3-79
XmlDomGetNodeValueLen() 3-79
XmlDomGetNodeValueStream() 3-80
XmlDomGetOwnerDocument() 3-81
XmlDomGetParentNode() 3-81
XmlDomGetPrevSibling() 3-82
XmlDomGetPullNodeAsBinaryStream() 3-82
XmlDomGetPullNodeAsCharacterStream() 3-83
XmlDomGetPushNodeAsBinaryStream() 3-83
XmlDomGetPushNodeAsCharacterStream() 3-84
XmlDomGetSourceEntity() 3-84
XmlDomGetSourceLine() 3-85
XmlDomGetSourceLocation() 3-85
XmlDomHasAttrs() 3-85
XmlDomHasChildNodes() 3-86
XmlDomInsertBefore() 3-86
XmlDomNormalize() 3-87
XmlDomNumAttrs() 3-87
XmlDomNumChildNodes() 3-88
XmlDomPrefixToURI() 3-88
XmlDomRemoveChild() 3-89
XmlDomRenameNode() 3-89
XmlDomRenameNodeNS() 3-90
XmlDomReplaceChild() 3-91
XmlDomSetDefaultNS() 3-91
XmlDomSetNodePrefix() 3-92
XmlDomSetNodeValue() 3-92
XmlDomSetNodeValueLen() 3-93
XmlDomSetNodeValueStream() 3-94
XmlDomSetPullNodeAsBinaryStream() 3-94
XmlDomSetPullNodeAsCharacterStream() 3-95
XmlDomSetPushNodeAsBinaryStream() 3-95
XmlDomSetPushNodeAsCharacterStream() 3-96
XmlDomValidate() 3-96
NodeList Interface for DOM XML C APIs 3-97
XmlDomFreeNodeList() 3-97
XmlDomGetNodeListItem() 3-97
XmlDomGetNodeListLength() 3-98
Notation Interface for DOM XML C APIs 3-99
XmlDomGetNotationPubID() 3-99
XmlDomGetNotationSysID() 3-99

ProcessingInstruction Interface for DOM XML C APIs 3-100
XmlDomGetPIData() 3-100
XmlDomGetPITarget() 3-101
XmlDomSetPIData() 3-101
Text Interface for DOM XML C APIs 3-102
XmlDomSplitText() 3-102

4 Package Event for XML C APIs

XmlEvCleanPPCtx() 4-5
XmlEvCreatePPCtx() 4-5
XmlEvCreateSVCtx() 4-6
XmlEvDestroyPPCtx() 4-7
XmlEvDestroySVCtx() 4-7
XmlEvGetAttrCount() 4-8
XmlEvGetAttrDeclBody() 4-8
XmlEvGetAttrDeclBody0() 4-9
XmlEvGetAttrDeclCount() 4-9
XmlEvGetAttrDeclElName() 4-9
XmlEvGetAttrDeclElName0() 4-10
XmlEvGetAttrDeclLocalName() 4-10
XmlEvGetAttrDeclLocalName0() 4-11
XmlEvGetAttrDeclName() 4-11
XmlEvGetAttrDeclName0() 4-12
XmlEvGetAttrDeclPrefix() 4-12
XmlEvGetAttrDeclPrefix0() 4-13
XmlEvGetAttrID() 4-13
XmlEvGetAttrLocalName() 4-14
XmlEvGetAttrLocalName0() 4-14
XmlEvGetAttrName() 4-14
XmlEvGetAttrName0() 4-15
XmlEvGetAttrPrefix() 4-15
XmlEvGetAttrPrefix0() 4-16
XmlEvGetAttrURI() 4-16
XmlEvGetAttrURI0() 4-17
XmlEvGetAttrUriID() 4-17
XmlEvGetAttrValue() 4-18
XmlEvGetAttrValue0() 4-18
XmlEvGetElDeclContent() 4-18
XmlEvGetElDeclContent0() 4-19
XmlEvGetEncoding() 4-19

XmlEvGetError() 4-20
XmlEvGetName() 4-20
XmlEvGetName0() 4-21
XmlEvGetLocalName() 4-21
XmlEvGetLocalName0() 4-22
XmlEvGetLocation() 4-22
XmlEvGetPIData() 4-23
XmlEvGetPIData0() 4-23
XmlEvGetPITarget() 4-23
XmlEvGetPITarget0() 4-24
XmlEvGetPEIsGen() 4-24
XmlEvGetPERepl() 4-25
XmlEvGetPERepl0() 4-25
XmlEvGetPrefix() 4-25
XmlEvGetPrefix0() 4-26
XmlEvGetPubId() 4-26
XmlEvGetPubId0() 4-27
XmlEvGetSysId() 4-27
XmlEvGetSysId0() 4-28
XmlEvGetTagID() 4-28
XmlEvGetTagUriID() 4-29
XmlEvGetText() 4-29
XmlEvGetText0() 4-30
XmlEvGetUENdata() 4-30
XmlEvGetUENdata0() 4-31
XmlEvGetURI() 4-31
XmlEvGetURI0() 4-31
XmlEvGetVersion() 4-32
XmlEvIsEncodingSpecified() 4-32
XmlEvIsNamespaceAttr() 4-33
XmlEvIsStandalone() 4-33
XmlEvNext() 4-33
XmlEvNextTag() 4-34
XmlEvLoadPPDoc() 4-34
XmlEvSchemaValidate() 4-35

5 Package Orastream for XML C APIs

OraStreamClose() 5-2
OraStreamHandle() 5-2
OraStreamInit() 5-3

OraStreamIsOpen() 5-4
OraStreamOpen() 5-4
OraStreamRead() 5-4
OraStreamReadable() 5-5
OraStreamReadChar() 5-5
OraStreamSid() 5-6
OraStreamTerm() 5-6
OraStreamWrite() 5-7
OraStreamWritable() 5-7
OraStreamWriteChar() 5-7

6 Package Range for XML C APIs

XmlDomCreateRange() 6-1
XmlDomRangeClone() 6-2
XmlDomRangeCloneContents() 6-2
XmlDomRangeCollapse() 6-3
XmlDomRangeCompareBoundaryPoints() 6-3
XmlDomRangeDeleteContents() 6-4
XmlDomRangeDetach() 6-4
XmlDomRangeExtractContents() 6-5
XmlDomRangeGetCollapsed() 6-5
XmlDomRangeGetCommonAncestor() 6-6
XmlDomRangeGetDetached() 6-6
XmlDomRangeGetEndContainer() 6-7
XmlDomRangeGetEndOffset() 6-7
XmlDomRangeGetStartContainer() 6-8
XmlDomRangeGetStartOffset() 6-8
XmlDomRangeIsConsistent() 6-9
XmlDomRangeSelectNode() 6-9
XmlDomRangeSelectNodeContents() 6-10
XmlDomRangeSetEnd() 6-10
XmlDomRangeSetEndBefore() 6-11
XmlDomRangeSetStart() 6-11
XmlDomRangeSetStartAfter() 6-12
XmlDomRangeSetStartBefore() 6-13

7 Package SAX for XML C APIs

XmlSaxAttributeDecl() 7-1
XmlSaxBeginGen() 7-2

XmlSaxCDATA() 7-2
XmlSaxCharacters() 7-3
XmlSaxComment() 7-4
XmlSaxElementDecl() 7-4
XmlSaxEndDocument() 7-5
XmlSaxEndElement() 7-5
XmlSaxEndGen() 7-5
XmlSaxNotationDecl() 7-6
XmlSaxPI() 7-6
XmlSaxParsedEntityDecl() 7-6
XmlSaxStartDocument() 7-7
XmlSaxStartElement() 7-8
XmlSaxStartElementNS() 7-8
XmlSaxUnparsedEntityDecl() 7-9
XmlSaxWhitespace() 7-10
XmlSaxXmlDecl() 7-11

8 Package Schema for XML C APIs

XmlSchemaCreate() 8-1
XmlSchemaDestroy() 8-2
XmlSchemaErrorWhere() 8-2
XmlSchemaLoad() 8-3
XmlSchemaLoadedList() 8-4
XmlSchemaSetErrorHandler() 8-4
XmlSchemaSetValidateOptions() 8-5
XmlSchemaTargetNamespace() 8-6
XmlSchemaUnload() 8-6
XmlSchemaValidate() 8-7
XmlSchemaVersion() 8-8
XmlSchemaClean() 8-8

9 Package SOAP for XML C APIs

XmlSoapAddBodyElement() 9-2
XmlSoapAddFaultReason() 9-3
XmlSoapAddFaultSubDetail() 9-3
XmlSoapAddHeaderElement() 9-4
XmlSoapCall() 9-5
XmlSoapCreateConnection() 9-5
XmlSoapCreateCtx() 9-7

XmlSoapCreateMsg() 9-8
XmlSoapDestroyConnection() 9-8
XmlSoapDestroyCtx() 9-9
XmlSoapDestroyMsg() 9-9
XmlSoapError() 9-10
XmlSoapGetBody() 9-10
XmlSoapGetBodyElement() 9-11
XmlSoapGetEnvelope() 9-12
XmlSoapGetFault() 9-12
XmlSoapGetHeader() 9-13
XmlSoapGetHeaderElement() 9-14
XmlSoapGetMustUnderstand() 9-14
XmlSoapGetReasonLang() 9-15
XmlSoapGetReasonNum() 9-16
XmlSoapGetRelay() 9-16
XmlSoapGetRole() 9-17
XmlSoapHasFault() 9-17
XmlSoapSetFault() 9-18
XmlSoapSetMustUnderstand() 9-19
XmlSoapSetRelay() 9-20
XmlSoapSetRole() 9-20

10 Package Traversal for XML C APIs

DocumentTraversal Interface for Traversal XML C APIs 10-1
XmlDomCreateNodeIter() 10-1
XmlDomCreateTreeWalker() 10-2
NodeFilter Interface for Traversal XML C APIs 10-3
XMLDOM_ACCEPT_NODE_F() 10-3
NodeIterator Interface for Traversal XML C APIs 10-4
XmlDomIterDetach() 10-4
XmlDomIterNextNode() 10-5
XmlDomIterPrevNode() 10-5
TreeWalker Interface for Traversal XML C APIs 10-6
XmlDomWalkerFirstChild() 10-7
XmlDomWalkerGetCurrentNode() 10-7
XmlDomWalkerGetRoot() 10-8
XmlDomWalkerLastChild() 10-8
XmlDomWalkerNextNode() 10-9
XmlDomWalkerNextSibling() 10-9
XmlDomWalkerParentNode() 10-10

XmlDomWalkerPrevNode() 10-11
XmlDomWalkerPrevSibling() 10-11
XmlDomWalkerSetCurrentNode() 10-12
XmlDomWalkerSetRoot() 10-12

11 Package XML for XML C APIs

XmlAccess() 11-1
XmlCreate() 11-2
XmlCreateDTD() 11-5
XmlCreateDocument() 11-6
XmlDestroy() 11-6
XmlDiff() 11-7
XmlFreeDocument() 11-8
XmlGetEncoding() 11-8
XmlHasFeature() 11-9
XmlIsSimple() 11-10
XmlIsUnicode() 11-10
XmlLoadDom() 11-11
XmlLoadSax() 11-12
XmlLoadSaxVA() 11-13
XmlSaveDom() 11-13
XmlVersion() 11-14

12 Package XmlDiff for XML C APIs

XmlDiff() 12-1
XmlHash() 12-3
XmlPatch() 12-4

13 Package XPath for XML C APIs

XmlXPathCreateCtx() 13-1
XmlXPathDestroyCtx() 13-2
XmlXPathEval() 13-2
XmlXPathGetObjectBoolean() 13-3
XmlXPathGetObjectFragment() 13-3
XmlXPathGetObjectNSetNode() 13-4
XmlXPathGetObjectNSetNum() 13-4
XmlXPathGetObjectNumber() 13-5
XmlXPathGetObjectString() 13-5
XmlXPathGetObjectType() 13-6

XmlXPathParse() 13-6

14 Package XPointer for XML C APIs

XPointer Interface for XPointer XML C APIs 14-1
XmlXPointerEval() 14-1
XPtrLoc Interface for XPointer XML C APIs 14-1
XmlXPtrLocGetNode() 14-2
XmlXPtrLocGetPoint() 14-2
XmlXPtrLocGetRange() 14-2
XmlXPtrLocGetType() 14-3
XmlXPtrLocToString() 14-3
XPtrLocSet Interface for XPointer XML C APIs 14-4
XmlXPtrLocSetFree() 14-4
XmlXPtrLocSetGetItem() 14-4
XmlXPtrLocSetGetLength() 14-5

15 Package XSLT for XML C APIs

XmlXslCreate() 15-1
XmlXslDestroy() 15-2
XmlXslGetBaseURI() 15-2
XmlXslGetOutput() 15-3
XmlXslGetStylesheetDom() 15-3
XmlXslGetTextParam() 15-3
XmlXslProcess() 15-4
XmlXslResetAllParams() 15-4
XmlXslSetOutputDom() 15-5
XmlXslSetOutputEncoding() 15-5
XmlXslSetOutputMethod() 15-5
XmlXslSetOutputSax() 15-6
XmlXslSetOutputStream() 15-6
XmlXslSetTextParam() 15-7

16 Package XSLTVM for XML C APIs

XSLTC Interface of XSLTVM for XML C APIs 16-1
XmlXvmCompileBuffer() 16-1
XmlXvmCompileDom() 16-1
XmlXvmCompileFile() 16-2
XmlXvmCompileURI() 16-3
XmlXvmCompileXPath() 16-4

XmlXvmCreateComp() 16-4
XmlXvmDestroyComp() 16-5
XmlXvmGetBytecodeLength() 16-5
XSLTVM Interface of XSLTVM for XML C APIs 16-6
XMLXVM_DEBUG_F() 16-7
XmlXvmCreate() 16-7
XmlXvmDestroy() 16-8
XmlXvmEvaluateXPath() 16-8
XmlXvmGetObjectBoolean() 16-9
XmlXvmGetObjectNSetNode() 16-10
XmlXvmGetObjectNSetNum() 16-10
XmlXvmGetObjectNumber() 16-11
XmlXvmGetObjectString() 16-11
XmlXvmGetObjectType() 16-12
XmlXvmGetOutputDom() 16-12
XmlXvmResetParams() 16-13
XmlXvmSetBaseURI() 16-13
XmlXvmSetBytecodeBuffer() 16-13
XmlXvmSetBytecodeFile() 16-14
XmlXvmSetBytecodeURI() 16-14
XmlXvmSetDebugFunc() 16-15
XmlXvmSetOutputDom() 16-15
XmlXvmSetOutputEncoding() 16-16
XmlXvmSetOutputSax() 16-16
XmlXvmSetOutputStream() 16-17
XmlXvmSetTextParam() 16-17
XmlXvmTransformBuffer() 16-18
XmlXvmTransformDom() 16-18
XmlXvmTransformFile() 16-19
XmlXvmTransformURI() 16-20

17 JSON DOM Interfaces

JSON DOM Functions 17-1
JsonDomGetRoot() 17-2
JsonDomSetRoot() 17-2
JsonDomGetNumFields() 17-3
JsonDomGetArraySize() 17-4
JsonDomGetElements() 17-4
JsonDomGetNodeType() 17-5
JsonDomGetScalarInfoOci() 17-6

JsonDomCreateObject() 17-7
JsonDomCreateOCIInterval() 17-7
JsonDomCreateArray() 17-8
JsonDomCreateString() 17-8
JsonDomCreateBinary() 17-9
JsonDomCreateFloat() 17-9
JsonDomCreateDouble() 17-10
JsonDomCreateBoolean() 17-10
JsonDomCreateOCINumber() 17-11
JsonDomCreateOCIDate() 17-11
JsonDomCreateOCIDateTime() 17-12
JsonDomCreateNull() 17-12
JsonDomSetField() 17-13
JsonDomAppendElement() 17-14
JsonDomFreeNode() 17-14
JsonDomGetError() 17-15
JsonDomGetErrorMessage() 17-15
JsonDomClearError() 17-16
Constants 17-16

A Mapping of APIs used before Oracle Database 10g Release 1

C Package Changes A-1
Initializing and Parsing Sequence Changes A-1
Datatype Mapping between oraxml and xml Packages A-3
Method Mapping between oraxml and xml Packages A-4

Index

List of Tables

1-1	Summary of Datatypes for XML C Implementation	1-1
2-1	Summary of Callback Methods for XML C Implementation	2-1
3-1	Summary of Attr DOM Methods for XML C Implementation	3-1
3-2	Summary of CharacterData DOM Methods for XML C Implementation	3-11
3-3	Summary of Document DOM Methods for XML C Implementation	3-17
3-4	Summary of DocumentType DOM Methods for XML C Implementation	3-38
3-5	Summary of Element DOM Methods for XML C Implementation	3-42
3-6	Summary of Entity DOM Methods for XML C Implementation	3-55
3-7	Summary of NamedNodeMap DOM Methods for XML C Implementation	3-57
3-8	Summary of Node DOM Methods for XML C Implementation	3-63
3-9	Summary of NodeList DOM Methods for XML C Implementation	3-97
3-10	Summary of NodeList DOM Methods for XML C Implementation	3-99
3-11	Summary of ProcessingInstruction DOM Package Methods for XML C Implementation	3-100
3-12	Summary of Text DOM Methods for XML C Implementation	3-102
4-1	Summary of Event Methods for XML C Implementation	4-1
5-1	Orastream Error Codes for XML C Implementation	5-1
5-2	Summary of OraStream Methods for XML C Implementation	5-1
6-1	Summary of Range Methods for XML C Implementation	6-1
7-1	Summary of SAX Methods for XML C Implementation	7-1
8-1	Summary of Schema Methods for XML C Implementation	8-1
9-1	Summary of SOAP Methods for XML C Implementation	9-1
10-1	Summary of DocumentTraversal Traversal Methods for XML C Implementation	10-1
10-2	Summary of NodeFilter Traversal Methods for XML C Implementation	10-3
10-3	Summary of NodeIterator Traversal Methods for XML C Implementation	10-4
10-4	Summary of TreeWalker Traversal Methods for XML C Implementation	10-6
11-1	Summary of XML Methods for XML C Implementation	11-1
12-1	Summary of XmlDiff Methods for XML C Implementation	12-1
13-1	Summary of XPath Methods for XML C Implementation	13-1
14-1	Summary of XPointer XPointer Methods for XML C Implementation	14-1
14-2	Summary of XPtrLoc XPointer Methods for XML C Implementation	14-2
14-3	Summary of XPtrLocSet XPointer Methods for XML C Implementation	14-4
15-1	Summary of XSLT Methods for XML C Implementation	15-1
16-1	Summary of XSLTC XSLTVM Methods for XML C Implementation	16-1
16-2	Summary of XSLTVM XSLTVM Methods for XML C Implementation	16-6
17-1	Summary of DOM Methods for JSON C Implementation	17-1

A-1 Datatypes Supported by oraxml Package versus xml Package A-3
A-2 Methods of the oraxml Package versus the xml Package A-4

Preface

The *Oracle Database XML C API Reference* describes Oracle XML Developer's Kits (XDK) and Oracle XML DB APIs for the C programming language. It primarily lists the syntax of functions, methods, and procedures associated with these APIs.

Audience

Oracle Database XML C API Reference is intended for developers who are building XML applications in Oracle.

To use this document, you need a basic understanding of object-oriented programming concepts, familiarity with Structured Query Language (SQL), and working knowledge of application development using the C programming language.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Database 12*c* Release 2 (12.2.0.1) documentation set:

- *Oracle Database Concepts*
- *Oracle Database SQL Language Reference*
- *Oracle Database Object-Relational Developer's Guide*
- *Oracle Database New Features Guide*
- *Oracle XML Developer's Kit Programmer's Guide*
- *Oracle XML DB Developer's Guide*
- *Oracle Database Sample Schemas*

Additional information may be found at `http://www.oracle.com/technetwork/database-features/xdk/overview/index.html`

Conventions

Oracle documentation uses font conventions to specify information type.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Datatypes for XML C APIs

The following table lists all C datatypes and their descriptions.

Table 1-1 Summary of Datatypes for XML C Implementation

Datatype	Purpose
oracheck	Checkword for validating data structures.
oraerr	Error code: `0` is success, non-`0` is failure.
oraprop_id	The id of property; if `>= 0` it is valid, if `< 0`, it is invalid.
oramemctx	Opaque memory context.
oraprop	Property name.
oraprop_t	Property value type.
oraprop_v	Value: union of storage for all data types.
orastream	Opaque stream object.
orastreamhdl	Storage for file handles.
xmlcmphow	Constant used for DOM Range comparisons.
xmlctx	Context shared for all documents in an XML session.
xmldfsrct	Specifies input types for `XmlDiff` operations
xmlerr	Numeric error code returned by many functions.
xmlevctx	XML Event context.
xmlhasht	The hash value of an XML tree or sub-tree; also known as a digest.
xmlistream	Generic user-defined input stream.
xmliter	Control structure for DOM2 `NodeIterator` and `TreeWalker`.
xmlnodetype	The numeric type code of a node.
xmlostream	Generic user-defined output stream.
xmlpoint	XPointer point location.
xmlrange	Controls structure for DOM2 Range.
xmlsoapbind	Binding for SOAP connections.
xmlsoapcon	SOAP connection object.
xmlsoapctx	Context for SOAP operations.
xmlsoaprole	Role for a SOAP node.
xmlshowbits	Bit flags used to select which node types to show.
xmlurlacc	This is an enumeration of the known access methods for retrieving data from a URL.
xmlurlhdl	This union contains the handle(s) needed to access URL data, be it a stream or `stdio` pointer, file descriptor(s), and so on.
xmlurlpart	This structure contains the sub-parts of a URL.
xmlxptrloc	XPointer location datatype.

Table 1-1 (Cont.) Summary of Datatypes for XML C Implementation

Datatype	Purpose
xmlxptrlocset	XPointer location set datatype.
xmlxslobjtype	Type of XSLT object that may be returned.
xmlxslomethod	Type of output produced by the XSLT processor.
xmlxvm	An object of type `xmlxvm` is used for XML document transformation.
xmlxvmcomp	An object of type `xmlxvmcomp` is used for compiling XSL stylesheets.
xmlxvmflags	Control flags for the XSLT compiler.
xmlxvmobjtype	Type of XSLTVM object.
xpctx	XPath top-level context.
xpexpr	XPath expression.
xpobj	XPath object.
xsdctx	XMLSchema validator context.
xslctx	XSL top-level context.
xvmobj	XSLVM processor run-time object; contents are private and must not be accessed by users.
JsonDomDoc	JSON DOM tree container.
JsonDomNode	Opaque JSON DOM node abstract type.
JsonDomObject	Opaque JSON DOM object type.
JsonDomArray	Opaque JSON DOM array type.
JsonDomScalar	Opaque JSON DOM scalar type.
jznnodetype	Enum type that indicates the type of DOM node.
jznvaltype	Enum type that indicates the type of scalar value.
jznScalarVal	DOM Scalar Value Output Structure.
JsonOCIVal	Union of helper output structured (OCINumber, JsonDateTime, JsonDayInterval and JsonYearInterval) forms.
JsonDateTime	Structure for date and timestamp scalar output values.
JsonDayInterval	Structure for day-second interval scalar output values.
JsonYearInterval	Structure for year-month interval scalar output values.
jsonerr	Parameter to handle DOM API errors.

oracheck

Checkword for validating data structures.

Definition

```
typedef ub4 oracheck;
```

oraerr

Error code: `0` is success, non-`0` is failure.

Definition

```
typedef ub4 oraerr;
```

oraprop_id

The id of property; if >= 0 it is valid, if < 0, it is invalid.

Definition

```
typedef sb2 oraprop_id;
```

oramemctx

Opaque memory context.

Definition

```
typedef struct oramemctx oramemctx;
```

oraprop

Property name.

Definition

```
typedef struct oraprop {
    oratext    *name_oraprop;
    oraprop_id  id_oraprop;
    oraprop_t   type_oraprop;
    oraprop_v   value_oraprop;
} oraprop;
```

oraprop_t

Property value type.

Definition

```
typedef enum {
    ORAPROP_TYPE_BOOLEAN,
    ORAPROP_TYPE_SIGNED,
    ORAPROP_TYPE_UNSIGNED,
    ORAPROP_TYPE_POINTER
} oraprop_t;
```

oraprop_v

Value: union of storage for all data types.

Definition

```
typedef union oraprop_v {
    boolean b_oraprop_v;
```

```
    sb4     s_oraprop_v;
    ub4     u_oraprop_v;
    void   *p_oraprop_v;
} oraprop_v;
```

orastream

Opaque stream object.

Definition

```
typedef struct orastream orastream;
```

orastreamhdl

Storage for file handles.

Definition

```
typedef union orastreamhdl {
    void  *ptr_orastreamhdl;    /* generic pointer stream/file/etc */
    struct {
        sb4 fd_orastreamhdl;    /* file descriptor(s) [FTP needs all 3!] */
        sb4 fd2_orastreamhdl;
        sb4 fd3_orastreamhdl;
    } fds_lpihdl;
} orastreamhdl;
```

xmlcmphow

Constant used for DOM Range comparisons.

Definition

```
typedef enum {
    XMLDOM_START_TO_START ,
    XMLDOM_START_TO_END   ,
    XMLDOM_END_TO_END     ,
    XMLDOM_END_TO_START
} xmlcmphow;
```

xmlctx

Context shared for all documents in an XML session. Contains encoding information, low-level memory allocation function pointers, error message language or encoding and optional handler function, and so on. Required to load (parse) documents and create DOM, generate SAX, and so on.

Definition

```
struct xmlctx;
typedef struct xmlctx xmlctx;
```

xmldfoptype

Operation type, represents one or more operations. Used for passing the `diff` to a custom Operation Buildder (OB) in XmlDiff()XmlDiff().

Definition

```
typedef enum {
  XMLDF_OP_NONE, /* Should not be set to non-zero for XMLDF_NUM_OP macro below */
  XMLDF_OP_UPDATE,
  XMLDF_OP_RENAME,
  XMLDF_OP_DELETE,
  XMLDF_OP_INSERT_BEFORE,
  XMLDF_OP_APPEND
} xmldfoptype;
```

xmldfsrct

Specifies input types for `XmlDiff` operations.

Definition

```
typedef enum {
   XMLDF_SRCT_NONE ,    /* default is DOM */
   XMLDF_SRCT_DOM,      /* DOM: doc node must be specified */
   XMLDF_SRCT_FILE,     /* file name must be specified */
   XMLDF_SRCT_URL,      /* URL in compiler encoding */
   XMLDF_SRCT_BUFFER,   /* buffer: buffer pointer and length must be specified */
   XMLDF_SRCT_FILEP,    /* FILE */
   XMLDF_SRCT_OSTREAM,  /* orastream: stream pointer must be specified */
   XMLDF_SRCT_DOMNODE   /* DOM node, used with XmlHash() */
} xmldfsrct;
```

xmlerr

Numeric error code returned by many functions. A zero value indicates success; a nonzero value indicates error.

Definition

```
typedef enum {
   XMLERR_OK              , /* success return */
   XMLERR_NULL_PTR        , /* NULL pointer */
   XMLERR_NO_MEMORY       , /* out of memory */
   XMLERR_HASH_DUP        , /* duplicate entry in hash table */
   XMLERR_INTERNAL        ,  /* internal error */
   XMLERR_BUFFER_OVERFLOW ,  /* name/quoted string too long */
   XMLERR_BAD_CHILD       ,  /* invalid child for parent */
   XMLERR_EOI             ,  /* unexpected EndOfInformation */
   XMLERR_BAD_MEMCB       ,  /* invalid memory callbacks */
   XMLERR_UNICODE_ALIGN   , /* Unicode data misalignment */
   XMLERR_NODE_TYPE       , /* wrong node type */
   XMLERR_UNCLEAN         , /* context is not clean */
   XMLERR_NESTED_STRINGS  , /* internal: nested open str */
   XMLERR_PROP_NOT_FOUND  , /* property not found */
   XMLERR_SAVE_OVERFLOW   , /* save output overflowed */
```

```
   XMLERR_NOT_IMP         , /* feature not implemented */
   XMLERR_NLS_MISMATCH    , /* specify lxglo/lxd or neither*/
   XMLERR_NLS_INIT        , /* error at NLS initialization */
   XMLERR_LEH_INIT        , /* error at LEH initialization */
   XMLERR_LML_INIT        , /* error at LML initialization */
   XMLERR_LPU_INIT          /* error at LPU initialization */
} xmlerr;
```

xmlevctx

XML Event context.

Definition

```
typedef  struct {
   void *ctx_xmlevctx;       /* implementation specific context */
   xmlevdisp disp_xmlevctx;  /* dispatch table */
   ub4 checkword_xmlevctx;   /* checkword for integrity check */
   ub4 flags_xmlevctx;       /* mode; default: expand_entity */
   struct xmlevctx;          /* input xmlevctx; chains the XML Event context
*/
} xmlevctx;
```

xmlevtype

The event type for parser pull events.

Definition

```
typedef enum  xmlevtype {
   XML_EVENT_FATAL_ERROR,            /* Fatal Error */
   XML_EVENT_BEFORE_START,           /* Before Start Document */
   XML_EVENT_START_DOCUMENT,         /* Indicates Start Document */
   XML_EVENT_START_DTD,              /* Start DTD */
   XML_EVENT_END_DTD,                /* End DTD */
   XML_EVENT_NOTATION_DECLARATION,   /* Notation Decl */
   XML_EVENT_PE_DECLARATION,         /* PE Decl */
   XML_EVENT_UE_DECLARATION,         /* US Decl */
   XML_EVENT_ELEMENT_DECLARATION,    /* Element Decl */
   XML_EVENT_ATTLIST_DECLARATION,    /* Attribute Decl */
   XML_EVENT_START_ELEMENT,          /* Start Element */
   XML_EVENT_END_ELEMENT,            /* End Element */
   XML_EVENT_CHARACTERS,             /* Characters (text) */
   XML_EVENT_CHARACTERS_CONT,        /* Characters Continued */
   XML_EVENT_PI,                     /* Processing Instruction */
   XML_EVENT_PI_CONT,                /* Processing Instruction Continued */
   XML_EVENT_COMMENT,                /* Comment */
   XML_EVENT_COMMENT_CONT,           /* Comment Continued */
   XML_EVENT_SPACE,                  /* White Space */
   XML_EVENT_SPACE_CONT,             /* White Space Continued */
   XML_EVENT_ENTITY_REFERENCE,       /* Entity Reference */
   XML_EVENT_CDATA,                  /* CDATA */
   XML_EVENT_CDATA_CONT,             /* CDATA continued */
   XML_EVENT_START_ENTITY,           /* Start Entity */
   XML_EVENT_END_ENTITY,             /* End Entity */
   XML_EVENT_END_DOCUMENT,           /* End Document */
   XML_EVENT_ERROR                   /* Error */
}xmlevtype;
```

xmlhasht

The hash value of an XML tree or sub-tree; also known as a digest.

If the hash values for two XML trees are equal, the trees are considered equal, to a very high probability; uses the MD5 algorithm.

Definition

```
struct xmlhasht {
   ub4  l_xmlhasht;   /* lenght of digest in bytes */
   ub1  d_xmlhasht[XMLDF_DIGEST_MAX];   /* the digest */
};
typedef struct xmlhasht xmlhasht;
```

xmlistream

Generic user-defined input stream. The three function pointers are required (but may be stubs). The context pointer is entirely user-defined; point it to whatever state information is required to manage the stream; it will be passed as first argument to the user functions.

Definition

```
typedef struct xmlistream {
   XML_STREAM_OPEN_F(
      (*open_xmlistream),
      xctx,
      sctx,
      path,
      parts,
      length);
   XML_STREAM_READ_F(
      (*read_xmlistream),
      xctx,
      sctx,
      path,
      dest,
      size,
      nraw, eoi);
   XML_STREAM_CLOSE_F(
      (*close_xmlistream),
      xctx,
      sctx);
   void *ctx_xmlistream;                          /* user's stream context */
} xmlistream;
```

xmliter

Control structure for DOM 2 `NodeIterator` and `TreeWalker`.

Definition

```
struct xmliter {
   xmlnode *root_xmliter;   /* root node of the iteration space */
   xmlnode *cur_xmliter;    /* current position iterator ref node */
   ub4      show_xmliter;   /* node filter mask */
```

```
   void    *filt_xmliter;   /* node filter function */
   boolean  attach_xmliter; /* is iterator valid? */
   boolean  expan_xmliter;  /* are external entities expanded? */
   boolean  before_xmliter; /* iter position before ref node? */
};
typedef struct xmliter xmliter;
typedef struct xmliter xmlwalk;
```

xmlnodetype

The numeric type code of a node. `0` means invalid, 1-13 are the standard numberings from DOM 1.0, and higher numbers are for internal use only.

Definition

```
typedef enum {
    XMLDOM_NONE     , /* bogus node */
    XMLDOM_ELEM     , /* element */
    XMLDOM_ATTR     , /* attribute */
    XMLDOM_TEXT     , /* char data not escaped by CDATA */
    XMLDOM_CDATA    , /* char data escaped by CDATA */
    XMLDOM_ENTREF   , /* entity reference */
    XMLDOM_ENTITY   , /* entity */
    XMLDOM_PI       , /* <?processing instructions?> */
    XMLDOM_COMMENT  , /* <!-- Comments --> */
    XMLDOM_DOC      , /* Document */
    XMLDOM_DTD      , /* DTD */
    XMLDOM_FRAG     , /* Document fragment */
    XMLDOM_NOTATION , /* notation */

    /* Oracle extensions from here on */
    XMLDOM_ELEMDECL , /* DTD element declaration */
    XMLDOM_ATTRDECL , /* DTD attribute declaration */

    /* Content Particles (nodes in element's Content Model) */
    XMLDOM_CPELEM   , /* element */
    XMLDOM_CPCHOICE , /* choice (a|b) */
    XMLDOM_CPSEQ    , /* sequence (a,b) */
    XMLDOM_CPPCDATA , /* #PCDATA */
    XMLDOM_CPSTAR   , /* '*' (zero or more) */
    XMLDOM_CPPLUS   , /* '+' (one or more) */
    XMLDOM_CPOPT    , /* '?' (optional) */
    XMLDOM_CPEND      /* end marker */
} xmlnodetype;
```

xmlostream

Generic user-defined output stream. The three function pointers are required (but may be stubs). The context pointer is entirely user-defined; point it to whatever state information is required to manage the stream; it will be passed as first argument to the user functions.

Definition

```
typedef struct xmlostream {
   XML_STREAM_OPEN_F(
      (*open_xmlostream),
      xctx,
```

```
      sctx,
      path,
      parts,
      length);
   XML_STREAM_WRITE_F(
      (*write_xmlostream),
      xctx,
      sctx,
      path,
      src,
      size);
   XML_STREAM_CLOSE_F(
      (*close_xmlostream),
      xctx,
      sctx);
   void *ctx_xmlostream;        /* user's stream context */
} xmlostream;
```

xmlpoint

XPointer point location.

Definition

```
typedef struct xmlpoint xmlpoint;
```

xmlrange

Control structure for DOM 2 Range.

Definition

```
typedef struct xmlrange {
 xmlnode *startnode_xmlrange;  /* start point container */
 ub4      startofst_xmlrange;  /* start point index */
 xmlnode *endnode_xmlrange;    /* end point container */
 ub4      endofst_xmlrange;    /* end point index */
 xmlnode *doc_xmlrange;        /* document node */
 xmlnode *root_xmlrange;       /* root node of the range */
 boolean  collapsed_xmlrange;  /* is range collapsed? */
 boolean  detached_xmlrange;   /* range invalid, invalidated?*/
} xmlrange;
```

xmlsoapbind

Binding for SOAP connections. SOAP does not dictate the binding (transport) used for conveying messages; however the HTTP protocol is well-defined and currently the only choice.

Definition

```
typedef enum xmlsoapbind {
   XMLSOAP_BIND_NONE ,  /* none */
   XMLSOAP_BIND_HTTP    /* HTTP */ } xmlsoapbind;
```

xmlsoapcon

SOAP connection object. Each distinct connection requires an instance of this type, which contains binding and endpoint information.

Definition

```
typedef struct xmlsoapcon xmlsoapcon;
```

xmlsoapctx

Context for SOAP operations. Only a single context is needed and it can be shared by several SOAP messages.

Definition

```
typedef struct xmlsoapctx xmlsoapctx;
```

xmlsoaprole

Role for a SOAP node.

Definition

```
typedef enum xmlsoaprole {
    XMLSOAP_ROLE_UNSET,   /* not specified */
    XMLSOAP_ROLE_NONE,    /* "none" */
    XMLSOAP_ROLE_NEXT,    /* "next" */
    XMLSOAP_ROLE_ULT,     /* "ultimateReceiver" */
    XMLSOAP_ROLE_OTHER    /* other - user defined */
} xmlsoaprole;
```

xmlshowbits

Bit flags used to select which nodes types to show.

Definition

```
typedef ub4 xmlshowbits;
#define XMLDOM_SHOW_ALL        ~(ub4)0
#define XMLDOM_SHOW_BIT(ntype) ((ub4)1 << (ntype))
#define XMLDOM_SHOW_ELEM       XMLDOM_SHOW_BIT(XMLDOM_ELEM)
#define XMLDOM_SHOW_ATTR       XMLDOM_SHOW_BIT(XMLDOM_ATTR)
#define XMLDOM_SHOW_TEXT       XMLDOM_SHOW_BIT(XMLDOM_TEXT)
#define XMLDOM_SHOW_CDATA      XMLDOM_SHOW_BIT(XMLDOM_CDATA)
#define XMLDOM_SHOW_ENTREF     XMLDOM_SHOW_BIT(XMLDOM_ENTREF)
#define XMLDOM_SHOW_ENTITY     XMLDOM_SHOW_BIT(XMLDOM_ENTITY)
#define XMLDOM_SHOW_PI         XMLDOM_SHOW_BIT(XMLDOM_PI)
#define XMLDOM_SHOW_COMMENT    XMLDOM_SHOW_BIT(XMLDOM_COMMENT)
#define XMLDOM_SHOW_DOC        XMLDOM_SHOW_BIT(XMLDOM_DOC)
#define XMLDOM_SHOW_DTD        XMLDOM_SHOW_BIT(XMLDOM_DTD)
#define XMLDOM_SHOW_FRAG       XMLDOM_SHOW_BIT(XMLDOM_FRAG)
#define XMLDOM_SHOW_NOTATION   XMLDOM_SHOW_BIT(XMLDOM_NOTATION)
#define XMLDOM_SHOW_DOC_TYPE   XMLDOM_SHOW_BIT(XMLDOM_DOC_TYPE)
```

xmlurlacc

This is an enumeration of the known access methods for retrieving data from a URL. Open/read/close functions may be plugged in to override the default behavior.

Definition

```
typedef enum {
    XML_ACCESS_NONE    , /* not specified */
    XML_ACCESS_UNKNOWN , /* specified but unknown */
    XML_ACCESS_FILE    , /* filesystem access */
    XML_ACCESS_HTTP    , /* HTTP */
    XML_ACCESS_FTP     , /* FTP */
    XML_ACCESS_GOPHER  , /* Gopher */
    XML_ACCESS_ORADB   , /* Oracle DB */
    XML_ACCESS_STREAM    /* user-defined stream */
} xmlurlacc;
```

xmlurlhdl

This union contains the handle(s) needed to access URL data, be it a stream or stdio pointer, file descriptor(s), and so on.

Definition

```
typedef union xmlurlhdl {
    void  *ptr_xmlurlhdl;  /* generic stream/file/... handle */
    struct {
        sb4 fd1_xmlurlhdl; /* file descriptor(s) [FTP needs all 3!] */
        sb4 fd2_xmlurlhdl;
        sb4 fd3_xmlurlhdl;
    } fds_lpihdl;
} xmlurlhdl;
```

xmlurlpart

This structure contains the sub-parts of a URL. The original URL is parsed and the pieces copies (`NULL`-terminated) to a working buffer, then this structure is filled in to point to the parts. Given URL `http://user:pwd@baz.com:8080/pub/baz.html;quux=1?huh#fraggy`, the example component part from this URL will be shown.

Definition

```
typedef struct xmlurlpart {
 xmlurlacc access_xmlurlpart;   /* access method code, XMLACCESS_HTTP */
 oratext   *accbuf_xmlurlpart;  /* access method name: "http" */
 oratext   *host_xmlurlpart;    /* hostname:          "baz.com" */
 oratext   *dir_xmlurlpart;     /* directory:         "pub" */
 oratext   *file_xmlurlpart;    /* filename:          "baz.html" */
 oratext   *uid_xmlurlpart;     /* userid/username:   "user" */
 oratext   *passwd_xmlurlpart;  /* password:          "pwd" */
 oratext   *port_xmlurlpart;    /* port (as string): "8080" */
 oratext   *frag_xmlurlpart;    /* fragment:          "fraggy" */
 oratext   *query_xmlurlpart;   /* query:             "huh" */
 oratext   *param_xmlurlpart;   /* parameter:         "quux=1" */
 ub2        portnum_xmlurlpart; /* port (as number): 8080 */
```

```
 ub1        abs_xmlurlpart;     /* absolute path?     TRUE */
} xmlurlpart;
```

xmlxptrloc

XPointer location data type.

Definition

```
typedef struct xmlxptrloc xmlxptrloc;
```

xmlxptrlocset

XPointer location set data type.

Definition

```
typedef struct xmlxptrlocset xmlxptrlocset;
```

xmlxslobjtype

Type of XSLT object that may be returned.

Definition

```
typedef enum xmlxslobjtype {
    XMLXSL_TYPE_UNKNOWN , /* Not a defined type */
    XMLXSL_TYPE_NDSET   , /* Node-set */
    XMLXSL_TYPE_BOOL    , /* Boolean value */
    XMLXSL_TYPE_NUM     , /* Numeric value (double) */
    XMLXSL_TYPE_STR     , /* String */
    XMLXSL_TYPE_FRAG      /* Document Fragment */
} xmlxslobjtype;
```

xmlxslomethod

Type of output to be produced by the XSLT processor.

Definition

```
typedef enum xmlxslomethod {
 XMLXSL_OUTPUT_UNKNOWN , /* Not defined */
 XMLXSL_OUTPUT_XML     , /* Produce a Document Fragment */
 XMLXSL_OUTPUT_STREAM  , /* Stream out formatted result */
 XMLXSL_OUTPUT_HTML      /* Stream out HTML formatted result */
} xmlxslomethod;
```

xmlxvm

An object of type `xmlxvm` is used for XML documents transformation. The contents of `xmlxvm` are private and must not be accessed by users.

Definition

```
struct xmlxvm;
typedef struct xmlxvm xmlxvm;
```

xmlxvmcomp

An object of type `xmlxvmcomp` is used for compiling XSL stylesheets. The contents of `xmlxvmcomp` are private and must not be accessed by users.

Definition

```
struct xmlxvmcomp;
typedef struct xmlxvmcomp xmlxvmcomp;
```

xmlxvmflags

Control flags for the XSLT compiler.

- `XMLXVM_DEBUG` forces compiler to insert debug information into the bytecode.
- `XMLXVM_STRIPSPACE` forces the same behavior as `xsl:strip-space elements="*"`

Definition

```
typedef ub4 xmlxvmflag;
#define XMLXVM_NOFLAG
#define XMLXVM_DEBUG      /* insert debug info into bytecode */
#define XMLXVM_STRIPSPACE /* same as xsl:strip-space elements="*" */
```

xmlxvmobjtype

Type of XSLTVM object.

Definition

```
typedef enum xmlxvmobjtype {
    XMLXVM_TYPE_UNKNOWN ,
    XMLXVM_TYPE_NDSET   ,
    XMLXVM_TYPE_BOOL    ,
    XMLXVM_TYPE_NUM     ,
    XMLXVM_TYPE_STR     ,
    XMLXVM_TYPE_FRAG
} xmlxvmobjtype;
```

xpctx

XPath top-level context.

Definition

```
struct xpctx;
typedef struct xpctx xpctx;
```

xpexpr

XPath expression.

Definition

```
struct xpexpr;
typedef struct xpexpr xpexpr;
```

xpobj

Xpath object.

Definition

```
struct xpobj;typedef struct xpobj xpobj;
```

xsdctx

XML Schema validator context, created by `XmlSchemaCreate` and passed to most Schema functions.

Definition

```
# define XSDCTX_DEFINED
struct xsdctx; typedef struct xsdctx xsdctx;
```

xslctx

XSL top-level context.

Definition

```
struct xslctx;
typedef struct xslctx xslctx;
```

xvmobj

XSLVM processor run-time object; content is private and must not be accessed by users.

Definition

```
struct xvmobj;
typedef struct xvmobj xvmobj;
```

JsonDomDoc

JSON DOM tree container. It is the first argument to all the JSON DOM functions.

JsonDomNode

Opaque JSON DOM node abstract type.

Definition

```
typedef struct JsonDomNode JsonDomNode;
```

JsonDomObject

Opaque JSON DOM object type.

Definition

```
typedef struct JsonDomObject JsonDomObject;
```

JsonDomArray

Opaque JSON DOM array type.

Definition

```
typedef struct JsonDomArray JsonDomArray;
```

JsonDomScalar

Opaque JSON DOM scalar type.

Definition

```
typedef struct JsonDomScalar JsonDomScalar;
```

jznnodetype

Enum type to indicate the type of DOM node.

Definition

```
typedef enum
{
  JZNDOM_SCALAR  = 1,
  JZNDOM_OBJECT  = 2,
  JZNDOM_ARRAY   = 3
} jznnodetype;
```

jznvaltype

Enum type to indicate the type of the scalar value.

Definition

```
typedef enum jznvaltype
{
```

```
  JZNVAL_NULL               = 2
  JZNVAL_STRING             = 3
  JZNVAL_FALSE              = 5
  JZNVAL_TRUE               = 6
  JZNVAL_FLOAT              = 11,
  JZNVAL_DOUBLE             = 12,
  JZNVAL_BINARY             = 13,
  JZNVAL_ORA_NUMBER         = 17,
  JZNVAL_ORA_DATE           = 18,
  JZNVAL_ORA_TIMESTAMP      = 19,
  JZNVAL_ORA_TIMESTAMP      = 20,
  JZNVAL_ORA_YEARMONTH_DUR  = 21,
  JZNVAL_ORA_DAYSECOND_DUR  = 22
} jznvaltype;
```

jznScalarVal

DOM Scalar Value Output Structure.

The scalar values can be extracted from the scalar nodes using the function `JsonDomGetScalarInfoOci()`.

Scalar Type	C Program Variable	Value Pointer	Value Length
String	oratext[n]	val_jznScalarVal	len_jznScalarVal
Binary	ub1[n]	binval_jznScalarVal	binlen_jznScalarVal
Double	double	db_jznScalarVal	NA
Float	float	flt_jznScalarVal	NA
True	boolean	NA	NA
False	boolean	NA	NA

Scalar Type	C Program Variable	Value pointer
Number	ub1[JZN_ORA_NUM_MAX_LEN] or OCINumber*	num_JsonOCIVal
Date	JsonDateTime	dt_JsonOCIVal
Timestamp	JsonDateTime	dt_JsonOCIVal
Day-second Interval	JsonDayInterval	dayInv_JsonOCIVal
Year-month Interval	JsonYearInterval	yrInv_JsonOCIVal

JsonOCIVal

Union of helper output structured (OCINumber, JsonDateTime, JsonDayInterval and JsonYearInterval) forms.

Definition

```
typedef union JsonOCIVal
{
  JsonDateTime      dt_JsonOCIVal;
  JsonDayInterval   dayInv_JsonOCIVal;
  JsonYearInterval  yrInv_JsonOCIVal;
  ub1               num_JsonOCIVal[JZN_ORA_NUM_MAX_LEN]; /* Equivalent to
```

```
OCINumber[22] */
} JsonOCIVal;
```

JsonDateTime

Structure for date and timestamp scalar output values.

Definition

```
typedef struct JsonDateTime
{
  sb2  year_JsonDateTime;
  ub1  month_JsonDateTime;
  ub1  day_JsonDateTime;
  ub1  hour_JsonDateTime;
  ub1  minute_JsonDateTime;
  ub1  second_JsonDateTime;
  ub4  fsecond_JsonDateTime;
  sb1  tzHourOffset_JsonDateTime;
  sb1  tzMinuteOffset_JsonDateTime;
} JsonDateTime;
```

JsonDayInterval

Structure for day-second interval scalar output values.

Definition

```
typedef struct JsonDayInterval
{
  sb4  days_JsonDayInterval;
  sb4  hours_JsonDayInterval;
  sb4  minutes_JsonDayInterval;
  sb4  seconds_JsonDayInterval;
  sb4  fseconds_JsonDayInterval;
} JsonDayInterval;
```

JsonYearInterval

Structure for year-month interval scalar output values.

Definition

```
typedef struct JsonYearInterval
{
  sb4  years_JsonYearInterval;
  sb4  months_JsonYearInterval;
} JsonYearInterval;
```

jsonerr

A parameter to handle the DOM API error that returns an error code.

Error code: `JZNERR_OK (0)`

- A value of `(0)` indicates success.

- Any value other than `(0)` indicates failure.

2
Package Callback for XML C APIs

The Callback package defines macros that declare functions (or function pointers) for XML callbacks. Callbacks are used for error-message handling, memory allocation and freeing, and stream operations.

The following table summarizes the methods available through the Callback interface.

Table 2-1 Summary of Callback Methods for XML C Implementation

Function	Summary
XML_ACCESS_CLOSE_F()	User-defined access method close callback.
XML_ACCESS_OPEN_F()	User-defined access method open callback.
XML_ACCESS_READ_F()	User-defined access method read callback.
XML_ALLOC_F()	Low-level memory allocation.
XML_ERRMSG_F()	Handles error message.
XML_FREE_F()	Low-level memory freeing.
XML_STREAM_CLOSE_F()	User-defined stream close callback.
XML_STREAM_OPEN_F()	User-defined stream open callback.
XML_STREAM_READ_F()	User-defined stream read callback.
XML_STREAM_WRITE_F()	User-defined stream write callback.

XML_ACCESS_CLOSE_F()

This macro defines a prototype for the close function callback used to access a URL.

XML_ACCESS_OPEN_F()

This macro defines a prototype for the open function callback used to access a URL.

XML_ACCESS_READ_F()

This macro defines a prototype for the read function callback used to access a URL.

XML_ALLOC_F()

This macro defines a prototype for the low-level memory `alloc` function provided by the user. If no allocator is provided, `malloc` is used. Memory should not be zeroed by this function. Matches XML_FREE_F().

XML_ERRMSG_F()

This macro defines a prototype for the error message handling function. If no error message callback is provided at XML initialization time, errors will be printed to `stderr`. If a handler is provided, it will be invoked instead of printing to stderr.

XML_FREE_F()

This macro defines a prototype for the low-level memory free function provided by the user. If no allocator is provided, free() is used. Matches XML_ALLOC_F().

XML_STREAM_CLOSE_F()

This macro defines a prototype for the close function callback, called to close an open source and free its resources.

XML_STREAM_OPEN_F()

This macro defines a prototype for the open function callback, which is called once to open the input source. This function should return `XMLERR_OK` on success.

XML_STREAM_READ_F()

This macro defines a prototype for the read function callback, called to read data from an open source into a buffer, returning the number of bytes read (< `0` on error). The `eoi` flag determines if this is the final block of data.

On EOI, the close function will be called automatically.

Syntax

```
#define XML_STREAM_READ_F(func, xctx, sctx, path, dest, size, nraw, eoi)
xmlerr func(
   xmlctx *xctx,
   void *sctx,
   oratext *path,
   oratext *dest,
   size_t size,
   sbig_ora *nraw,
   boolean *eoi);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`sctx`	`IN`	user-defined stream context
`path`	`IN`	full URI of the open source (for error messages)

Parameter	In/Out	Description
dest	(OUT)	destination buffer to read data into
size	IN	size of destination buffer
nraw	(OUT)	number of bytes read
eoi	(OUT)	signal to end of information; last chunk

Returns

(xmlerr) numeric error code, 0 on success

> **See Also:**
>
> XML_STREAM_OPEN_F(), XML_STREAM_CLOSE_F(), XML_STREAM_WRITE_F()

XML_STREAM_WRITE_F()

This macro defines a prototype for the write function callback, called to write data to a user-defined stream.

Syntax

```
#define XML_STREAM_WRITE_F(func, xctx, sctx, path, src, size)
xmlerr func(
   xmlctx *xctx,
   void *sctx,
   oratext *path,
   oratext *src,
   size_t size);
```

Parameter	In/Out	Description
xctx	IN	XML context
sctx	IN	user-defined stream context
path	IN	full URI of the open source (for error messages)
src	IN	source buffer to read data from
size	IN	size of source in bytes

Returns

`(xmlerr)` numeric error code, `0` on success

> **See Also:**
>
> XML_STREAM_OPEN_F(), XML_STREAM_CLOSE_F(), XML_STREAM_READ_F()

3
Package DOM for XML C APIs

This implementation follows REC-DOM-Level-1-19981001. Because the DOM standard is object-oriented, some changes were made for C language adaptation.

- Reused function names have to be expanded; `getValue` in the `Attr` interface has the unique name `XmlDomGetAttrValue` that matches the pattern established by DOM 2's `getNodeValue`.
- Functions were added to extend the DOM beyond the standard; one example is `XmlDomNumChildNodes`, which returns the number of children of a node.

Note that if the `data_encoding` parameter is set to UTF-16, the APIs process wide-`CHAR` arrays, not `oratext` byte arrays.

Attr Interface for DOM XML C APIs

The following table summarizes the methods available through the `Attr` interface of DOM for XML C APIs.

Table 3-1 Summary of Attr DOM Methods for XML C Implementation

Function	Summary
XmlDomGetAttrLocal()	Returns an attribute's namespace local name as `NULL`-terminated string.
XmlDomGetAttrLocalLen()	Returns an attribute's namespace local name as length-encoded string.
XmlDomGetAttrName()	Return attribute's name as `NULL`-terminated string.
XmlDomGetAttrNameLen()	Return attribute's name as length-encoded string.
XmlDomGetAttrPrefix()	Returns an attribute's namespace prefix.
XmlDomGetAttrSpecified()	Return flag that indicates whether an attribute was explicitly created.
XmlDomGetAttrURI()	Returns an attribute's namespace URI as `NULL`-terminated string.
XmlDomGetAttrURILen()	Returns an attribute's namespace URI as length-encoded string.
XmlDomGetAttrValue()	Return attribute's value as `NULL`-terminated string.
XmlDomGetAttrValueLen()	Return attribute's value as length-encoded string.
XmlDomGetAttrValueStream()	Get attribute value stream-style,i.e.chunked.
XmlDomGetOwnerElem()	Return an attribute's "owning" element.
XmlDomSetAttrValue()	Set an attribute's value.
XmlDomSetAttrValueStream()	Sets an attribute value stream style (chunked).

XmlDomGetAttrLocal()

Returns an attribute's namespace local name (in the data encoding). If the attribute's name is not fully qualified (has no prefix), then the local name is the same as the name.

A length-encoded version is available as `XmlDomGetAttrURILen` which returns the local name as a pointer and length, for use if the data is known to use `XMLType` backing store.

Syntax

```
oratext* XmlDomGetAttrLocal(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`attr`	`IN`	attribute node

Returns

`(oratext *)` attribute's local name [data encoding]

> **See Also:**
>
> XmlDomGetAttrLocalLen(), XmlDomGetAttrName(), XmlDomGetAttrURI(), XmlDomGetAttrPrefix()

XmlDomGetAttrLocalLen()

Returns an attribute's namespace local name (in the data encoding). If the attribute's name is not fully qualified (has no prefix), then the local name is the same as the name.

A `NULL`-terminated version is available as `XmlDomGetAttrLocal` which returns the local name as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the attribute's data will be stored internally as length-encoded. Using the length-based `Get`*`XXX`* functions will avoid having to copy and `NULL`-terminate the data.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than `buflen`, then a truncated value will be copied into the buffer and len will return the actual length.

Syntax

```
oratext* XmlDomGetAttrLocalLen(
   xmlctx *xctx,
   xmlattrnode *attr,
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node
buf	IN	input buffer; optional
buflen	IN	input buffer length; optional
len	OUT	length of local name, in characters

Returns

`(oratext *) Attr`'s local name [data encoding]

> **See Also:**
>
> XmlDomGetAttrLocal(), XmlDomGetAttrName(), XmlDomGetAttrURI(), XmlDomGetAttrPrefix()

XmlDomGetAttrName()

Returns the fully-qualified name of an attribute (in the data encoding) as a `NULL`-terminated string, for example `bar\0` or `foo:bar\0`.

A length-encoded version is available as `XmlDomGetAttrNameLen` which returns the attribute name as a pointer and length, for use if the data is known to use `XMLType` backing store.

Syntax

```
oratext* XmlDomGetAttrName(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
attr	IN	attribute node

Returns

(oratext *) name of attribute [data encoding]

> **See Also:**
>
> XmlDomGetAttrNameLen(), XmlDomGetAttrURI(), XmlDomGetAttrPrefix(), XmlDomGetAttrLocal()

XmlDomGetAttrNameLen()

Returns the fully-qualified name of an attribute (in the data encoding) as a length-encoded string, for example ("bar", 3) or ("foo:bar", 7).

A NULL-terminated version is available as XmlDomGetAttrName which returns the attribute name as NULL-terminated string. If the backing store is known to be XMLType, then the attribute's data will be stored internally as length-encoded. Using the length-based Get*XXX*() functions will avoid having to copy and NULL-terminate the data.

If both the input buffer is non-NULL and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than buflen, then a truncated value will be copied into the buffer and len will return the actual length.

Syntax

```
oratext* XmlDomGetAttrNameLen(
   xmlctx *xctx,
   xmlattrnode *attr,
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node
buf	IN	input buffer; optional
buflen	IN	input buffer length; optional

Parameter	In/Out	Description
len	OUT	length of local name, in characters

Returns

(oratext *) name of attribute [data encoding]

> **See Also:**
>
> XmlDomGetAttrName(), XmlDomGetAttrURI(), XmlDomGetAttrPrefix(), XmlDomGetAttrLocal()

XmlDomGetAttrPrefix()

Returns an attribute's namespace prefix (in the data encoding). If the attribute's name is not fully qualified (has no prefix), NULL is returned.

Syntax

```
oratext* XmlDomGetAttrPrefix(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node

Returns

(oratext *) attribute's namespace prefix [data encoding] or NULL

> **See Also:**
>
> XmlDomGetAttrName(), XmlDomGetAttrURI(), XmlDomGetAttrLocal()

XmlDomGetAttrSpecified()

Return the 'specified' flag for an attribute. If the attribute was explicitly given a value in the original document, this is TRUE; otherwise, it is FALSE. If the node is not an attribute, returns FALSE. If the user sets an attribute's value through DOM, its specified flag will be TRUE. To return an attribute to its default value (if it has one), the attribute should be deleted; it will then be re-created automatically with the default value (and specified will be FALSE).

Syntax

```
boolean XmlDomGetAttrSpecified(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node

Returns

(boolean) attribute's "specified" flag

> **See Also:**
>
> XmlDomSetAttrValue()

XmlDomGetAttrURI()

Returns an attribute's namespace URI (in the data encoding). If the attribute's name is not qualified (does not contain a namespace prefix), it will have the default namespace in effect when the node was created (which may be NULL).

A length-encoded version is available as XmlDomGetAttrURILen which returns the URI as a pointer and length, for use if the data is known to use XMLType backing store.

Syntax

```
oratext* XmlDomGetAttrURI(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node

Returns

(oratext *) attribute's namespace URI [data encoding] or NULL

> **See Also:**
>
> XmlDomGetAttrURILen(), XmlDomGetAttrPrefix(), XmlDomGetAttrLocal()

XmlDomGetAttrURILen()

Returns an attribute's namespace URI (in the data encoding) as length-encoded string. If the attribute's name is not qualified (does not contain a namespace prefix), it will have the default namespace in effect when the node was created (which may be `NULL`).

A `NULL`-terminated version is available as `XmlDomGetAttrURI` which returns the URI as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the attribute's data will be stored internally as length-encoded. Using the length-based Get functions will avoid having to copy and `NULL`-terminate the data.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than buflen, then a truncated value will be copied into the buffer and len will return the actual length.

Syntax

```
oratext* XmlDomGetAttrURILen(
   xmlctx *xctx,
   xmlattrnode *attr,
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`attr`	`IN`	attribute node
`buf`	`IN`	input buffer; optional
`buflen`	`IN`	input buffer length; optional
`len`	`OUT`	length of URI, in characters

Returns

`(oratext *)` attribute's namespace URI [data encoding] or `NULL`

> **See Also:**
>
> XmlDomGetAttrURI(), XmlDomGetAttrPrefix(), XmlDomGetAttrLocal()

XmlDomGetAttrValue()

Returns the "value" (character data) of an attribute (in the data encoding) as `NULL`-terminated string. Character and general entities will have been replaced.

A length-encoded version is available as `XmlDomGetAttrValueLen` which returns the attribute value as a pointer and length, for use if the data is known to use `XMLType` backing store.

Syntax

```
oratext* XmlDomGetAttrValue(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`attr`	`IN`	attribute node

Returns

`(oratext *)` attribute's value

> **See Also:**
>
> XmlDomGetAttrValueLen(), XmlDomSetAttrValue()

XmlDomGetAttrValueLen()

Returns the "value" (character data) of an attribute (in the data encoding) as length-encoded string. Character and general entities will have been replaced.

A `NULL`-terminated version is available as `XmlDomGetAttrValue` which returns the attribute value as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the attribute's data will be stored internally as length-encoded. Using the length-based `Get`*`XXX`*() functions will avoid having to copy and `NULL`-terminate the data.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than buflen, then a truncated value will be copied into the buffer and len will return the actual length.

Syntax

```
oratext* XmlDomGetAttrValueLen(
   xmlctx *xctx,
   xmlattrnode *attr,
```

```
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node
buf	IN	input buffer; optional
buflen	IN	input buffer length; optional
len	OUT	length of attribute's value, in characters

Returns

(oratext *) attribute's value

> **See Also:**
>
> XmlDomGetAttrValue(), XmlDomSetAttrValue()

XmlDomGetAttrValueStream()

Returns the large "value" (associated character data) for an attribute and sends it in pieces to the user's output stream. For very large values, it is not always possible to store them [efficiently] as a single contiguous chunk. This function is used to access chunked data of that type.

Syntax

```
xmlerr XmlDomGetAttrValueStream(
   xmlctx *xctx,
   xmlnode *attr,
   xmlostream *ostream);
```

Parameter	In/Out	Description
xctx	IN	XML context
attr	IN	attribute node
ostream	IN	output stream object

Returns

(xmlerr) numeric error code, 0 on success

XmlDomGetOwnerElem()

Returns the `Element` node associated with an attribute. Each attr either belongs to an element (one and only one), or is detached and not yet part of the DOM tree. In the former case, the element node is returned; if the attr is unassigned, `NULL` is returned.

Syntax

```
xmlelemnode* XmlDomGetOwnerElem(
   xmlctx *xctx,
   xmlattrnode *attr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`attr`	`IN`	attribute node

Returns

`(xmlelemnode *)` attribute's element node [or `NULL`]

> **See Also:**
>
> XmlDomGetOwnerDocument()

XmlDomSetAttrValue()

Sets the given attribute's value to data. If the node is not an attribute, does nothing. Note that the new value must be in the data encoding! It is not verified, converted, or checked. The attribute's specified flag will be `TRUE` after setting a new value.

Syntax

```
void XmlDomSetAttrValue(
   xmlctx *xctx,
   xmlattrnode *attr,
   oratext *value);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`attr`	`IN`	attribute node
`value`	`IN`	new value of attribute; data encoding

> **See Also:**
>
> XmlDomGetAttrValue()

XmlDomSetAttrValueStream()

Sets the large "value" (associated character data) for an attribute piecemeal from an input stream. For very large values, it is not always possible to store them efficiently as a single contiguous chunk. This function is used to access chunked data of that type.

Syntax

```
xmlerr XmlDomSetAttrValueStream(
   xmlctx *xctx,
   xmlnode *attr,
   xmlistream *istream);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`attr`	`IN`	attribute node
`isream`	`IN`	input stream

Returns

`(xmlerr)` numeric error code, `0` on success

CharacterData Interface for DOM XML C APIs

The following table summarizes the methods available through the `CharacterData` interface of DOM for XML C APIs.

Table 3-2 Summary of CharacterData DOM Methods for XML C Implementation

Function	Summary
XmlDomAppendData()	Append data to end of node's current data.
XmlDomDeleteData()	Remove part of node's data.
XmlDomGetCharData()	Return data for node.
XmlDomGetCharDataLength()	Return length of data for node.
XmlDomInsertData()	Insert string into node's current data.
XmlDomReplaceData()	Replace part of node's data.
XmlDomSetCharData()	Set data for node.
XmlDomSubstringData()	Return substring of node's data.

XmlDomAppendData()

Append a string to the end of a `CharacterData` node's data. If the node is not `Text`, `Comment` or `CDATA`, or if the string to append is `NULL`, does nothing. The appended data should be in the data encoding. It will not be verified, converted, or checked.

The new node data will be allocated and managed by DOM, but if the previous node value was allocated and manager by the user, they are responsible for freeing it, which is why it is returned.

Syntax

```
void XmlDomAppendData(
   xmlctx *xctx,
   xmlnode *node,
   oratext *data,
   oratext **old);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	`CharacterData` node
`data`	`IN`	data to append; data encoding
`old`	`OUT`	previous data for node; data encoding

See Also:

XmlDomGetCharData(), XmlDomInsertData(), XmlDomDeleteData(), XmlDomReplaceData(), XmlDomSplitText()

XmlDomDeleteData()

Remove a range of characters from a `CharacterData` node's data. If the node is not text, comment or `CDATA`, or if the offset is outside of the original data, does nothing. The `offset` is zero-based, so offset zero refers to the start of the data. Both `offset` and `count` are in characters, not bytes. If the sum of offset and count exceeds the data length then all characters from `offset` to the end of the data are deleted.

The new node data will be allocated and managed by DOM, but if the previous node value was allocated and managed by the user, they are responsible for freeing it, which is why it is returned.

Syntax

```
void XmlDomDeleteData(
   xmlctx *xctx,
   xmlnode *node,
```

```
   ub4 offset,
   ub4 count,
   oratext **old);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	CharacterData node
offset	IN	character offset where to start removing
count	IN	number of characters to delete
old	OUT	previous data for node; data encoding

See Also:

XmlDomGetCharData(), XmlDomAppendData(), XmlDomInsertData(), XmlDomReplaceData(), XmlDomSplitText()

XmlDomGetCharData()

Returns the data for a CharacterData node (type text, comment or CDATA) in the data encoding. For other node types, or if there is no data, returns NULL.

Syntax

```
oratext* XmlDomGetCharData(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	CharacterData node; Text, Comment or CDATA

Returns

(oratext *) character data of node [data encoding]

See Also:

XmlDomSetCharData(), XmlDomCreateText(), XmlDomCreateComment(), XmlDomCreateCDATA()

XmlDomGetCharDataLength()

Returns the length of the data for a `CharacterData` node, type `Text`, `Comment` or `CDATA`) in characters, not bytes. For other node types, returns 0.

Syntax

```
ub4 XmlDomGetCharDataLength(
   xmlctx *xctx,
   xmlnode *cdata);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	`CharacterData` node; `Text`, `Comment` or `CDATA`

Returns

`(ub4)` length in characters, not bytes, of node's data

> **See Also:**
>
> XmlDomGetCharData()

XmlDomInsertData()

Insert a string into a `CharacterData` node's data at the specified position. If the node is not `Text`, `Comment` or `CDATA`, or if the data to be inserted is `NULL`, or the offset is outside the original data, does nothing. The inserted data must be in the data encoding. It will not be verified, converted, or checked. The offset is specified as characters, not bytes. The offset is zero-based, so inserting at offset zero prepends the data.

The new node data will be allocated and managed by DOM, but if the previous node value was allocated and managed by the user, they are responsible for freeing it (which is why it's returned).

Syntax

```
void XmlDomInsertData(
   xmlctx *xctx,
   xmlnode *node,
   ub4 offset,
   oratext *arg,
   oratext **old);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
node	IN	CharacterData node; Text, Comment, or CDATA
offset	IN	character offset where to start inserting
arg	IN	data to insert
old	OUT	previous data for node; data encoding

See Also:

XmlDomGetCharData(), XmlDomAppendData(), XmlDomDeleteData(), XmlDomReplaceData(), XmlDomSplitText()

XmlDomReplaceData()

Replaces a range of characters in a CharacterData node's data with a new string. If the node is not text, comment or CDATA, or if the offset is outside of the original data, or if the replacement string is NULL, does nothing. If the count is zero, acts just as XmlDomInsertData. The offset is zero-based, so offset zero refers to the start of the data. The replacement data must be in the data encoding. It will not be verified, converted, or checked. The offset and count are both in characters, not bytes. If the sum of offset and count exceeds length, then all characters to the end of the data are replaced.

The new node data will be allocated and managed by DOM, but if the previous node value was allocated and managed by the user, they are responsible for freeing it, which is why it is returned.

Syntax

```
void XmlDomReplaceData(
   xmlctx *xctx,
   xmlnode *node,
   ub4 offset,
   ub4 count,
   oratext *arg,
   oratext **old);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	CharacterData node; Text, Comment, or CDATA
offset	IN	character offset where to start replacing

Parameter	In/Out	Description
count	IN	number of characters to replace
arg	IN	replacement substring; data encoding
old	OUT	previous data for node; data encoding

See Also:

XmlDomGetCharData(), XmlDomAppendData(), XmlDomInsertData(), XmlDomDeleteData(), XmlDomSplitText()

XmlDomSetCharData()

Sets data for a CharacterData node (type text, comment or CDATA), replacing the old data. For other node types, does nothing. The new data is not verified, converted, or checked; it should be in the data encoding.

Syntax

```
void XmlDomSetCharData(
   xmlctx *xctx,
   xmlnode *node,
   oratext *data);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	CharacterData node; Text, Comment, or CDATA
data	IN	new data for node

See Also:

XmlDomGetCharData()

XmlDomSubstringData()

Returns a range of character data from a CharacterData node, type Text, Comment or CDATA. For other node types, or if count is zero, returns NULL. Since the data is in the data encoding, offset and count are in characters, not bytes. The beginning of the

string is offset `0`. If the sum of offset and count exceeds the length, then all characters to the end of the data are returned.

The substring is permanently allocated in the node's document's memory pool. To free the substring, use `XmlDomFreeString`.

Syntax

```
oratext* XmlDomSubstringData(
   xmlctx *xctx,
   xmlnode *node,
   ub4 offset,
   ub4 count);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	`CharacterData` node; `Text`, `Comment`, or `CDATA`
`offset`	`IN`	character offset where to start extraction of substring
`count`	`IN`	number of characters to extract

Returns

`(oratext *)` specified substring.

> **See Also:**
>
> XmlDomAppendData(), XmlDomInsertData(), XmlDomDeleteData(), XmlDomReplaceData(), XmlDomSplitText(), XmlDomFreeString()

Document Interface for DOM XML C APIs

The following table summarizes the methods available through the `Document` interface of DOM for XML C APIs.

Table 3-3 Summary of Document DOM Methods for XML C Implementation

Function	Summary
XmlDomCreateAttr()	Create attribute node.
XmlDomCreateAttrNS()	Create attribute node with namespace information.
XmlDomCreateCDATA()	Create `CDATA` node.
XmlDomCreateComment()	Create comment node.
XmlDomCreateElem()	Create an element node.
XmlDomCreateElemNS()	Create an element node with namespace information.

Table 3-3 (Cont.) Summary of Document DOM Methods for XML C Implementation

Function	Summary
XmlDomCreateEntityRef()	Create entity reference node.
XmlDomCreateFragment()	Create a document fragment.
XmlDomCreatePI()	Create PI node.
XmlDomCreateText()	Create text node.
XmlDomFreeString()	Frees a string allocate by `XmlDomSubstringData`, and others.
XmlDomGetBaseURI()	Returns the base URI for a document.
XmlDomGetDTD()	Get DTD for document.
XmlDomGetDecl()	Returns a document's `XMLDecl` information.
XmlDomGetDocElem()	Get top-level element for document.
XmlDomGetDocElemByID()	Get document element given ID.
XmlDomGetDocElemsByTag()	Obtain document elements.
XmlDomGetDocElemsByTagNS()	Obtain document elements (namespace aware version).
XmlDomGetLastError()	Return last error code for document.
XmlDomGetSchema()	Returns URI of schema associated with document.
XmlDomImportNode()	Import a node from another DOM.
XmlDomIsSchemaBased()	Indicate whether a schema is associated with a document.
XmlDomSaveString()	Saves a string permanently in a document's memory pool.
XmlDomSaveString2()	Saves a Unicode string permanently in a document's memory pool.
XmlDomSetDTD()	Sets DTD for document.
XmlDomSetDocOrder()	Set document order for all nodes.
XmlDomSetLastError()	Sets last error code for document.
XmlDomSync()	Synchronizes the persistent version of a document with its DOM.

XmlDomCreateAttr()

Creates an attribute node with the given name and value (in the data encoding). Note this function differs from the DOM specification, which does not allow the initial value of the attribute to be set (see `XmlDomSetAttrValue`). The name is required, but the value may be `NULL`; neither is verified, converted, or checked.

This is the non-namespace aware function (see `XmlDomCreateAttrNS`): the new attribute will have `NULL` namespace URI and prefix, and its local name will be the same as its name, even if the name specified is a qualified name.

If given an initial value, the attribute's specified flag will be `TRUE`.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild`, and so on.

See `XmlDomSetAttr` which creates and adds an attribute in a single operation.

The name and value are not copied, their pointers are just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlattrnode* XmlDomCreateAttr(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *name,
   oratext *value);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
name	IN	new node's name; data encoding; user control
value	IN	new node's value; data encoding; user control

Returns

`(xmlattrnode *)` new `Attr` node.

> **See Also:**
>
> XmlDomSetAttrValue(), XmlDomCreateAttrNS(), XmlDomSetAttr(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateAttrNS()

Creates an attribute node with the given namespace URI and qualified name; this is the namespace-aware version of `XmlDomCreateAttr`. Note this function differs from the DOM specification, which does not allow the initial value of the attribute to be set (see `XmlDomSetAttrValue`). The name is required, but the value may be `NULL`; neither is verified, converted, or checked.

If given an initial value, the attribute's specified flag will be `TRUE`.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild`, and so on. See `XmlDomSetAttr` which creates and adds an attribute in a single operation.

The URI, qualified name and value are not copied, their pointers are just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlattrnode* XmlDomCreateAttrNS(
   xmlctx *xctx,
   xmldocnode *doc,
```

```
   oratext *uri,
   oratext *qname,
   oratext *value);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
uri	IN	node's namespace URI; data encoding; user control
qname	IN	node's qualified name; data encoding; user control
value	IN	new node's value; data encoding; user control

Returns

`(xmlattrnode *)` new `Attr` node.

> **See Also:**
>
> XmlDomSetAttrValue(), XmlDomCreateAttr(), XmlDomSetAttr(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateCDATA()

Creates a `CDATASection` node with the given initial data (which should be in the data encoding). A `CDATASection` is considered verbatim and is never parsed; it will not be joined with adjacent `Text` nodes by the normalize operation. The initial data may be `NULL`; if provided, it is not verified, converted, or checked. The name of a `CDATA` node is always "`#cdata-section`".

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild` and so on.

The `CDATA` is not copied, its pointer is just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlcdatanode* XmlDomCreateCDATA(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *data);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
doc	IN	XML document node
data	IN	new node's CDATA; data encoding; user control

Returns

(xmlcdatanode *) new CDATA node.

> **See Also:**
>
> XmlDomCreateText(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateComment()

Creates a Comment node with the given initial data (which must be in the data encoding). The data may be NULL; if provided, it is not verified, converted, or checked. The name of a Comment node is always "#comment".

The new node is an orphan with no parent; it must be added to the DOM tree with XmlDomAppendChild and so on.

The comment data is not copied, its pointer is just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlcommentnode* XmlDomCreateComment(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *data);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
data	IN	new node's comment; data encoding; user control

Returns

(xmlcommentnode *) new Comment node.

> **See Also:**
>
> XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateElem()

Creates an element node with the given tag name (which should be in the data encoding). Note that the tag name of an element is case sensitive. This is the non-namespace aware function: the new node will have `NULL` namespace URI and prefix, and its local name will be the same as its tag name, even if the tag name specified is a qualified name.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild` and so on.

The `tagname` is not copied, its pointer is just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlelemnode* XmlDomCreateElem(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *tagname);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`tagname`	`IN`	new node's name; data encoding; user control

Returns

`(xmlelemnode *)` new `Element` node.

> **See Also:**
>
> XmlDomCreateElemNS(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateElemNS()

Creates an element with the given namespace URI and qualified name. Note that element names are case sensitive, and the qualified name is required though the URI may be `NULL`. The qualified name will be split into prefix and local parts, retrievable with `XmlDomGetNodePrefix`, `XmlDomGetNodeLocal`, and so on; the tagName will be the full qualified name.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild` and so on.

The URI and qualified name are not copied, their pointers are just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlelemnode* XmlDomCreateElemNS(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *uri,
   oratext *qname);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`uri`	`IN`	new node's namespace URI; data encoding, user control
`qname`	`IN`	new node's qualified name; data encoding; user control

Returns

`(xmlelemnode *)` new `Element` node.

> **See Also:**
>
> XmlDomCreateElem(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateEntityRef()

Creates an `EntityReference` node; the name (which should be in the data encoding) is the name of the entity to be referenced. The named entity does not have to exist. The name is not verified, converted, or checked.

`EntityReference` nodes are never generated by the parser; instead, entity references are expanded as encountered. On output, an entity reference node will turn into a "&name;" style reference.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild`, and so on.

The entity reference name is not copied, its pointer is just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlentrefnode* XmlDomCreateEntityRef(
   xmlctx *xctx,
```

```
   xmldocnode *doc,
   oratext *name);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
name	IN	name of referenced entity; data encoding; user control

Returns

`(xmlentrefnode *)` new `EntityReference` node.

XmlDomCreateFragment()

Creates an empty `DocumentFragment` node. A document fragment is treated specially when it is inserted into a DOM tree: the children of the fragment are inserted in order instead of the fragment node itself. After insertion, the fragment node will still exist, but have no children. See `XmlDomInsertBefore`, `XmlDomReplaceChild`, `XmlDomAppendChild`, and so on. The name of a fragment node is always "`#document-fragment`".

Syntax

```
xmlfragnode* XmlDomCreateFragment(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node

Returns

`(xmlfragnode *)` new empty `DocumentFragment` node

> **See Also:**
>
> XmlDomInsertBefore(), XmlDomReplaceChild(), XmlDomAppendChild()

XmlDomCreatePI()

Creates a `ProcessingInstruction` node with the given target and data (which should be in the data encoding). The data may be `NULL` initially, and may be changed later (with `XmlDomSetPIData`), but the target is required and cannot be changed. Note the

target and data are not verified, converted, or checked. The name of a PI node is the same as the target.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild` and so on.

The PI's target and data are not copied, their pointers are just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlpinode* XmlDomCreatePI(
   xmlctx *xctx
   xmldocnode *doc,
   oratext *target,
   oratext *data);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`target`	`IN`	new node's target; data encoding; user control
`data`	`IN`	new node's data; data encoding; user control

Returns

`(xmlpinode *)` new PI node.

> **See Also:**
>
> XmlDomGetPITarget(), XmlDomGetPIData(), XmlDomSetPIData(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomCreateText()

Creates a `Text` node with the given initial data (which must be non-`NULL` and in the data encoding). The data may be `NULL`; if provided, it is not verified, converted, checked, or parsed (entities will not be expanded). The name of a fragment node is always "`#text`". New data for a `Text` node can be set; see the `CharacterData` interface for editing methods.

The new node is an orphan with no parent; it must be added to the DOM tree with `XmlDomAppendChild` and so on.

The text data is not copied, its pointer is just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmltextnode* XmlDomCreateText(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *data);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
data	IN	new node's text; data encoding; user control

Returns

`(xmltextnode *)` new `Text` node.

> **See Also:**
>
> XmlDomCreateCDATA(), XmlDomSetNodeValue(), XmlDomGetNodeValue(), XmlDomSetCharData(), XmlDomGetCharData(), XmlDomGetCharDataLength(), XmlDomSubstringData(), XmlDomAppendData(), XmlDomInsertData(), XmlDomDeleteData(), XmlDomReplaceData(), XmlDomCleanNode(), XmlDomFreeNode()

XmlDomFreeString()

Frees the string allocated by `XmlDomSubstringData` or similar functions. Note that strings explicitly saved with `XmlDomSaveString` are not freeable individually.

Syntax

```
void XmlDomFreeString(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *str);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	document where the string belongs
str	IN	string to free

> **See Also:**
>
> XmlDomSaveString(), XmlDomSaveString2()

XmlDomGetBaseURI()

Returns the base URI for a document. Usually only documents that were loaded from a URI will automatically have a base URI; documents loaded from other sources (`stdin`, buffer, and so on) will not naturally have a base URI, but a base URI may have been set for them using `XmlDomSetBaseURI`, for the purposes of resolving relative URIs in inclusion.

Syntax

```
oratext *XmlDomGetBaseURI(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Returns

`(oratext *)` document's base URI [or `NULL`]

> **See Also:**
>
> XmlDomSetBaseURI()

XmlDomGetDTD()

Returns the DTD node associated with current document; if there is no DTD, returns `NULL`. The DTD cannot be edited, but its children may be retrieved with `XmlDomGetChildNodes` as for other node types.

Syntax

```
xmldtdnode* XmlDomGetDTD(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
doc	IN	XML document node

Returns

(xmldtdnode *) DTD node for document [or NULL]

> **See Also:**
>
> XmlDomSetDTD(), XmlCreateDTD() and XmlCreate() in Package XML APIs for C, XmlDomGetDTDName(), XmlDomGetDTDEntities(), and XmlDomGetDTDNotations()

XmlDomGetDecl()

Returns the information from a document's XMLDecl. If there is no XMLDecl, returns XMLERR_NO_DECL. Returned are the XML version# ("1.0" or "2.0"), the specified encoding, and the standalone value. If encoding is not specified, NULL will be set. The standalone flag is three-state: < 0 if standalone was not specified, 0 if it was specified and FALSE, > 0 if it was specified and TRUE.

Syntax

```
xmlerr XmlDomGetDecl(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext **ver,
   oratext **enc,
   sb4 *std);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
ver	OUT	XML version
enc	OUT	encoding specification
std	OUT	standalone specification

Returns

(xmlerr) XML error code, perhaps version/encoding/standalone set

XmlDomGetDocElem()

Returns the root element (node) of the DOM tree, or `NULL` if there is none. Each document has only one uppermost `Element` node, called the root element. It is created after a document is parsed successfully, or manually by `XmlDomCreateElem` then `XmlDomAppendChild`, and so on.

Syntax

```
xmlelemnode* XmlDomGetDocElem(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Returns

`(xmlelemnode *)` root element [or `NULL`]

> **See Also:**
>
> XmlDomCreateElem()

XmlDomGetDocElemByID()

Returns the element node which has the given ID. If no such ID is defined, returns `NULL`. Note that attributes named "ID" are not automatically of type ID; ID attributes (which can have any name) must be declared as type ID in the DTD.

The given ID should be in the data encoding or it might not match.

Syntax

```
xmlelemnode* XmlDomGetDocElemByID(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *id);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Parameter	In/Out	Description
id	IN	element's unique ID; data encoding

Returns

(xmlelemnode *) matching element.

> **See Also:**
>
> XmlDomGetDocElemsByTag(), XmlDomGetDocElemsByTagNS()

XmlDomGetDocElemsByTag()

Returns a list of all elements in the document tree rooted at the root node with a given tag name, in document order (the order in which they would be encountered in a preorder traversal of the tree). If root is NULL, the entire document is searched.

The special name "*" matches all tag names; a NULL name matches nothing. Note that tag names are case sensitive, and should be in the data encoding or a mismatch might occur.

This function is not namespace aware; the full tag names are compared. If two qualified names with two different prefixes both of which map to the same URI are compared, the comparison will fail. See XmlDomGetElemsByTagNS for the namespace-aware version.

The list should be freed with XmlDomFreeNodeList when it is no longer needed.

The list is not live, it is a snapshot. That is, if a new node which matched the tag name were added to the DOM after the list was returned, the list would not automatically be updated to include the node.

Syntax

```
xmlnodelist* XmlDomGetDocElemsByTag(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *name);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
name	IN	tagname to match; data encoding; * for all

Returns

`(xmlnodelist *)` new `NodeList` containing all matched `Element`s.

> **See Also:**
>
> XmlDomGetDocElemByID(), XmlDomGetDocElemsByTagNS(), XmlDomFreeNodeList()

XmlDomGetDocElemsByTagNS()

Returns a list of all elements (in the document tree rooted at the given node) with a given namespace URI and local name, in the order in which they would be encountered in a preorder traversal of the tree. If root is `NULL`, the entire document is searched.

The URI and local name should be in the data encoding. The special local name "*" matches all local names; a `NULL` local name matches nothing. Namespace URIs must always match, however, no wildcard is allowed. Note that comparisons are case sensitive. See `XmlDomGetDocElemsByTag` for the non-namespace aware version.

The list should be freed with `XmlDomFreeNodeList` when it is no longer needed.

The list is not live, it is a snapshot. That is, if a new node which matched the tag name were added to the DOM after the list was returned, the list would not automatically be updated to include the node.

Syntax

```
xmlnodelist* XmlDomGetDocElemsByTagNS(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`uri`	`IN`	namespace URI to match; data encoding; `*` matches all
`local`	`IN`	local name to match; data encoding; `*` matches all

Returns

`(xmlnodelist *)` new `NodeList` containing all matched `Element`s.

> **See Also:**
>
> XmlDomGetDocElemByID(), XmlDomGetDocElemsByTag(), XmlDomFreeNodeList()

XmlDomGetLastError()

Returns the error code of the last error which occurred in the given document.

Syntax

```
xmlerr XmlDomGetLastError(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Returns

`(xmlerr)` numeric error code, `0` if no error

XmlDomGetSchema()

Returns URI of schema associated with document, if there is one, else returns `NULL`. The `XmlLoadDom` functions take a schema location hint (URI); the schema is used for efficient layout of `XMLType` data. If a schema was provided at load time, this function returns `TRUE`.

Syntax

```
oratext* XmlDomGetSchema(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Returns

`(oratext *)` Schema URI or `NULL`

> **See Also:**
>
> XmlDomIsSchemaBased(), XmlLoadDom() in Package XML APIs for C

XmlDomImportNode()

Imports a node from one `Document` to another. The new node is an orphan and has no parent; it must be added to the DOM tree with `XmlDomAppendChild`, and so on. The original node is not modified in any way or removed from its document; instead, a new node is created with copies of all the original node's qualified name, prefix, namespace URI, and local name.

As with `XmlDomCloneNode`, the deep controls whether the children of the node are recursively imported. If `FALSE`, only the node itself is imported, and it will have no children. If `TRUE`, all descendents of the node will be imported as well, and an entire new subtree created.

`Document` and `DocumentType` nodes cannot be imported. Imported attributes will have their specified flags set to `TRUE`. Elements will have only their specified attributes imported; non-specified (default) attributes are omitted. New default attributes (for the destination document) are then added.

Syntax

```
xmlnode* XmlDomImportNode(
   xmlctx *xctx,
   xmldocnode *doc,
   xmlctx *nctx,
   xmlnode *node,
   boolean deep);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`nctx`	`IN`	XML context of imported node
`node`	`IN`	node to import
`deep`	`IN`	`TRUE` to import the subtree recursively

Returns

`(xmlnode *)` newly imported node (in this `Document`).

> **See Also:**
>
> XmlDomCloneNode()

XmlDomIsSchemaBased()

Returns flag specifying whether there is a schema associated with this document. The `XmlLoadDom` functions take a schema location hint (URI); the schema is used for efficient layout of `XMLType` data. If a schema was provided at load time, this function returns `TRUE`.

Syntax

```
boolean XmlDomIsSchemaBased(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Returns

`(boolean) TRUE` if there is a schema associated with the document

> **See Also:**
>
> XmlDomGetSchema(), XmlLoadDom() in Package XML APIs for C

XmlDomSaveString()

Copies the given string into the document's memory pool, so that it persists for the life of the document. The individual string will not be freeable, and the storage will be returned only when the entire document is freed. Works on single-byte or multibyte encodings; for Unicode strings, use `XmlDomSaveString2`.

Syntax

```
oratext* XmlDomSaveString(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *str);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`str`	`IN`	string to save; data encoding; single- or multi-byte only

Returns

`(oratext *)` saved copy of string

> **See Also:**
>
> XmlDomSaveString2(), XmlFreeDocument() in Package XML APIs for C

XmlDomSaveString2()

Copies the given string into the document's memory pool, so that it persists for the life of the document. The individual string will not be freeable, and the storage will be returned only when the entire document is free. Works on Unicode strings only; for single-byte or multibyte strings, use `XmlDomSaveString`.

Syntax

```
ub2* XmlDomSaveString2(
   xmlctx *xctx,
   xmldocnode *doc,
   ub2 *ustr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node
`ustr`	`IN`	string to save; data encoding; Unicode only

Returns

`(ub2 *)` saved copy of string

> **See Also:**
>
> XmlDomSaveString(), XmlFreeDocument() in Package XML APIs for C

XmlDomSetBaseURI()

Only documents that were loaded from a URI will automatically have a base URI; documents loaded from other sources (stdin, buffer, and so on) will not naturally have a base URI, so this API is used to set a base URI, for the purposes of relative URI resolution in includes. The base URI should be in the data encoding, and a copy will be made.

Syntax

```
xmlerr XmlDomSetBaseURI(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *uri);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
uri	IN	base URI to set; data encoding

Returns

(xmlerr) XML error code

> **See Also:**
>
> XmlDomGetBaseURI()

XmlDomSetDTD()

Sets the DTD for document. Note this call may only be used for a blank document, before any parsing has taken place. A single DTD can be set for multiple documents, so when a document with a set DTD is freed, the set DTD is not also freed.

Syntax

```
xmlerr XmlDomSetDTD(
   xmlctx *xctx,
   xmldocnode *doc,
   xmldtdnode *dtdnode);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
dtdnode	IN	DocumentType node to set

Returns

(xmlerr) numeric error code, 0 on success

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDName(), XmlDomGetDTDEntities(), XmlDomGetDTDNotations()

XmlDomSetDocOrder()

Sets the document order for each node in the current document. Must be called once on the final document before XSLT processing can occur. Note this is called automatically by the XSLT processor, so ordinarily the user need not make this call.

Syntax

```
ub4 XmlDomSetDocOrder(
   xmlctx *xctx,
   xmldocnode *doc,
   ub4 start_id);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
start_id	IN	string ID number

Returns

(ub4) highest ordinal assigned

XmlDomSetLastError()

Sets the Last Error code for the given document. If doc is NULL, sets the error code for the XML context.

Syntax

```
xmlerr XmlDomSetLastError(
   xmlctx *xctx,
   xmldocnode *doc,
   xmlerr errcode);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	XML document node
errcode	IN	error code to set, 0 to clear error

Returns

`(xmlerr)` original error code

XmlDomSync()

Causes a modified DOM to be written back out to its original source, synchronizing the persistent store and in-memory versions.

Syntax

```
xmlerr XmlDomSync(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`doc`	`IN`	XML document node

Returns

`(xmlerr)` numeric error code, `0` on success

DocumentType Interface for DOM XML C APIs

The following table summarizes the methods available through the `DocumentType` interface of DOM for XML C APIs.

Table 3-4 Summary of DocumentType DOM Methods for XML C Implementation

Function	Summary
XmlDomGetDTDEntities()	Get entities of DTD.
XmlDomGetDTDInternalSubset()	Get DTD's internal subset.
XmlDomGetDTDName()	Get name of DTD.
XmlDomGetDTDNotations()	Get notations of DTD.
XmlDomGetDTDPubID()	Get DTD's public ID.
XmlDomGetDTDSysID()	Get DTD's system ID.

XmlDomGetDTDEntities()

Returns a named node map of general entities defined by the DTD. If the node is not a DTD, or has no general entities, returns `NULL`.

Syntax

```
xmlnamedmap* XmlDomGetDTDEntities(
   xmlctx *xctx,
   xmldtdnode *dtd);
```

Parameter	In/Out	Description
xctx	IN	XML context
dtd	IN	DTD node

Returns

(xmlnamedmap *) named node map containing entities declared in DTD

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDName(), XmlDomGetDTDNotations(), XmlDomGetDTDSysID(), XmlDomGetDTDInternalSubset()

XmlDomGetDTDInternalSubset()

Returns the content model for an element. If there is no DTD, returns NULL.

Syntax

```
xmlnode* XmlDomGetDTDInternalSubset(
   xmlctx *xctx,
   xmldtdnode *dtd,
   oratext *name);
```

Parameter	In/Out	Description
xctx	IN	XML context
dtd	IN	DTD node
name	IN	name of Element; data encoding

Returns

(xmlnode *) content model subtree

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDName(), XmlDomGetDTDEntities(), XmlDomGetDTDNotations(), XmlDomGetDTDPubID()

XmlDomGetDTDName()

Returns a DTD's name (specified immediately after the `DOCTYPE` keyword), or `NULL` if the node is not type DTD.

Syntax

```
oratext* XmlDomGetDTDName(
   xmlctx *xctx,
   xmldtdnode *dtd);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`dtd`	`IN`	DTD node

Returns

`(oratext *)` name of DTD

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDEntities(), XmlDomGetDTDNotations(), XmlDomGetDTDSysID(), XmlDomGetDTDInternalSubset()

XmlDomGetDTDNotations()

Returns named node map of notations declared by the DTD. If the node is not a DTD or has no `Notation`s, returns `NULL`.

Syntax

```
xmlnamedmap* XmlDomGetDTDNotations(
   xmlctx *xctx,
   xmldtdnode *dtd);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`dtd`	`IN`	DTD node

Returns

`(xmlnamedmap *)` named node map containing notations declared in DTD

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDName(), XmlDomGetDTDEntities(), XmlDomGetDTDSysID(), XmlDomGetDTDInternalSubset()

XmlDomGetDTDPubID()

Returns a DTD's public identifier.

Syntax

```
oratext* XmlDomGetDTDPubID(
   xmlctx *xctx,
   xmldtdnode *dtd);
```

Parameter	In/Out	Description
xctx	IN	XML context
dtd	IN	DTD node

Returns

`(oratext *)` DTD's public identifier [data encoding]

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDName(), XmlDomGetDTDEntities(), XmlDomGetDTDSysID(), XmlDomGetDTDInternalSubset()

XmlDomGetDTDSysID()

Returns a DTD's system identifier.

Syntax

```
oratext* XmlDomGetDTDSysID(
   xmlctx *xctx,
   xmldtdnode *dtd);
```

Parameter	In/Out	Description
xctx	IN	XML context
dtd	IN	DTD node

Returns

`(oratext *)` DTD's system identifier [data encoding]

> **See Also:**
>
> XmlDomGetDTD(), XmlDomGetDTDName(), XmlDomGetDTDEntities(), XmlDomGetDTDPubID(), XmlDomGetDTDInternalSubset()

Element Interface for DOM XML C APIs

The following table summarizes the methods available through the `Element` Interface of DOM for XML C APIs.

Table 3-5 Summary of Element DOM Methods for XML C Implementation

Function	Summary
XmlDomGetAttr()	Return attribute's value given its name.
XmlDomGetAttrNS()	Return attribute's value given its URI and local name.
XmlDomGetAttrNode()	Get attribute by name.
XmlDomGetAttrNodeNS()	Get attribute by name (namespace aware version).
XmlDomGetChildrenByTag()	Get children of element with given tag name (non-namespace aware).
XmlDomGetChildrenByTagNS()	Get children of element with tag name (namespace aware version).
XmlDomGetDocElemsByTag()	Obtain doc elements.
XmlDomGetDocElemsByTagNS()	Obtain doc elements (namespace aware version).
XmlDomGetTag()	Return an element node's tag name.
XmlDomHasAttr()	Does named attribute exist?
XmlDomHasAttrNS()	Does named attribute exist (namespace aware version)?
XmlDomRemoveAttr()	Remove attribute with specified name.
XmlDomRemoveAttrNS()	Remove attribute with specified URI and local name.
XmlDomRemoveAttrNode()	Remove attribute node.
XmlDomSetAttr()	Set new attribute for element.
XmlDomSetAttrNS()	Set new attribute for element (namespace aware version).
XmlDomSetAttrNode()	Set attribute node.
XmlDomSetAttrNodeNS()	Set attribute node (namespace aware version).

XmlDomGetAttr()

Returns the value of an element's attribute (specified by name). Note that an attribute may have the empty string as its value, but cannot be `NULL`. If the element does not have an attribute with the given name, `NULL` is returned.

Syntax

```
oratext* XmlDomGetAttr(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	element node
`name`	`IN`	attribute's name

Returns

`(oratext *)` named attribute's value [data encoding; may be `NULL`]

> **See Also:**
>
> XmlDomGetAttrNS(), XmlDomGetAttrs(), XmlDomGetAttrNode()

XmlDomGetAttrNS()

Returns the value of an element's attribute (specified by URI and local name). Note that an attribute may have the empty string as its value, but cannot be `NULL`. If the element does not have an attribute with the given name, `NULL` is returned.

Syntax

```
oratext* XmlDomGetAttrNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	element node

Parameter	In/Out	Description
uri	IN	attribute's namespace URI; data encoding
local	IN	attribute's local name; data encoding

Returns

(oratext *) named attribute's value [data encoding; may be NULL]

> **See Also:**
>
> XmlDomGetAttr(), XmlDomGetAttrs(), XmlDomGetAttrNode()

XmlDomGetAttrNode()

Returns an element's attribute specified by name. If the node is not an element or the named attribute does not exist, returns NULL.

Syntax

```
xmlattrnode* XmlDomGetAttrNode(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
name	IN	attribute's name; data encoding

Returns

(xmlattrnode *) attribute with the specified name [or NULL]

> **See Also:**
>
> XmlDomGetAttrNodeNS(), XmlDomGetAttr()

XmlDomGetAttrNodeNS()

Returns an element's attribute specified by URI and localname. If the node is not an element or the named attribute does not exist, returns `NULL`.

Syntax

```
xmlattrnode* XmlDomGetAttrNodeNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	element node
`uri`	`IN`	attribute's namespace URI; data encoding
`local`	`IN`	attribute's local name; data encoding

Returns

`(xmlattrnode *)` attribute node with the given URI/local name [or `NULL`]

> **See Also:**
>
> XmlDomGetAttrNode(), XmlDomGetAttr()

XmlDomGetChildrenByTag()

Returns a list of children of an element with the given tag name, in the order in which they would be encountered in a preorder traversal of the tree. The tag name should be in the data encoding. The special name "`*`" matches all tag names; a `NULL` name matches nothing. Note that tag names are case sensitive. This function is not namespace aware; the full tag names are compared. If two prefixes which map to the same URI are compared, the comparison will fail. See `XmlDomGetChildrenByTagNS` for the namespace-aware version. The returned list can be freed with `XmlDomFreeNodeList`.

Syntax

```
xmlnodelist* XmlDomGetChildrenByTag(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
name	IN	tag name to match; data encoding; * for all

Returns

(xmlnodelist *) node list of matching children

> **See Also:**
>
> XmlDomGetChildrenByTagNS(), XmlDomFreeNodeList()

XmlDomGetChildrenByTagNS()

Returns a list of children of an element with the given URI and local name, in the order in which they would be encountered in a preorder traversal of the tree. The URI and local name should be in the data encoding. The special name "*" matches all URIs or tag names; a NULL name matches nothing. Note that names are case sensitive. See XmlDomGetChildrenByTag for the non-namespace version. The returned list can be freed with XmlDomFreeNodeList.

Syntax

```
xmlnodelist* XmlDomGetChildrenByTagNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
uri	IN	namespace URI to match; data encoding; * matches all
local	IN	local name to match; data encoding; * matches all

Returns

(xmlnodelist *) node list of matching children

> **See Also:**
>
> XmlDomGetChildrenByTag(), XmlDomFreeNodeList()

XmlDomGetElemsByTag()

Returns a list of all elements (in the document tree rooted at the root node) with a given tag name, in the order in which they would be encountered in a preorder traversal of the tree. If root is `NULL`, the entire document is searched. The tag name should be in the data encoding. The special name "`*`" matches all tag names; a `NULL` name matches nothing. Note that tag names are case sensitive. This function is not namespace aware; the full tag names are compared. If two prefixes which map to the same URI are compared, the comparison will fail. See `XmlDomGetElemsByTagNS` for the namespace-aware version. The returned list can be freed with `XmlDomFreeNodeList`.

Syntax

```
xmlnodelist* XmlDomGetElemsByTag(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	element node
`name`	`IN`	tag name to match; data encoding; `*` for all

Returns

`(xmlnodelist *)` node list of matching elements

> **See Also:**
>
> XmlDomGetElemsByTagNS(), XmlDomFreeNodeList()

XmlDomGetElemsByTagNS()

Returns a list of all elements (in the document tree rooted at the root node) with a given URI and localname, in the order in which they would be encountered in a preorder traversal of the tree. If root is `NULL`, the entire document is searched. The tag name should be in the data encoding. The special name "`*`" matches all tag names; a `NULL` name matches nothing. Note that tag names are case sensitive. This function is not namespace aware; the full tag names are compared. If two prefixes which map to the same URI are compared, the comparison will fail. The returned list can be freed with `XmlDomFreeNodeList`.

Syntax

```
xmlnodelist* XmlDomGetElemsByTagNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
uri	IN	namespace URI to match; data encoding; * for all
local	IN	local name to match; data encoding; * for all

Returns

(xmlnodelist *) node list of matching elements

> **See Also:**
>
> XmlDomGetDocElemsByTag(), XmlDomFreeNodeList()

XmlDomGetTag()

Returns the tagName of a node, which is the same as its name. DOM 1.0 states "...even though there is a generic nodeName attribute on the Node interface, there is still a tagName attribute on the Element interface; these two attributes must contain the same value, but the Working Group considers it worthwhile to support both, given the different constituencies the DOM API must satisfy."

Syntax

```
oratext* XmlDomGetTag(
   xmlctx *xctx,
   xmlelemnode *elem);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	Element node

Returns

(oratext *) element's name [data encoding]

> **See Also:**
>
> XmlDomGetNodeName()

XmlDomHasAttr()

Determines if an element has an attribute with the given name. Returns `TRUE` if so, `FALSE` if not.

Syntax

```
boolean XmlDomHasAttr(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	`Element` node
`name`	`IN`	attribute's name; data encoding

Returns

`(boolean) TRUE` if element has attribute with given name

> **See Also:**
>
> XmlDomHasAttrNS()

XmlDomHasAttrNS()

Determines if an element has an attribute with the given URI and localname. Returns `TRUE` if so, `FALSE` if not.

Syntax

```
boolean XmlDomHasAttrNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	Element node
uri	IN	attribute's namespace URI; data encoding
local	IN	attribute's local name; data encoding

Returns

(boolean) TRUE if element has attribute with given URI/localname

> **See Also:**
>
> XmlDomHasAttr()

XmlDomRemoveAttr()

Removes an attribute (specified by name). If the removed attribute has a default value it is immediately re-created with that default. Note that the attribute is removed from the element's list of attributes, but the attribute node itself is not destroyed.

Syntax

```
void XmlDomRemoveAttr(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
name	IN	attribute's name; data encoding

> **See Also:**
>
> XmlDomRemoveAttrNS(), XmlDomRemoveAttrNode()

XmlDomRemoveAttrNS()

Removes an attribute (specified by URI and local name). If the removed attribute has a default value it is immediately re-created with that default. Note that the attribute is removed from the element's list of attributes, but the attribute node itself is not destroyed.

Syntax

```
void XmlDomRemoveAttrNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
uri	IN	attribute's namespace URI
local	IN	attribute's local name

> **See Also:**
>
> XmlDomRemoveAttr(), XmlDomRemoveAttrNode()

XmlDomRemoveAttrNode()

Removes an attribute from an element. If the attribute has a default value, it is immediately re-created with that value (Specified set to `FALSE`). Returns the removed attribute on success, else `NULL`.

Syntax

```
xmlattrnode* XmlDomRemoveAttrNode(
   xmlctx *xctx,
   xmlelemnode *elem,
   xmlattrnode *oldAttr);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node

Parameter	In/Out	Description
oldAtrr	IN	attribute node to remove

Returns

(xmlattrnode *) replaced attribute node [or NULL]

> **See Also:**
>
> XmlDomRemoveAttr()

XmlDomSetAttr()

Creates a new attribute for an element with the given name and value (which should be in the data encoding). If the named attribute already exists, its value is simply replaced. The name and value are not verified, converted, or checked. The value is not parsed, so entity references will not be expanded. The attribute's specified flag will be set.

Syntax

```
void XmlDomSetAttr(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *name,
   oratext *value);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
name	IN	attribute's name; data encoding
value	IN	attribute's value; data encoding

> **See Also:**
>
> XmlDomSetAttrNS(), XmlDomCreateAttr(), XmlDomSetAttrValue(), XmlDomRemoveAttr()

XmlDomSetAttrNS()

Creates a new attribute for an element with the given URI, localname and value (which should be in the data encoding). If the named attribute already exists, its value is simply replaced. The name and value are not verified, converted, or checked.

The value is not parsed, so entity references will not be expanded.

The attribute's specified flag will be set.

Syntax

```
void XmlDomSetAttrNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   oratext *uri,
   oratext *qname,
   oratext *value);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	element node
`uri`	`IN`	attribute's namespace URI; data encoding
`qname`	`IN`	attribute's qualified name; data encoding
`value`	`IN`	attribute's value; data encoding

> **See Also:**
>
> XmlDomSetAttr(), XmlDomCreateAttr(), XmlDomSetAttrValue(), XmlDomRemoveAttr()

XmlDomSetAttrNode()

Adds a new attribute to an element. If an attribute with the given name already exists, it is replaced and the old attribute returned through `oldNode`. If the attribute is new, it is added to the element's list and `oldNode` set to `NULL`.

Syntax

```
xmlattrnode* XmlDomSetAttrNode(
   xmlctx *xctx,
   xmlelemnode *elem,
   xmlattrnode *newAttr);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
newAttr	IN	attribute node to add

Returns

(xmlattrnode *) replaced attribute node (or NULL)

> **See Also:**
>
> XmlDomSetAttrNodeNS(), XmlDomCreateAttr(), XmlDomSetAttrValue()

XmlDomSetAttrNodeNS()

Adds a new attribute to an element. If an attribute with newNode's URI and localname already exists, it is replaced and the old attribute returned through oldNode. If the attribute is new, it is added to the element's list and oldNode set to NULL.

Syntax

```
xmlattrnode* XmlDomSetAttrNodeNS(
   xmlctx *xctx,
   xmlelemnode *elem,
   xmlattrnode *newAttr);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	element node
newAttr	IN	attribute node to add

Returns

(xmlattrnode *) replaced attribute node [or NULL]

> **See Also:**
>
> XmlDomSetAttrNode(), XmlDomCreateAttr(), XmlDomSetAttrValue()

Entity Interface for DOM XML C APIs

The following table summarizes the methods available through the `Entity` interface of DOM for XML C APIs.

Table 3-6 Summary of Entity DOM Methods for XML C Implementation

Function	Summary
XmlDomGetEntityNotation()	Get entity's notation.
XmlDomGetEntityPubID()	Get entity's public ID.
XmlDomGetEntitySysID()	Get entity's system ID.
XmlDomGetEntityType()	Get entity's type.

XmlDomGetEntityNotation()

For unparsed entities, returns the name of its notation (in the data encoding). For parsed entities and other node types, returns `NULL`.

Syntax

```
oratext* XmlDomGetEntityNotation(
   xmlctx *xctx,
   xmlentnode *ent);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`ent`	`IN`	entity node

Returns

`(oratext *)` entity's notation [data encoding; may be `NULL`]

> **See Also:**
>
> XmlDomGetEntityPubID(), XmlDomGetEntitySysID()

XmlDomGetEntityPubID()

Returns an entity's public identifier (in the data encoding). If the node is not an entity, or has no defined public ID, returns `NULL`.

Syntax

```
oratext* XmlDomGetEntityPubID(
   xmlctx *xctx,
   xmlentnode *ent);
```

Parameter	In/Out	Description
xctx	IN	XML context
ent	IN	entity node

Returns

(oratext *) entity's public identifier [data encoding; may be NULL]

> **See Also:**
>
> XmlDomGetEntitySysID(), XmlDomGetEntityNotation()

XmlDomGetEntitySysID()

Returns an entity's system identifier (in the data encoding). If the node is not an entity, or has no defined system ID, returns NULL.

Syntax

```
oratext* XmlDomGetEntitySysID(
   xmlctx *xctx,
   xmlentnode *ent);
```

Parameter	In/Out	Description
xctx	IN	XML context
ent	IN	entity node

Returns

(oratext *) entity's system identifier [data encoding; may be NULL]

> **See Also:**
>
> XmlDomGetEntityPubID(), XmlDomGetEntityNotation()

XmlDomGetEntityType()

Returns a boolean for an entity describing whether it is general (`TRUE`) or parameter (`FALSE`).

Syntax

```
boolean XmlDomGetEntityType(
   xmlctx *xctx,
   xmlentnode *ent);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`ent`	`IN`	entity node

Returns

`(boolean) TRUE` for general entity, `FALSE` for parameter entity

> **See Also:**
>
> XmlDomGetEntityPubID(), XmlDomGetEntitySysID(), XmlDomGetEntityNotation()

NamedNodeMap Interface for DOM XML C APIs

The following table summarizes the methods available through the `NamedNodeMap` interface of DOM for XML C APIs.

Table 3-7 Summary of NamedNodeMap DOM Methods for XML C Implementation

Function	Summary
XmlDomGetNamedItem()	Return named node from list.
XmlDomGetNamedItemNS()	Return named node from list (namespace aware version).
XmlDomGetNodeMapItem()	Return n^{th} node in list.
XmlDomGetNodeMapLength()	Return length of named node map.
XmlDomRemoveNamedItem()	Remove node from named node map.
XmlDomRemoveNamedItemNS()	Remove node from named node map (namespace aware version).
XmlDomSetNamedItem()	Set node in named node list.
XmlDomSetNamedItemNS()	Set node in named node list (namespace aware version).

XmlDomGetNamedItem()

Retrieves an item from a `NamedNodeMap`, specified by name (which should be in the data encoding). This is a non-namespace-aware function; it just matches (case sensitively) on the whole qualified name. Note this function differs from the DOM spec in that the index of the matching item is also returned.

Syntax

```
xmlnode* XmlDomGetNamedItem(
   xmlctx *xctx,
   xmlnamedmap *map,
   oratext *name);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`map`	`IN`	`NamedNodeMap`
`name`	`IN`	name of the node to retrieve

Returns

`(xmlnode *) Node` with the specified name [or `NULL`]

> **See Also:**
>
> XmlDomGetNamedItemNS(), XmlDomGetNodeMapItem(), XmlDomGetNodeMapLength()

XmlDomGetNamedItemNS()

Retrieves an item from a `NamedNodeMap`, specified by URI and localname (which should be in the data encoding). Note this function differs from the DOM spec in that the index of the matching item is also returned.

Syntax

```
xmlnode* XmlDomGetNamedItemNS(
   xmlctx *xctx,
   xmlnamedmap *map,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
map	IN	NamedNodeMap
uri	IN	namespace URI of the node to retrieve; data encoding
local	IN	local name of the node to retrieve; data encoding

Returns

(xmlnode *) node with given local name and namespace URI [or NULL]

> **See Also:**
>
> XmlDomGetNamedItem(), XmlDomGetNodeMapItem(), XmlDomGetNodeMapLength()

XmlDomGetNodeMapItem()

Retrieves an item from a NamedNodeMap, specified by name (which should be in the data encoding). This is a non-namespace-aware function; it just matches (case sensitively) on the whole qualified name. Note this function differs from the DOM specification in that the index of the matching item is also returned. Named "item" in W3C specification.

Syntax

```
xmlnode* XmlDomGetNodeMapItem(
   xmlctx *xctx,
   xmlnamedmap *map,
   ub4 index);
```

Parameter	In/Out	Description
xctx	IN	XML context
map	IN	NamedNodeMap
index	IN	0-based index for the map

Returns

(xmlnode *) node at the nth position in the map (or NULL)

> **See Also:**
>
> XmlDomGetNamedItem(), XmlDomSetNamedItem(), XmlDomRemoveNamedItem(), XmlDomGetNodeMapLength()

XmlDomGetNodeMapLength()

Returns the number of nodes in a `NamedNodeMap` (the length). Note that nodes are referred to by index, and the range of valid indexes is `0` through length-1.

Syntax

```
ub4 XmlDomGetNodeMapLength(
   xmlctx *xctx,
   xmlnamedmap *map);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`map`	`IN`	`NamedNodeMap`

Returns

`(ub4)` number of nodes in `NamedNodeMap`

> **See Also:**
>
> XmlDomGetNodeMapItem(), XmlDomGetNamedItem()

XmlDomRemoveNamedItem()

Removes a node from a `NamedNodeMap`, specified by name. This is a non-namespace-aware function; it just matches (case sensitively) on the whole qualified name. If the removed node is an attribute with default value (not specified), it is immediately replaced. The removed node is returned; if no removal took place, `NULL` is returned.

Syntax

```
xmlnode* XmlDomRemoveNamedItem(
   xmlctx *xctx,
   xmlnamedmap *map,
   oratext *name);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
map	IN	NamedNodeMap
name	IN	name of node to remove

Returns

(xmlnode *) node removed from this map

> **See Also:**
>
> XmlDomRemoveNamedItemNS(), XmlDomGetNamedItem(), XmlDomGetNamedItemNS(), XmlDomSetNamedItem(), XmlDomSetNamedItemNS()

XmlDomRemoveNamedItemNS()

Removes a node from a NamedNodeMap, specified by URI and localname. If the removed node is an attribute with default value (not specified), it is immediately replaced. The removed node is returned; if no removal took place, NULL is returned.

Syntax

```
xmlnode* XmlDomRemoveNamedItemNS(
   xmlctx *xctx,
   xmlnamedmap *map,
   oratext *uri,
   oratext *local);
```

Parameter	In/Out	Description
xctx	IN	XML context
map	IN	NamedNodeMap
uri	IN	namespace URI of the node to remove; data encoding
local	IN	local name of the node to remove; data encoding

Returns

(xmlnode *) node removed from this map

> **See Also:**
>
> XmlDomRemoveNamedItem(), XmlDomGetNamedItem(), XmlDomGetNamedItemNS(), XmlDomSetNamedItem(), XmlDomSetNamedItemNS()

XmlDomSetNamedItem()

Adds a new node to a `NamedNodeMap`. If a node already exists with the given name, replaces the old node and returns it. If no such named node exists, adds the new node to the map and sets old to `NULL`. This is a non-namespace-aware function; it just matches (case sensitively) on the whole qualified name. Since some node types have fixed names (`Text`, `Comment`, and so on), trying to set another of the same type will always cause replacement.

Syntax

```
xmlnode* XmlDomSetNamedItem(
   xmlctx *xctx,
   xmlnamedmap *map,
   xmlnode *newNode);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`map`	`IN`	`NamedNodeMap`
`newNode`	`IN`	new node to store in map

Returns

`(xmlnode *)` the replaced node (or `NULL`)

> **See Also:**
>
> XmlDomSetNamedItemNS(), XmlDomGetNamedItem(), XmlDomGetNamedItemNS(), XmlDomGetNodeMapItem(), XmlDomGetNodeMapLength()

XmlDomSetNamedItemNS()

Adds a new node to a `NamedNodeMap`. If a node already exists with the given URI and localname, replaces the old node and returns it. If no such named node exists, adds the new node to the map and sets old to `NULL`. Since some node types have fixed names (`Text`, `Comment`, and so on), trying to set another of the same type will always cause replacement.

Syntax

```
xmlnode* XmlDomSetNamedItemNS(
   xmlctx *xctx,
   xmlnamedmap *map,
   xmlnode *newNode);
```

Parameter	In/Out	Description
xctx	IN	XML context
map	IN	NamedNodeMap
newNode	IN	new node to store in map

Returns

(xmlnode *) replaced Node [or NULL]

> **See Also:**
>
> XmlDomSetNamedItem(), XmlDomGetNamedItem(), XmlDomGetNamedItemNS(), XmlDomGetNodeMapItem(), XmlDomGetNodeMapLength()

Node Interface for DOM XML C APIs

The following table summarizes the methods available through the Node interface of DOM for XML C APIs.

Table 3-8 Summary of Node DOM Methods for XML C Implementation

Function	Summary
XmlDomAppendChild()	Append new child to node's list of children.
XmlDomCleanNode()	Clean a node (free DOM allocations).
XmlDomCloneNode()	Clone a node.
XmlDomFreeNode()	Free a node allocated with XmlDomCreateXXX.
XmlDomGetAttrs()	Return attributes of node.
XmlDomGetChildNodes()	Return children of node.
XmlDomGetDefaultNS()	Get default namespace for node.
XmlDomGetFirstChild()	Returns first child of node.
XmlDomGetFirstPfnsPair()	Get first prefix namespace pair.
XmlDomGetLastChild()	Returns last child of node.
XmlDomGetNextPfnsPair()	Get subsequent prefix namespace pair.

Table 3-8 (Cont.) Summary of Node DOM Methods for XML C Implementation

Function	Summary
XmlDomGetNextSibling()	Return next sibling of node.
XmlDomGetNodeLocal()	Get local part of node's qualified name as `NULL`-terminated string.
XmlDomGetNodeLocalLen()	Get local part of node's qualified name as length-encoded string.
XmlDomGetNodeName()	Get node's name as `NULL`-terminated string.
XmlDomGetNodeNameLen()	Get node's name as length-encoded string.
XmlDomGetNodePrefix()	Return namespace prefix of node.
XmlDomGetNodeType()	Get node's numeric type code.
XmlDomGetNodeURI()	Return namespace URI of node as a `NULL`-terminated string.
XmlDomGetNodeURILen()	Return namespace URI of node as length-encoded string.
XmlDomGetNodeValue()	Get node's value as `NULL`-terminated string.
XmlDomGetNodeValueLen()	Get node value as length-encoded string.
XmlDomGetNodeValueStream()	Returns the large data for a node and sends it in pieces to the user's output stream.
XmlDomGetOwnerDocument()	Get the owner document of node.
XmlDomGetParentNode()	Get parent node.
XmlDomGetPrevSibling()	Return previous sibling of node.
XmlDomGetPullNodeAsBinaryStream()	Returns the address of a binary stream using the pull paradigm.
XmlDomGetPullNodeAsCharacterStream()	Returns the address of a character stream using the pull paradigm.
XmlDomGetPushNodeAsBinaryStream()	Returns the address of a binary stream, as an `OUT ostream` parameter, using the push paradigm.
XmlDomGetPushNodeAsCharacterStream()	Returns the address of a character stream, as an `OUT ostream` parameter, using the push paradigm.
XmlDomGetSourceEntity()	Return the entity node if the input file is an external entity.
XmlDomGetSourceLine()	Return source line number of node.
XmlDomGetSourceLocation()	Return source location (path, URI, and so on) of node.
XmlDomHasAttr()	Does named attribute exist?
XmlDomHasChildNodes()	Test if node has children.
XmlDomInsertBefore()	Insert new child in to node's list of children.
XmlDomNormalize()	Normalize a node by merging adjacent text nodes.
XmlDomNumAttrs()	Return number of attributes of element.
XmlDomNumChildNodes()	Return number of children of node.

Table 3-8 (Cont.) Summary of Node DOM Methods for XML C Implementation

Function	Summary
XmlDomPrefixToURI()	Get namespace URI for prefix.
XmlDomRemoveChild()	Remove an existing child node.
XmlDomRenameNode()	Updates the name of a node, for element and attribute nodes only.
XmlDomRenameNodeNS()	Updates the name and URI of a node, for element and attribute nodes only.
XmlDomReplaceChild()	Replace an existing child of a node.
XmlDomSetDefaultNS()	Set default namespace for node.
XmlDomSetNodePrefix()	Set namespace prefix of node.
XmlDomSetNodeValue()	Set node value.
XmlDomSetNodeValueLen()	Set node value as length-encoded string.
XmlDomSetNodeValueStream()	Sets the large "value" (character data) for a node piecemeal from an input stream.
XmlDomSetPullNodeAsBinaryStream()	Returns the address of a binary input stream, as an `OUT istream` parameter, using the pull paradigm.
XmlDomSetPullNodeAsCharacterStream()	Returns the address of an input character stream, as an `OUT istream` parameter, using the pull paradigm.
XmlDomSetPushNodeAsBinaryStream()	Returns the address of an input binary stream using the push paradigm.
XmlDomSetPushNodeAsCharacterStream()	Returns the address of a character stream using the push paradigm.
XmlDomValidate()	Validate a node against current DTD.

XmlDomAppendChild()

Appends the node to the end of the parent's list of children and returns the new node. If `newChild` is a `DocumentFragment`, all of its children are appended in original order; the `DocumentFragment` node itself is not.

Syntax

```
xmlnode* XmlDomAppendChild(
   xmlctx *xctx,
   xmlnode *parent,
   xmlnode *newChild);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`parent`	`IN`	parent to receive a new node

Parameter	In/Out	Description
newChild	IN	node to add

Returns

(xmlnode *) node added

> **See Also:**
>
> XmlDomInsertBefore(), XmlDomReplaceChild()

XmlDomCleanNode()

Frees parts of the node which were allocated by DOM itself, but does not recurse to children or touch the node's attributes. After freeing part of the node (such as name), a DOM call to get that part (such as XmlDomGetNodeName) should return a NULL pointer. Used to manage the allocations of a node parts of which are controlled by DOM, and part by the user. Calling clean frees all allocations may by DOM and leaves the user's allocations alone. The user is responsible for freeing their own allocations.

Syntax

```
void XmlDomCleanNode(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	node to clean

> **See Also:**
>
> XmlDomFreeNode()

XmlDomCloneNode()

Creates and returns a duplicate of a node. The duplicate node has no parent. Cloning an element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but it does not copy any text it contains unless it is a deep clone, since the text is contained in a child text node. Cloning any other type of node simply returns a copy of the node. Note that a clone of an unspecified attribute node is specified. If deep is TRUE, all children of the node are

recursively cloned, and the cloned node will have cloned children; a non-deep clone will have no children.

Syntax

```
xmlnode* XmlDomCloneNode(
   xmlctx *xctx,
   xmlnode *node,
   boolean deep);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node
`deep`	`IN`	`TRUE` to recursively clone children

Returns

`(xmlnode *)` duplicate (cloned) node

> **See Also:**
>
> XmlDomImportNode()

XmlDomFreeNode()

Free a node allocated with `XmlDomCreate`*`XXX`*. Frees all resources associated with a node, then frees the node itself. Certain parts of the node are under DOM control, and some parts may be under user control. DOM keeps flags tracking who owns what, and only frees its own allocations. The user is responsible for freeing their own parts of the node before calling XmlDomFreeNode.

Syntax

```
void XmlDomFreeNode(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node to free

> **See Also:**
>
> XmlDomCleanNode()

XmlDomGetAttrs()

Returns a `NamedNodeMap` of attributes of an element node, or `NULL` if it has no attributes. For other node types, `NULL` is returned. Note that if an element once had attributes, but they have all been removed, an empty list will be returned. So, presence of the list does not mean the element has attributes. You must check the size of the list with `XmlDomNumAttrs` or use `XmlDomHasChildNodes` first.

Syntax

```
xmlnamedmap* XmlDomGetAttrs(
   xmlctx *xctx,
   xmlelemnode *elem);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`elem`	`IN`	XML element node

Returns

`(xmlnamedmap *) NamedNodeMap` of node's attributes

> **See Also:**
>
> XmlDomNumAttrs(), XmlDomHasChildNodes()

XmlDomGetChildNodes()

Returns a list of the node's children, or `NULL` if it has no children. Only `Element`, `Document`, DTD, and `DocumentFragment` nodes may have children; all other types will return `NULL`.

Note that an empty list may be returned if the node once had children, but all have been removed! That is, the list may exist but have no members. So, presence of the list alone does not mean the node has children. You must check the size of the list with `XmlDomNumChildNodes` or use `XmlDomHasChildNodes` first.

The xmlnodelist structure is opaque and can only be manipulated with functions in the `NodeList` interface.

The returned list is live; all changes in the original node are reflected immediately.

Syntax

```
xmlnodelist* XmlDomGetChildNodes(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(xmlnodelist *) live NodeList containing all children of node

XmlDomGetDefaultNS()

Gets the default namespace for a node.

Syntax

```
oratext* XmlDomGetDefaultNS(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	element or attribute DOM node

Returns

(oratext *) default namespace for node [data encoding; may be NULL]

XmlDomGetFirstChild()

Returns the first child of a node, or NULL if the node has no children. Only Element, Document, DTD, and DocumentFragment nodes may have children; all other types will return NULL.

Syntax

```
xmlnode* XmlDomGetFirstChild(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
node	IN	XML node

Returns

(xmlnode *) first child of node

> **See Also:**
>
> XmlDomGetLastChild(), XmlDomHasChildNodes(), XmlDomGetChildNodes(), XmlDomNumChildNodes()

XmlDomGetFirstPfnsPair()

This function is to allow implementations an opportunity to speedup the iteration of all available prefix-URI bindings available on a given node. It returns a state structure and the prefix and URI of the first prefix-URI mapping. The state structure should be passed to XmlDomGetNextPfnsPair on the remaining pairs.

Syntax

```
xmlpfnspair* XmlDomGetFirstPfnsPair(
   xmlctx *xctx,
   xmlnode *node,
   oratext **prefix,
   oratext **uri);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
prefix	OUT	prefix of first mapping; data encoding
uri	OUT	URI of first mapping; data encoding

Returns

(xmlpfnspair *) iterating object or NULL of no prefixes

XmlDomGetLastChild()

Returns the last child of a node, or NULL if the node has no children. Only Element, Document, DTD, and DocumentFragment nodes may have children; all other types will return NULL.

Syntax

```
xmlnode* XmlDomGetLastChild(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(xmlnode *) last child of node

> **See Also:**
>
> XmlDomGetFirstChild(), XmlDomHasChildNodes(), XmlDomGetChildNodes(), XmlDomNumChildNodes()

XmlDomGetNextPfnsPair()

This function is to allow implementations an opportunity to speedup the iteration of all available prefix-URI bindings available on a given node. Given an iterator structure from XmlDomGetFirstPfnsPair, returns the next prefix-URI mapping; repeat calls to XmlDomGetNextPfnsPair until NULL is returned.

Syntax

```
xmlpfnspair* XmlDomGetNextPfnsPair(
   xmlctx *xctx
   xmlpfnspair *pfns,
   oratext **prefix,
   oratext **uri);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
prefix	OUT	prefix of next mapping; data encoding
uri	OUT	URI of next mapping; data encoding

Returns

(xmlpfnspair *) iterating object, NULL when no more pairs

XmlDomGetNextSibling()

Returns the node following a node at the same level in the DOM tree. That is, for each child of a parent node, the next sibling of that child is the child which comes after it. If a node is the last child of its parent, or has no parent, `NULL` is returned.

Syntax

```
xmlnode* XmlDomGetNextSibling(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node

Returns

`(xmlnode *)` node immediately following node at same level

> **See Also:**
>
> XmlDomGetPrevSibling()

XmlDomGetNodeLocal()

Returns the namespace local name for a node as a `NULL`-terminated string. If the node's name is not fully qualified (has no prefix), then the local name is the same as the name.

A length-encoded version is available as `XmlDomGetNodeLocalLen` which returns the local name as a pointer and length, for use if the data is known to use `XMLType` backing store.

Syntax

```
oratext* XmlDomGetNodeLocal(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node

Returns

`(oratext *)` local name of node [data encoding]

> **See Also:**
>
> XmlDomGetNodeLocalLen(), XmlDomGetNodePrefix(), XmlDomGetNodeURI()

XmlDomGetNodeLocalLen()

Returns the namespace local name for a node as a length-encoded string. If the node's name is not fully qualified (has no prefix), then the local name is the same as the name.

A `NULL`-terminated version is available as `XmlDomGetNodeLocal` which returns the local name as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the node's data will be stored internally as length-encoded. Using the length-based Get functions will avoid having to copy and `NULL`-terminate the data.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than buflen, then a truncated value will be copied into the buffer and len will return the actual length.

Syntax

```
oratext* XmlDomGetNodeLocalLen(
   xmlctx *xctx,
   xmlnode *node,
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node
`buf`	`IN`	input buffer; optional
`buflen`	`IN`	input buffer length; optional
`len`	`OUT`	length of local name, in characters

Returns

`(oratext *)` local name of node [data encoding]

> **See Also:**
>
> XmlDomGetNodeLocal(), XmlDomGetNodePrefix(), XmlDomGetNodeURILen()

XmlDomGetNodeName()

Returns the (fully-qualified) name of a node (in the data encoding) as a `NULL`-terminated string, for example `bar\0` or `foo:bar\0`.

Note that some node types have fixed names: "`#text`", "`#cdata-section`", "`#comment`", "`#document`", "`#document-fragment`".

A node's name cannot be changed once it is created, so there is no matching `SetNodeName` function.

A length-based version is available as `XmlDomGetNodeNameLen` which returns the node name as a pointer and length, for use if the data is known to use `XMLType` backing store.

Syntax

```
oratext* XmlDomGetNodeName(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node

Returns

`(oratext *)` name of node [data encoding]

> **See Also:**
>
> XmlDomGetNodeNameLen()

XmlDomGetNodeNameLen()

Returns the (fully-qualified) name of a node (in the data encoding) as a length-encoded string, for example `"bar", 3` or `"foo:bar", 7`.

Note that some node types have fixed names: "`#text`", "`#cdata-section`", "`#comment`", "`#document`", "`#document-fragment`".

A node's name cannot be changed once it is created, so there is no matching SetNodeName function.

A `NULL`-terminated version is available as XmlDomGetNodeName which returns the node name as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the node's name will be stored internally as length-encoded. Using the length-encoded `Get`*`XXX`* functions will avoid having to copy and `NULL`-terminate the name.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than buflen, then a truncated value will be copied into the buffer and `len` will return the actual length.

Syntax

```
oratext* XmlDomGetNodeNameLen(
   xmlctx *xctx,
   xmlnode *node,
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
buf	IN	input buffer; optional
buflen	IN	input buffer length; optional
len	OUT	length of name, in characters

Returns

`(oratext *)` name of node, with length of name set in 'len'

> **See Also:**
>
> XmlDomGetNodeName()

XmlDomGetNodePrefix()

Returns the namespace prefix for a node (as a `NULL`-terminated string). If the node's name is not fully qualified (has no prefix), `NULL` is returned.

Syntax

```
oratext* XmlDomGetNodePrefix(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(oratext *) namespace prefix of node [data encoding; may be NULL]

XmlDomGetNodeType()

Returns the type code of a node. The type names and numeric values match the DOM specification:

- ELEMENT_NODE=1
- ATTRIBUTE_NODE=2
- TEXT_NODE=3
- CDATA_SECTION_NODE=4
- ENTITY_REFERENCE_NODE=5
- ENTITY_NODE=6
- PROCESSING_INSTRUCTION_NODE=7
- COMMENT_NODE=8
- DOCUMENT_NODE=9
- DOCUMENT_TYPE_NODE=10
- DOCUMENT_FRAGMENT_NODE=11
- NOTATION_NODE=12

Additional Oracle extension node types are as follows:

- ELEMENT_DECL_NODE
- ATTR_DECL_NODE
- CP_ELEMENT_NODE
- CP_CHOICE_NODE
- CP_PCDATA_NODE
- CP_STAR_NODE
- CP_PLUS_NODE
- CP_OPT_NODE

Syntax

```
xmlnodetype XmlDomGetNodeType(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(xmlnodetype) numeric type-code of the node

XmlDomGetNodeURI()

Returns the namespace URI for a node (in the data encoding) as a NULL-terminated string. If the node's name is not qualified (does not contain a namespace prefix), it will have the default namespace in effect when the node was created (which may be NULL).

A length-encoded version is available as XmlDomGetNodeURILen which returns the URI as a pointer and length, for use if the data is known to use XMLType backing store.

Syntax

```
oratext* XmlDomGetNodeURI(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(oratext *) namespace URI of node [data encoding; may be NULL]

> **See Also:**
>
> XmlDomGetNodeURILen(), XmlDomGetNodePrefix(), XmlDomGetNodeLocal()

XmlDomGetNodeURILen()

Returns the namespace URI for a node (in the data encoding) as length-encoded string. If the node's name is not qualified (does not contain a namespace prefix), it will have the default namespace in effect when the node was created (which may be `NULL`).

A `NULL`-terminated version is available as `XmlDomGetNodeURI` which returns the URI value as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the node's data will be stored internally as length-encoded. Using the length-based Get functions will avoid having to copy and `NULL`-terminate the data.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than buflen, then a truncated value will be copied into the buffer and `len` will return the actual length.

Syntax

```
oratext* XmlDomGetNodeURILen(
   xmlctx *xctx,
   xmlnode *node,
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node
`buf`	`IN`	input buffer; optional
`buflen`	`IN`	input buffer length; optional
`len`	`OUT`	length of URI, in characters

Returns

`(oratext *)` namespace URI of node [data encoding; may be `NULL`]

> **See Also:**
>
> XmlDomGetNodeURI(), XmlDomGetNodePrefix(), XmlDomGetNodeLocal()

XmlDomGetNodeValue()

Returns the "value" (associated character data) for a node as a `NULL`-terminated string. Character and general entities will have been replaced. Only `Attr`, `CDATA`, `Comment`, `ProcessingInstruction` and `Text` nodes have values, all other node types have `NULL` value.

A length-encoded version is available as `XmlDomGetNodeValueLen` which returns the node value as a pointer and length, for use if the data is known to use `XMLType` backing store.

Syntax

```
oratext* XmlDomGetNodeValue(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node

Returns

`(oratext *)` value of node

> **See Also:**
>
> XmlDomSetNodeValue(), XmlDomGetNodeValueLen()

XmlDomGetNodeValueLen()

Returns the "value" (associated character data) for a node as a length-encoded string. Character and general entities will have been replaced. Only `Attr`, `CDATA`, `Comment`, PI and `Text` nodes have values, all other node types have `NULL` value.

A `NULL`-terminated version is available as `XmlDomGetNodeValue` which returns the node value as `NULL`-terminated string. If the backing store is known to be `XMLType`, then the node's data will be stored internally as length-encoded. Using the length-based Get functions will avoid having to copy and `NULL`-terminate the data.

If both the input buffer is non-`NULL` and the input buffer length is nonzero, then the value will be stored in the input buffer. Else, the implementation will return its own buffer.

If the actual length is greater than `buflen`, then a truncated value will be copied into the buffer and len will return the actual length.

Syntax

```
oratext* XmlDomGetNodeValueLen(
   xmlctx *xctx,
   xmlnode *node,
```

```
   oratext *buf,
   ub4 buflen,
   ub4 *len);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
buf	IN	input buffer; optional
buflen	IN	input buffer length; optional
len	OUT	length of value, in bytes

Returns

(oratext *) value of node

> **See Also:**
>
> XmlDomSetNodeValueLen(), XmlDomGetNodeValue()

XmlDomGetNodeValueStream()

Returns the large data for a node and sends it in pieces to the user's output stream. For very large values, it is not always possible to store them [efficiently] as a single contiguous chunk. This function is used to access chunked data of that type. Only XMLType chunks its data (sometimes); XDK's data is always contiguous.

Syntax

```
xmlerr XmlDomGetNodeValueStream(
   xmlctx *xctx,
   xmlnode *node,
   xmlostream *ostream);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
ostream	IN	output stream object

Returns

`(xmlerr)` numeric error code, `0` on success

> **See Also:**
>
> XmlDomSetNodeValueStream(), XmlDomGetNodeValue(), XmlDomGetNodeValueLen()

XmlDomGetOwnerDocument()

Returns the `Document` node associated with a node. Each node may belong to only one document, or may not be associated with any document at all (such as immediately after `XmlDomCreateElem`, and so on). The "owning" document [node] is returned, or `NULL` for an orphan node.

Syntax

```
xmldocnode* XmlDomGetOwnerDocument(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node

Returns

`(xmldocnode *)` document node is in

XmlDomGetParentNode()

Returns a node's parent node. All nodes types except `Attr`, `Document`, `DocumentFragment`, `Entity`, and `Notation` may have a parent (these five exceptions always have a `NULL` parent). If a node has just been created but not yet added to the DOM tree, or if it has been removed from the DOM tree, its parent is also `NULL`.

Syntax

```
xmlnode* XmlDomGetParentNode(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
node	IN	XML node

Returns

(xmlnode *) parent of node

XmlDomGetPrevSibling()

Returns the node preceding a node at the same level in the DOM tree. That is, for each child of a parent node, the previous sibling of that child is the child which came before it. If a node is the first child of its parent, or has no parent, NULL is returned.

Syntax

```
xmlnode* XmlDomGetPrevSibling(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(xmlnode *) node immediately preceding node at same level

> **See Also:**
>
> XmlDomGetNextSibling()

XmlDomGetPullNodeAsBinaryStream()

Returns the address of a binary stream using the pull paradigm.

Syntax

```
orastream *XmlDomGetPullNodeAsBinaryStream(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
node	IN	XML node; may be RAW or BLOB, otherwise the function returns NULL

Returns

(orastream *) the readable binary stream; use OraStreamRead() on the output, not OraStreamReadChar()

XmlDomGetPullNodeAsCharacterStream()

Returns the address of a character stream using the pull paradigm.

Syntax

```
orastream *XmlDomGetPullNodeAsCharacterStream(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node; may be any type supported by XML DB

Returns

(orastream *) the readable character stream; use OraStreamReadChar() on the output, not OraStreamRead().

XmlDomGetPushNodeAsBinaryStream()

Returns the address of a binary stream, as an OUT ostream parameter, using the push paradigm.

Syntax

```
xmlerr XmlDomGetPushNodeAsBinaryStream(
   xmlctx *xctx,
   xmlnode *node,
   orastream *ostream);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node; may be RAW or BLOB, otherwise returns an error
ostream	OUT	application implementation of orastream; use OraStreamWrite() to write the value, not OraStreamWriteChar()

Returns

`(xmlerr *)` error code, `XMLERR_OK []` on success

XmlDomGetPushNodeAsCharacterStream()

Returns the address of a character stream, as an `OUT ostream` parameter, using the push paradigm.

Syntax

```
xmlerr XmlDomGetPushNodeAsCharacterStream(
   xmlctx *xctx,
   xmlnode *node,
   orastream *ostream);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node; any type supported by XML DB
`ostream`	`IN`	application implementation of `orastream`; use `OraStreamWriteChar()` to write the value, not `OraStreamWrite()`

Returns

`(xmlerr *)` error code, `XMLERR_OK []` on success

XmlDomGetSourceEntity()

Returns the external entity node whose inclusion caused the creation of the given node.

Syntax

```
xmlentnode* XmlDomGetSourceEntity(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node

Returns

`(xmlentnode *)` entity node if the input is from an external entity

XmlDomGetSourceLine()

Returns the line# in the original source where the node started. The first line in every input is line #1.

Syntax

```
ub4 XmlDomGetSourceLine(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(ub4) line number of node in original input source

XmlDomGetSourceLocation()

Return source location (path, URI, and so on) of node. Note this will be in the compiler encoding, not the data encoding!

Syntax

```
oratext* XmlDomGetSourceLocation(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(oratext *) full path of input source [in compiler encoding]

XmlDomHasAttrs()

Test if an element has attributes. Returns TRUE if any attributes of any sort are defined (namespace or regular).

Syntax

```
boolean XmlDomHasAttrs(
   xmlctx *xctx,
   xmlelemnode *elem);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	XML element node

Returns

(boolean) TRUE if element has attributes

XmlDomHasChildNodes()

Test if a node has children. Only Element, Document, DTD, and DocumentFragment nodes may have children. Note that just because XmlDomGetChildNodes returns a list does not mean the node actually has children, since the list may be empty, so a non-NULL return from XmlDomGetChildNodes should not be used as a test.

Syntax

```
boolean XmlDomHasChildNodes(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(boolean) TRUE if the node has any children

XmlDomInsertBefore()

Inserts the node newChild before the existing child node refChild in the parent node. If refChild is NULL, appends to parent's children as for each XmlDomAppendChild; otherwise it must be a child of the given parent. If newChild is a DocumentFragment, all of its children are inserted (in the same order) before refChild; the DocumentFragment node itself is not. If newChild is already in the DOM tree, it is first removed from its current position.

Syntax

```
xmlnode* XmlDomInsertBefore(
   xmlctx *xctx,
   xmlnode *parent,
   xmlnode *newChild,
   xmlnode *refChild);
```

Parameter	In/Out	Description
xctx	IN	XML context
parent	IN	parent that receives a new child
newChild	IN	node to insert
refChild	IN	reference node

Returns

(xmlnode *) node being inserted

> **See Also:**
>
> XmlDomAppendChild(), XmlDomReplaceChild(), XmlDomRemoveChild()

XmlDomNormalize()

Normalizes the subtree rooted at an element, merges adjacent Text nodes children of elements. Note that adjacent Text nodes will never be created during a normal parse, only after manipulation of the document with DOM calls.

Syntax

```
void XmlDomNormalize(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

XmlDomNumAttrs()

Returns the number of attributes of an element. Note that just because a list is returned by XmlDomGetAttrs does not mean it contains any attributes; it may be an empty list with zero length.

Syntax

```
ub4 XmlDomNumAttrs(
   xmlctx *xctx,
   xmlelemnode *elem);
```

Parameter	In/Out	Description
xctx	IN	XML context
elem	IN	XML element node

Returns

(ub4) number of attributes of node

XmlDomNumChildNodes()

Returns the number of children of a node. Only Element, Document, DTD, and DocumentFragment nodes may have children, all other types return 0. Note that just because XmlDomGetChildNodes returns a list does not mean that it contains any children; it may be an empty list with zero length.

Syntax

```
ub4 XmlDomNumChildNodes(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node

Returns

(ub4) number of children of node

XmlDomPrefixToURI()

Given a namespace prefix and a node, returns the namespace URI mapped to that prefix. If the given node doesn't have a matching prefix, its parent is tried, then its parent, and so on, all the way to the root node. If the prefix is undefined, NULL is returned.

Syntax

```
oratext* XmlDomPrefixToURI(
   xmlctx *xctx,
   xmlnode *node,
   oratext *prefix);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
node	IN	XML node
prefix	IN	prefix to map

Returns

(oratext *) URI for prefix [data encoding; NULL if no match]

XmlDomRemoveChild()

Removes a node from its parent's list of children and returns it. The node is orphaned; its parent will be NULL after removal.

Syntax

```
xmlnode* XmlDomRemoveChild(
   xmlctx *xctx,
   xmlnode *oldChild);
```

Parameter	In/Out	Description
xctx	IN	XML context
oldChild	IN	node to remove

Returns

(xmlnode *) node removed

> **See Also:**
>
> XmlDomAppendChild(), XmlDomInsertBefore(), XmlDomReplaceChild()

XmlDomRenameNode()

Updates the name of a node, for element and attribute nodes only.

If the prefix does not have a current mapping, the user should add the mapping by creating an xmlns attribute and associating it with this element node by calling XmlDomSetAttrNodeNS()XmlDomSetAttrNodeNS(). A namespace attribute node cannot be modified.

Syntax

```
xmlnode* XmlDomRenameNode(
   xmlctx *xctx,
```

```
   xmlnode *node,
   oratext *tagname);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
tagname	IN	The new tagname

Returns

(xmlnode *) the changed node

> **See Also:**
>
> XmlDomSetAttrNodeNS()

XmlDomRenameNodeNS()

Updates the name and URI of a node, for element and attribute nodes only.

If the prefix does not have a current mapping, the user should add the mapping by creating an xmlns attribute and associating it with this element node by calling XmlDomSetAttrNodeNS()XmlDomSetAttrNodeNS(). A namespace attribute node cannot be modified.

Syntax

```
xmlnode* XmlDomRenameNodeNS(
   xmlctx *xctx,
   xmlnode *node,
   oratext *uri,
   oratext *tagname);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
uri	IN	The new URI; if NULL, retains existing URI
tagname	IN	The new tagname

Returns

(xmlnode *) the changed node.

> **See Also:**
>
> XmlDomSetAttrNodeNS()

XmlDomReplaceChild()

Replaces the child node `oldChild` with the new node `newChild` in `oldChild`'s parent, and returns `oldChild` (which is now orphaned, with a `NULL` parent). If `newChild` is a `DocumentFragment`, all of its children are inserted in place of `oldChild`; the `DocumentFragment` node itself is not. If `newChild` is already in the DOM tree, it is first removed from its current position.

Syntax

```
xmlnode* XmlDomReplaceChild(
   xmlctx *xctx,
   xmlnode *newChild,
   xmlnode *oldChild);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`newChild`	`IN`	new node that is substituted
`oldChild`	`IN`	old node that is replaced

Returns

`(xmlnode *)` node replaced

> **See Also:**
>
> XmlDomAppendChild(), XmlDomInsertBefore(), XmlDomRemoveChild()

XmlDomSetDefaultNS()

Set the default namespace for a node

Syntax

```
void XmlDomSetDefaultNS(
   xmlctx *xctx,
   xmlnode *node,
   oratext *defns);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	element or attribute DOM node
defns	IN	new default namespace for the node

XmlDomSetNodePrefix()

Sets the namespace prefix of node (as NULL-terminated string). Does not verify the prefix is defined. Just causes a new qualified name to be formed from the new prefix and the old local name; the new qualified name will be under DOM control and should not be managed by the user.

Syntax

```
void XmlDomSetNodePrefix(
   xmlctx *xctx,
   xmlnode *node,
   oratext *prefix);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
prefix	OUT	new namespace prefix

XmlDomSetNodeValue()

Sets a node's value (character data) as a NULL-terminated string. Does not allow setting the value to NULL. Only Attr, CDATA, Comment, PI and Text nodes have values; trying to set the value of another type of node is a no-op. The new value must be in the data encoding. It is not verified, converted, or checked.

The value is not copied, its pointer is just stored. The user is responsible for persistence and freeing of that data.

Syntax

```
xmlerr XmlDomSetNodeValue(
   xmlctx *xctx,
   xmlnode *node,
   oratext *value);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
value	IN	node's new value; data encoding; user control

Returns

(xmlerr) numeric error code, 0 on success

> **See Also:**
>
> XmlDomGetNodeValue(), XmlDomSetNodeValueLen()

XmlDomSetNodeValueLen()

Sets the value (associated character data) for a node as a length-encoded string.

A NULL-terminated version is available as XmlDomSetNodeValue which takes the node value as a NULL-terminated string. If the backing store is known to be XMLType, then the node's data will be stored internally as length-encoded. Using the length-based Set functions will avoid having to copy and NULL-terminate the data.

Syntax

```
xmlerr XmlDomSetNodeValueLen(
   xmlctx *xctx,
   xmlnode *node,
   oratext *value,
   ub4 len);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node
value	IN	node's new value; data encoding; user control
len	IN	length of value, in bytes

Returns

(xmlerr) numeric error code, 0 on success

> **See Also:**
>
> XmlDomSetNodeValueLen(), XmlDomSetNodeValue()

XmlDomSetNodeValueStream()

Sets the large "value" (character data) for a node piecemeal from an input stream. For very large values, it is not always possible to store them [efficiently] as a single contiguous chunk. This function is used to store chunked data of that type. Used only for `XMLType` data; XDK's data is always contiguous.

Syntax

```
xmlerr XmlDomSetNodeValueStream(
   xmlctx *xctx,
   xmlnode *node,
   xmlistream *istream);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	XML node
`istream`	`IN`	input stream object

Returns

`(xmlerr)` numeric error code, `0` on success

> **See Also:**
>
> XmlDomGetNodeValueStream(), XmlDomSetNodeValue()

XmlDomSetPullNodeAsBinaryStream()

Returns the address of a binary input stream, as an `OUT istream` parameter, using the pull paradigm.

Syntax

```
xmlerr *XmlDomSetPullNodeAsBinaryStream(
   xmlctx *xctx,
   xmlnode *node
   orastream *istream);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node; may be RAW or BLOB, otherwise returns an error
istream	OUT	input stream object; the method OraStreamRead() must be used to read this value, not OraStreamReadChar()

Returns

(xmlerr *) error code, XMLERR_OK [] on success

XmlDomSetPullNodeAsCharacterStream()

Returns the address of an input character stream, as an OUT istream parameter, using the pull paradigm.

Syntax

```
xmlerr *XmlDomSetPullNodeAsCharacterStream(
   xmlctx *xctx,
   xmlnode *node
   orcharacterinputstream *istream);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node; may be any type supported by XML DB
istream	OUT	input stream object; the method OraStreamReadChar() must be used to read this value, not OraStreamRead().

Returns

(xmlerr *) error code, XMLERR_OK [] on success

XmlDomSetPushNodeAsBinaryStream()

Returns the address of an input binary stream using the push paradigm.

Syntax

```
orastream* XmlDomSetPushNodeAsBinaryStream(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
node	IN	XML node; may be RAW or BLOB

Returns

(orastream *) the binary stream; to read the output, use OraStreamWrite() instead of OraStreamWriteChar()

XmlDomSetPushNodeAsCharacterStream()

Returns the address of a character stream using the push paradigm.

Syntax

```
orastream *XmlDomSetPushNodeAsCharacterStream(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	XML node; any type supported by XML DB

Returns

(orastream *) the character stream; to read the output, use OraStreamWriteChar() instead of OraStreamWrite()

XmlDomValidate()

Given a root node, validates it against the current DTD.

Syntax

```
xmlerr XmlDomValidate(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
node	IN	node to validate

Returns

(xmlerr) error code, XMLERR_OK [0] means node is valid

NodeList Interface for DOM XML C APIs

The following table summarizes the methods available through the `NodeList` interface of DOM for XML C APIs.

Table 3-9 Summary of NodeList DOM Methods for XML C Implementation

Function	Summary
XmlDomFreeNodeList()	Free a node list returned by XmlDomGetElemsByTag, and so on.
XmlDomGetNodeListItem()	Return n^{th} node in list.
XmlDomGetNodeListLength()	Return length of node list.

XmlDomFreeNodeList()

Free a node list returned by `XmlDomGetElemsByTag` or related functions, releasing all resources associated with it. If given a node list that is part of the DOM proper (such as the children of a node), does nothing.

Syntax

```
void XmlDomFreeNodeList(
   xmlctx *xctx,
   xmlnodelist *list);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`list`	`IN`	`NodeList` to free

> **See Also:**
>
> XmlDomGetElemsByTag(), XmlDomGetElemsByTagNS(), XmlDomGetChildrenByTag(), XmlDomGetChildrenByTagNS()

XmlDomGetNodeListItem()

Return nth node in a node list. The first item is index `0`.

Syntax

```
xmlnode* XmlDomGetNodeListItem(
   xmlctx *xctx,
   xmlnodelist *list,
   ub4 index);
```

Parameter	In/Out	Description
xctx	IN	XML context
list	IN	NodeList
index	IN	index into list

Returns

(xmlnode *) node at the nth position in node list [or NULL]

> **See Also:**
>
> XmlDomGetNodeListLength(), XmlDomFreeNodeList()

XmlDomGetNodeListLength()

Returns the number of nodes in a node list (its length). Note that nodes are referred to by index, so the range of valid indexes is 0 through length-1.

Syntax

```
ub4 XmlDomGetNodeListLength(
   xmlctx *xctx,
   xmlnodelist *list);
```

Parameter	In/Out	Description
xctx	IN	XML context
list	IN	NodeList

Returns

(ub4) number of nodes in node list

> **See Also:**
>
> XmlDomGetNodeListItem(), XmlDomFreeNodeList()

Notation Interface for DOM XML C APIs

The following table summarizes the methods available through the `Notation` interface of DOM for XML C APIs.

Table 3-10 Summary of NodeList DOM Methods for XML C Implementation

Function	Summary
XmlDomGetNotationPubID()	Get notation's public ID
XmlDomGetNotationSysID()	Get notation's system ID.

XmlDomGetNotationPubID()

Return a notation's public identifier (in the data encoding).

Syntax

```
oratext* XmlDomGetNotationPubID(
   xmlctx *xctx,
   xmlnotenode *note);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`note`	`IN`	`Notation` node

Returns

`(oratext *)` notation's public identifier [data encoding; may be `NULL`]

> **See Also:**
>
> XmlDomGetNotationSysID()

XmlDomGetNotationSysID()

Return a notation's system identifier (in the data encoding).

Syntax

```
oratext* XmlDomGetNotationSysID(
   xmlctx *xctx,
   xmlnotenode *note);
```

Parameter	In/Out	Description
xctx	IN	XML context
note	IN	Notation node

Returns

(oratext *) notation's system identifier [data encoding; may be NULL]

> **See Also:**
>
> XmlDomGetNotationPubID()

ProcessingInstruction Interface for DOM XML C APIs

The following table summarizes the methods available through the ProcessingInstruction interface of DOM for XML C APIs.

Table 3-11 Summary of ProcessingInstruction DOM Package Methods for XML C Implementation

Function	Summary
XmlDomGetPIData()	Get processing instruction's data.
XmlDomGetPITarget()	Get PI's target.
XmlDomSetPIData()	Set processing instruction's data.

XmlDomGetPIData()

Returns the content (data) of a processing instruction (in the data encoding). If the node is not a ProcessingInstruction, returns NULL. The content is the part from the first non-whitespace character after the target until the ending "?>".

Syntax

```
oratext* XmlDomGetPIData(
   xmlctx *xctx,
   xmlpinode *pi);
```

Parameter	In/Out	Description
xctx	IN	XML context
pi	IN	ProcessingInstruction node

Returns

`(oratext *)` processing instruction's data [data encoding]

> **See Also:**
>
> XmlDomGetPITarget(), XmlDomSetPIData()

XmlDomGetPITarget()

Returns a processing instruction's target string. If the node is not a `ProcessingInstruction`, returns `NULL`. The target is the first token following the markup that begins the `ProcessingInstruction`. All `ProcessingInstruction`s must have a target, though the data part is optional.

Syntax

```
oratext* XmlDomGetPITarget(
   xmlctx *xctx,
   xmlpinode *pi);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`pi`	`IN`	`ProcessingInstruction` node

Returns

`(oratext *)` processing instruction's target [data encoding]

> **See Also:**
>
> XmlDomGetPIData(), XmlDomSetPIData()

XmlDomSetPIData()

Sets a `ProcessingInstruction`'s content, which must be in the data encoding. It is not permitted to set the data to `NULL`. If the node is not a `ProcessingInstruction`, does nothing. The new data is not verified, converted, or checked.

Syntax

```
void XmlDomSetPIData(
   xmlctx *xctx,
   xmlpinode *pi,
   oratext *data);
```

Parameter	In/Out	Description
xctx	IN	XML context
pi	IN	ProcessingInstruction node
data	IN	ProcessingInstruction's new data; data encoding

> **See Also:**
>
> XmlDomGetPITarget(), XmlDomGetPIData()

Text Interface for DOM XML C APIs

The following table summarizes the methods available through the Text interface of DOM for XML C APIs.

Table 3-12 Summary of Text DOM Methods for XML C Implementation

Function	Summary
XmlDomSplitText()	Split text node in to two.

XmlDomSplitText()

Splits a single text node into two text nodes; the original data is split between them. If the given node is not type text, or the offset is outside of the original data, does nothing and returns NULL. The offset is zero-based, and is in characters, not bytes. The original node is retained, its data is just truncated. A new text node is created which contains the remainder of the original data, and is inserted as the next sibling of the original. The new text node is returned.

Syntax

```
xmltextnode* XmlDomSplitText(
   xmlctx *xctx,
   xmltextnode *textnode,
   ub4 offset);
```

Parameter	In/Out	Description
xctx	IN	XML context
textnode	IN	Text node

Parameter	In/Out	Description
`offset`	`IN`	`0`-based character count at which to split text

Returns

`(xmltextnode *)` new text node

> **See Also:**
>
> XmlDomGetCharData(), XmlDomAppendData(), XmlDomInsertData(), XmlDomDeleteData(), XmlDomReplaceData()

4
Package Event for XML C APIs

The following table summarizes the methods available through the `Event` interface for XML C APIs.

Table 4-1 Summary of Event Methods for XML C Implementation

Function	Summary
XmlEvCleanPPCtx()	Cleans up intenal structures related to a parse operation. This will not destroy the event context. The event context can be reused after this call.
XmlEvCreatePPCtx()	Creates an Event context in pull-parse mode.
XmlEvCreateSVCtx()	Creates an event context for the streaming validadtor.
XmlEvDestroyPPCtx()	Destroys the event context. Terminates parsing. May be called at any time during a parsing operation.
XmlEvDestroySVCtx()	Terminates an event context created by a streaming validator.
XmlEvGetAttrCount()	Retrieves the number of attributes for the `XML_EVENT_START_ELEMENT` event.
XmlEvGetAttrDeclBody()	Retrieves the attribute body in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`. Also, provides the length as an `OUT len` parameter.
XmlEvGetAttrDeclBody0()	Retrieves the `NULL`-terminated attribute body in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.
XmlEvGetAttrDeclCount()	Retreives the number of attributes in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.
XmlEvGetAttrDeclElName()	Retrieves the element name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`. Also, provides the length as an `OUT len` parameter.
XmlEvGetAttrDeclElName0()	Retrieves the `NULL`-terminated element name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.
XmlEvGetAttrDeclLocalName()	Retrieves the local name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`. Also, provides the length as an `OUT len` parameter.
XmlEvGetAttrDeclLocalName0()	Retrieves the `NULL`-terminated local name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.
XmlEvGetAttrDeclName()	Retrieves the attribute name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`. Also, provides the length as an `OUT len` parameter.
XmlEvGetAttrDeclName0()	Retrieves the `NULL`-terminated attribute name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.
XmlEvGetAttrDeclPrefix()	Retrieves the attribute prefix in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`. Also, provides the length as an `OUT len` parameter.

Table 4-1 (Cont.) Summary of Event Methods for XML C Implementation

Function	Summary
XmlEvGetAttrDeclPrefix0()	Retrieves the NULL-terminated attribute prefix in attribute declaration XML_EVENT_ATTLIST_DECLARATION.
XmlEvGetAttrID()	Retrieves the ID for the attribute's QNAME, for XML_EVENT_START_ELEMENT events.
XmlEvGetAttrLocalName()	Retrieves the attribute local name for the XML_EVENT_START_ELEMENT events. Also, provides the length as an OUT len parameter.
XmlEvGetAttrLocalName0()	Retrieves the NULL-terminated attribute name for the XML_EVENT_START_ELEMENT events.
XmlEvGetAttrName()	Retrieves the attribute name for the XML_EVENT_START_ELEMENT events. Also, provides the length as an OUT len parameter.
XmlEvGetAttrName0()	Retrieves the NULL-terminated attribute name for the XML_EVENT_START_ELEMENT events.
XmlEvGetAttrPrefix()	Retrieves the prefix tag for XML_EVENT_START_ELEMENT events, and also returns the length of the event as an OUT len parameter.
XmlEvGetAttrPrefix0()	Retrieves the NULL-terminated attribute prefix for the XML_EVENT_START_ELEMENT events.
XmlEvGetAttrURI()	Retrieves the attribute URI for the XML_EVENT_START_ELEMENT events. Also, provides the length as an OUT len parameter.
XmlEvGetAttrURI0()	Retrieves the NULL-terminated attribute URI for the XML_EVENT_START_ELEMENT events.
XmlEvGetAttrUriID()	Retrieves the ID for the attribute's URI, for XML_EVENT_START_ELEMENT events.
XmlEvGetAttrValue()	Retrieves the attribute value for one of the XML_EVENT_START_ELEMENT events, and also returns the length of the event as an OUT len parameter.
XmlEvGetAttrValue0()	Retrieves the NULL-terminated attribute value for the XML_EVENT_START_ELEMENT events.
XmlEvGetElDeclContent()	Retrieves the element declaration content for XML_EVENT_ELEMENT_DECLARATION. Also, provides the length as an OUT len parameter.
XmlEvGetElDeclContent0()	Retrieves the element declaration content for XML_EVENT_ELEMENT_DECLARATION.
XmlEvGetEncoding()	Returns the value of the encoding specified.
XmlEvGetError()	Retrieves the error number when the XML_EVENT_FATAL_ERROR or XML_EVENT_ERROR event is returned by an XmlEvNext()
XmlEvGetName()	Retrurns the name of for either XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events, and the length of the event in the OUT len parameter.

Table 4-1 (Cont.) Summary of Event Methods for XML C Implementation

Function	Summary
XmlEvGetName0()	Retrieves a NULL-terminated name for either XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events
XmlEvGetLocalName()	Retrieves the local name tag for either XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events, and also returns the length of the event as an OUT len parameter:
XmlEvGetLocalName0()	Retrieves a NULL-terminated local name tag for either XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events, and also returns the length of the event as an OUT len parameter:
XmlEvGetLocation()	Retrieves the location during parsing, as OUT parameters for the line number of the input stream and its path.
XmlEvGetPIData()	Retrieves the text for XML_EVENT_PI or XML_EVENT_PI_CONT events, and also returns the length of the event as an OUT len parameter.
XmlEvGetPIData0()	Retrieves NULL-terminated text for XML_EVENT_PI or XML_EVENT_PI_CONT events.
XmlEvGetPITarget()	Retrieves the target for XML_EVENT_PI and XML_EVENT_PI_CONT events, and also returns the length of the event as an OUT len parameter.
XmlEvGetPITarget0()	Retrieves the NULL-terminated target for XML_EVENT_PI and XML_EVENT_PI_CONT events.
XmlEvGetPEIsGen()	Determines if the general entity was declared, XML_EVENT_PE_DECLARATION.
XmlEvGetPERepl()	Retrieves the replacement text of PE declaration, XML_EVENT_PE_DECLARATION. Also, provides the length as an OUT len parameter.
XmlEvGetPERepl0()	Retrieves the NULL-terminated replacement text of PE declaration, XML_EVENT_PE_DECLARATION.
XmlEvGetPrefix()	Retrieves the prefix tag for one of either XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events, and also returns the length of the event as an OUT len parameter.
XmlEvGetPrefix0()	Retrieves the prefix tag for one of either XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events..
XmlEvGetPubId()	Retrieves the public id for XML_EVENT_PE_DECLARATION, XML_EVENT_UE_DECLARATION, or XML_EVENT_NOTATION_DECLARATION events; also, provides the length as an OUT len parameter.
XmlEvGetPubId0()	Retrieves the NULL-terminated public id for XML_EVENT_PE_DECLARATION, XML_EVENT_UE_DECLARATION, or XML_EVENT_NOTATION_DECLARATION events.

Table 4-1 (Cont.) Summary of Event Methods for XML C Implementation

Function	Summary
XmlEvGetSysId()	Retrieves the system id for XML_EVENT_PE_DECLARATION, XML_EVENT_UE_DECLARATION, or XML_EVENT_NOTATION_DECLARATION events; also, provides the length as an OUT len parameter.
XmlEvGetSysId0()	Retrieves the NULL-terminated system id for XML_EVENT_PE_DECLARATION, XML_EVENT_UE_DECLARATION, or XML_EVENT_NOTATION_DECLARATION events.
XmlEvGetTagID()	Retrieves the ID for the tag's QNAME, for XML_EVENT_START_ELEMENT events.
XmlEvGetTagUriID()	Retrieves the ID for the tag's URI, for XML_EVENT_START_ELEMENT and XML_EVENT_END_ELEMENT events.
XmlEvGetText()	Retrieves the text for XML_EVENT_CHARACTERS, XML_EVENT_CHARACTERS_CONT, XML_EVENT_SPACE, XML_EVENT_SPACE_CONT, XML_EVENT_COMMENT, XML_EVENT_COMMENT_CONT, XML_EVENT_CDATA, and XML_EVENT_CDATA_CONT events, and also returns the length of the event as an OUT len parameter.
XmlEvGetText0()	Retrieves the NULL-terminated text for XML_EVENT_CHARACTERS, XML_EVENT_CHARACTERS_CONT, XML_EVENT_SPACE, XML_EVENT_SPACE_CONT, XML_EVENT_COMMENT, XML_EVENT_COMMENT_CONT, XML_EVENT_CDATA, and XML_EVENT_CDATA_CONT events.
XmlEvGetUENdata()	Retrieves the ndata for XML_EVENT_UE_DECLARATION event, and also returns the length of the event as an OUT len parameter.
XmlEvGetUENdata0()	Retrieves the NULL-terminated ndata for XML_EVENT_UE_DECLARATION event.
XmlEvGetURI()	Retrieves the URI tag for XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events, and also returns the length of the event as an OUT len parameter:
XmlEvGetURI0()	Retrieves the NULL-terminated URI tag for XML_EVENT_START_ELEMENT or XML_EVENT_END_ELEMENT events.
XmlEvGetVersion()	Provides information about version specification in XML declaration for the XML_EVENT_START_DOCUMENT event.
XmlEvIsEncodingSpecified()	Provides information about encoding specification in XML declaration for the XML_EVENT_START_DOCUMENT event.
XmlEvIsNamespaceAttr()	Determines if an attribute is a namespace atrribute for XML_EVENT_START_ELEMENT event.
XmlEvIsStandalone()	Provides information about standalone specification in XML declaration for the XML_EVENT_START_DOCUMENT event
XmlEvNext()	Gets the next event and advances the parser.
XmlEvNextTag()	Advances the parser to the next tag event.

Table 4-1 (Cont.) Summary of Event Methods for XML C Implementation

Function	Summary
XmlEvLoadPPDoc()	Loads a new document and configures it for pull parsing.
XmlEvSchemaValidate()	Validates XML documents represented by events.

XmlEvCleanPPCtx()

Cleans up intenal structures related to a parse operation. This will not destroy the event context. The event context can be reused after this call.

Syntax

```
xmlerr XmlEvCleanPPCtx(
   xmlctx *xctx,
   xmlevctx *evctx);
```

Parameter	In/Out	Description
xctx	IN	XML context
evtx	IN	XmlEvents context

Returns

(xmlerr) the error number

XmlEvCreatePPCtx()

Creates an Event context in pull-parse mode.

The document is loaded using XmlEvLoadPPDoc. The actual parsing is driven by multiple calls to XmlEvNext(). After each call, relevant information may be retrieved by calls to the various XmlEvGet*XXX*() functions. Basic set of properties are the same as for XmlLoadDom. Input source should be specified with XmlEvLoadPPDoc() call.

Syntax

```
xmlevctx *XmlEvCreatePPCtx(
   xmlctx *xctx,
   xmlerr  *xerr,
   list);
```

Parameter	In/Out	Description
xctx	IN	XML context
xerr	IN	numeric error code, XMLERR_OK[0] on success

Parameter	In/Out	Description

list	IN	These additional properties should be supplied with a terminal NULL:
		• ("expand_entities", boolean) that, when FALSE, causes parsed non-parameter entity references not be expanded. By default such references are expanded.
		• ("use_buffer", buffer) is the address of a buffer that when specified, will use the buffer to collect data that should be returned back to the user. The get*XXX*() functions will return this buffer as a data pointer.
		• ("use_buffer_len", lengthOfBuffer) is the number of bytes in a buffer, the actual lenght of the buffer, and no more than the specified length is collected. In the case, only part of the data is collected, generating the CONT flavor of the event is generated. Subsequent calls to XmlEventsNext provide additional data. Sequence of CONT-flavored events is always terminated by a non -CONT event. The buffer may be only partially filled.
		• ("get_id_callback", function) is the addres for the callback function, to convert text base names to 8-byte IDs. Once such function is supplied, the user is allowed to use XmlEvGetTagID, XmlEvGetAttrID, XmlEvGetTagUriID, and XmlEvGetAttrUriID.
		• ("raw_buffer_len", length) is the number of bytes in a buffer. By default, this parameter is 256K. Raw buffer is used to read the input data and perform character conversion, and also to convert CRLFs and CRs to LFs.
		• ("error_callback", callback) provides the address of a callback function tha is invoked to signal illegal use of an API for that event.
		These optional parameters should be used in the following manner:
		`xmlevctx *XmlEvCreatePPCtx(`
		`  xmlctx *xctx,`
		`  xmlerr  *xerr,`
		`  ("expand_entities", mode),`
		`  ("use_buffer", buffer),`
		`  ("use_buffer_len", length),`
		`  ("get_id_callback", function),`
		`  ("raw_buffer_len",  length),`
		`  ("error_callback", callback) );`

Returns

(xmlevctx) Event contex to be passed on subsequent calls to XmlEvNext()

XmlEvCreateSVCtx()

Creates an event context for the streaming validadtor. Initializes the streaming validator and returns an event context that can be used in subsequent calls.

Use in conjunction with XmlEvDestroySVCtx()XmlEvDestroySVCtx(). This is a transparent method. An alternate approach would be to use the opague XmlEvSchemaValidate()XmlEvSchemaValidate().

Syntax

```
xmlevctx *XmlEvCreateSVCtx(
   xmlctx *xctx,
   xsdctx *sctx,
   xmlevctx *docEvCtx,
   xmlerr *err);
```

Parameter	In/Out	Description
xctx	IN	XML context; must be valid
sctx	IN	Schema context; must be valid
docEvCtx	IN	Event context for the document that is validated
err	OUT	numeric error code, XMLERR_OK[0] on success

Returns

(xmlevctx) Event contex to be passed on subsequent calls to XmlEvNext()XmlEvNext()

XmlEvDestroyPPCtx()

Destroys the event context. Terminates parsing. May be called at any time during a parsing operation.

Syntax

```
void XmlEvDestroyPPCtx(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

XmlEvDestroySVCtx()

Terminates an event context created by a streaming validator. Returns XMLERR_OK[0] on success, or a numeric error code on failure.

Use in conjunction with XmlEvCreateSVCtx()XmlEvCreateSVCtx(). This is a transparent method. An alternate approach would be to use the opague XmlEvSchemaValidate()XmlEvSchemaValidate().

Syntax

```
xmlerr XmlEvDestroySVCtx(
   xmlctx *xctx,
   xmlevctx *evCtx);
```

Parameter	In/Out	Description
xctx	IN	XML context
evCtx	IN	Event context that should be terminatedt

Returns

(xmlerr) the error number

XmlEvGetAttrCount()

Retrieves the number of attributes for the XML_EVENT_START_ELEMENT event.

Syntax

```
ub4  XmlEvGetAttrCount(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(ub4) the number of attributes

XmlEvGetAttrDeclBody()

Retrieves the attribute body in attribute declaration XML_EVENT_ATTLIST_DECLARATION. Also, provides the length as an OUT len parameter.

Syntax

```
oratext *XmlEvGetAttrDeclBody(
   xmlevctx *evctx,
   ub4 index,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute
len	OUT	the length

Returns

`(oratext*)` the declaration body

XmlEvGetAttrDeclBody0()

Retrieves the `NULL`-terminated attribute body in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.

Syntax

```
oratext *XmlEvGetAttrDeclBody0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute

Returns

`(oratext*)` the declaration body

XmlEvGetAttrDeclCount()

Retreives the number of attributes in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.

Syntax

```
ub4 XmlEvGetAttrDeclCount(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(ub4)` number of attributes

XmlEvGetAttrDeclElName()

Retrieves the element name in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`. Also, provides the length as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetAttrDeclElName(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
len	OUT	the length

Returns

(oratext*) the element name

XmlEvGetAttrDeclElName0()

Retrieves the NULL-terminated element name in attribute declaration XML_EVENT_ATTLIST_DECLARATION.

Syntax

```
oratext *XmlEvGetAttrDeclElName0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) the element name

XmlEvGetAttrDeclLocalName()

Retrieves the local name of the attribute declaration event, XML_EVENT_ATTLIST_DECLARATION. Also, provides the length as an OUT len parameter.

Syntax

```
oratext *XmlEvGetAttrDeclLocalName(
   xmlevctx *evctx,
   ub4 index
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Parameter	In/Out	Description
index	IN	index of the attribute
len	OUT	the length

Returns

(oratext*) the local name

XmlEvGetAttrDeclLocalName0()

Retrieves the NULL-terminated local name in attribute declaration event, XML_EVENT_ATTLIST_DECLARATION.

Syntax

```
oratext *XmlEvGetAttrDeclLocalName0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute

Returns

(oratext*) the local name

XmlEvGetAttrDeclName()

Retrieves the attribute name in attribute declaration XML_EVENT_ATTLIST_DECLARATION. Also, provides the length as an OUT len parameter.

Syntax

```
oratext *XmlEvGetAttrDeclName(
   xmlevctx *evctx,
   ub4 index
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute

Parameter	In/Out	Description
len	OUT	the length

Returns

(oratext*) the attribute name

XmlEvGetAttrDeclName0()

Retrieves the NULL-terminated attribute name in attribute declaration XML_EVENT_ATTLIST_DECLARATION.

Syntax

```
oratext *XmlEvGetAttrDeclName0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute

Returns

(oratext*) the attribute name

XmlEvGetAttrDeclPrefix()

Retrieves the attribute prefix in attribute declaration XML_EVENT_ATTLIST_DECLARATION. Also, provides the length as an OUT len parameter.

Syntax

```
oratext *XmlEvGetAttrDeclPrefix(
   xmlevctx *evctx,
   ub4 index
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute
len	OUT	the length

Returns

`(oratext*)` the attribute prefix

XmlEvGetAttrDeclPrefix0()

Retrieves the `NULL`-terminated attribute prefix in attribute declaration `XML_EVENT_ATTLIST_DECLARATION`.

Syntax

```
oratext *XmlEvGetAttrDeclPrefix0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute

Returns

`(oratext*)` the attribute prefix

XmlEvGetAttrID()

Retrieves the ID for the attribute's `QNAME`, for `XML_EVENT_START_ELEMENT` events. Invokes the user-supplied ID callback specified in XmlEvCreatePPCtx(); if the callback is not specified, returns `0`.

Syntax

```
sb8 XmlEvGetAttrID(
   xmlevctx *evctx
   ub4 index);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of attribute

Returns

`(sb8)` the ID

XmlEvGetAttrLocalName()

Retrieves the attribute local name for the `XML_EVENT_START_ELEMENT` events. Also, provides the length as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetAttrLocalName(
   xmlevctx *evctx,
   ub4 index
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute; ignored for `XML_EVENT_START_ATTR`
`len`	`OUT`	the length

Returns

`(oratext*)` the attribute name

XmlEvGetAttrLocalName0()

Retrieves the `NULL`-terminated attribute local name for the `XML_EVENT_START_ELEMENT` events.

Syntax

```
oratext *XmlEvGetAttrLocalName0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute; ignored for `XML_EVENT_START_ATTR`

Returns

`(oratext*)` the attribute name

XmlEvGetAttrName()

Retrieves the attribute name for the `XML_EVENT_START_ELEMENT` events. Also, provides the length as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetAttrName(
   xmlevctx *evctx,
   ub4 index
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute; ignored for XML_EVENT_START_ATTR
len	OUT	the length

Returns

(oratext*) the attribute name

XmlEvGetAttrName0()

Retrieves the NULL-terminated attribute name for the XML_EVENT_START_ELEMENT events.

Syntax

```
oratext *XmlEvGetAttrName0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute; ignored for XML_EVENT_START_ATTR

Returns

(oratext*) the attribute name

XmlEvGetAttrPrefix()

Retrieves the prefix tag for XML_EVENT_START_ELEMENT events, and also returns the length of the event as an OUT len parameter.

Syntax

```
oratext *XmlEvGetAttrPrefix(
   xmlevctx *evctx,
   ub4 index,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute
len	OUT	length of the event name

Returns

(oratext*) the attribute prefix

XmlEvGetAttrPrefix0()

Retrieves the NULL-terminated attribute prefix for the XML_EVENT_START_ELEMENT events.

Syntax

```
oratext *XmlEvGetAttrPrefix0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute

Returns

(oratext*) the attribute prefix

XmlEvGetAttrURI()

Retrieves the attribute URI for the XML_EVENT_START_ELEMENT events. Also, provides the length as an OUT len parameter.

Syntax

```
oratext *XmlEvGetAttrURI(
   xmlevctx *evctx,
   ub4 index
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute

Parameter	In/Out	Description
len	OUT	the length

Returns

(oratext*) the attribute URI

XmlEvGetAttrURI0()

Retrieves the NULL-terminated attribute URI for the XML_EVENT_START_ELEMENT events.

Syntax

```
oratext *XmlEvGetAttrURI0(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of the attribute

Returns

(oratext*) the attribute URI

XmlEvGetAttrUriID()

Retrieves the ID for the attribute's URI, for XML_EVENT_START_ELEMENT events. Invokes the user-supplied ID callback specified in XmlEvCreatePPCtx(); if the callback is not specified, returns 0.

Syntax

```
sb8 XmlEvGetAttrUriID(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
index	IN	index of attribute

Returns

(sb8) the ID

XmlEvGetAttrValue()

Retrieves the attribute value for one of the `XML_EVENT_START_ELEMENT` events, and also returns the length of the event as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetAttrValue(
   xmlevctx *evctx,
   ub4 index,
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute
`len`	`OUT`	length of the event name

Returns

`(oratext*)` the attribute value

XmlEvGetAttrValue0()

Retrieves the `NULL`-terminated attribute value for the `XML_EVENT_START_ELEMENT` events.

Syntax

```
oratext *XmlEvGetAttrValue0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute

Returns

`(oratext*)` the attribute value

XmlEvGetElDeclContent()

Retrieves the element declaration content for `XML_EVENT_ELEMENT_DECLARATION`. Also, provides the length as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetElDeclContent(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
len	OUT	the length

Returns

(oratext*) the declaration content

XmlEvGetElDeclContent0()

Retrieves the element declaration content for XML_EVENT_ELEMENT_DECLARATION.

Syntax

```
oratext *XmlEvGetElDeclContent0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) the declaration content

XmlEvGetEncoding()

Returns the value of the encoding specified in either XmlEvCreatePPCtx() call or XmlEvCreateSVCtx() call.

Syntax

```
oratext *XmlEvGetEncoding(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) the encoding value in out-encoding; NULL if no encoding is specified

XmlEvGetError()

Retrieves the error number when the `XML_EVENT_FATAL_ERROR` or `XML_EVENT_ERROR` event is returned by a XmlEvNext() call.

Syntax

```
xmlerr XmlEvGetError(
   xmlevctx *evctx
   oratext **message);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`message`	`IN`	the error message

Returns

`(xmlerr)` the error number

XmlEvGetName()

Retrurns the name of the events, and the length of the event in the `OUT len` parameter. The event name could be on of the following:

- `XML_EVENT_START_ELEMENT`
- `XML_EVENT_END_ELEMENT`
- `XML_EVENT_START_ENTITY`
- `XML_EVENT_ENTITY_REFERENCE`
- `XML_EVENT_ELEMENT_DECLARATION`
- `XML_EVENT_PE_DECLARATION`
- `XML_EVENT_UE_DECLARATION`
- `XML_EVENT_NOTATTION_DECLARATION`

Syntax

```
oratext *XmlEvGetName(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`len`	`OUT`	length of the name

Returns

`(oratext*)` The name

XmlEvGetName0()

Retrieves a `NULL`-terminated name for one of the following events:

- `XML_EVENT_START_ELEMENT`
- `XML_EVENT_END_ELEMENT`
- `XML_EVENT_START_ENTITY`
- `XML_EVENT_ENTITY_REFERENCE`
- `XML_EVENT_ELEMENT_DECLARATION`
- `XML_EVENT_PE_DECLARATION`
- `XML_EVENT_UE_DECLARATION`
- `XML_EVENT_NOTATTION_DECLARATION`

Syntax

```
oratext *XmlEventGetName0(
   xmleventctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(oratext*)` The name

XmlEvGetLocalName()

Retrieves the local name tag for one of the following events, and also returns the length of the event as an `OUT len` parameter:

- `XML_EVENT_START_ELEMENT`
- `XML_EVENT_END_ELEMENT`

Syntax

```
oratext *XmlEvGetLocalName(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Parameter	In/Out	Description
len	OUT	length of the event name

Returns

(oratext*) local name tag

XmlEvGetLocalName0()

Retrieves the NULL-terminated local name tag for one of the following events:

- XML_EVENT_START_ELEMENT
- XML_EVENT_END_ELEMENT

Syntax

```
oratext *XmlEvGetLocalName0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) local name tag

XmlEvGetLocation()

Retrieves the location during parsing, as OUT parameters for the line number of the input stream and its path. Can be used at any time during the parsing processes.

Syntax

```
void *XmlEvGetLocation(
   xmlevctx *evctx,
   ub4 *line,
   oratext **path);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
line	OUT	line number
path	OUT	URL or file name

XmlEvGetPIData()

Retrieves the text for one of the following events, and also returns the length of the event as an `OUT len` parameter:

- `XML_EVENT_PI`
- `XML_EVENT_PI_CONT`

Syntax

```
oratext *XmlEvGetPIData(
   xmlevctx *evctx
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`len`	`OUT`	length of the event name

Returns

`(oratext*)` data

XmlEvGetPIData0()

Retrieves the `NULL`-terminated data for one of the following events:

- `XML_EVENT_PI`
- `XML_EVENT_PI_CONT`

Syntax

```
oratext *XmlEvGetPIData0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(oratext*)` data

XmlEvGetPITarget()

Retrieves the target for one of the following events, and also returns the length of the event as an `OUT len` parameter:

- `XML_EVENT_PI`

- XML_EVENT_PI_CONT

Syntax

```
oratext *XmlEvGetPITarget(
   xmlevctx *evctx
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
len	OUT	length of the event name

Returns

(oratext*) target

XmlEvGetPITarget0()

Retrieves the NULL-terminated target for one of the following events:

- XML_EVENT_PI
- XML_EVENT_PI_CONT

Syntax

```
oratext *XmlEvGetPITarget0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) target

XmlEvGetPEIsGen()

Determines if the general entity was declared, XML_EVENT_PE_DECLARATION.

Syntax

```
boolean XmlEvGetPEIsGen(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

`TRUE` for a general entity, `FALSE` if a parameter

XmlEvGetPERepl()

Retrieves the replacement text of PE declaration, `XML_EVENT_PE_DECLARATION`. Also, provides the length as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetPERepl(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`len`	`OUT`	the length

Returns

`(oratext*)` PE replacement text

XmlEvGetPERepl0()

Retrieves the `NULL`-terminated replacement text of PE declaration, `XML_EVENT_PE_DECLARATION`.

Syntax

```
oratext *XmlEvGetPERepl0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(oratext*)` PE replacement text

XmlEvGetPrefix()

Retrieves the prefix tag for one of the following events, and also returns the length of the event as an `OUT len` parameter:

- `XML_EVENT_START_ELEMENT`
- `XML_EVENT_END_ELEMENT`

Syntax

```
oratext *XmlEvGetPrefix(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
len	OUT	length of the prefix

Returns

(oratext*) the prefix tag

XmlEvGetPrefix0()

Retrieves the NULL-terminated prefix tag for one of the following events:

- XML_EVENT_START_ELEMENT
- XML_EVENT_END_ELEMENT

Syntax

```
oratext *XmlEvGetPrefix0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) the prefix tag

XmlEvGetPubId()

Retrieves the public id for one of the following events; also, provides the length as an OUT len parameter:

- XML_EVENT_PE_DECLARATION
- XML_EVENT_UE_DECLARATION
- XML_EVENT_NOTATION_DECLARATION

Syntax

```
oratext *XmlEvGetPubId(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
len	OUT	the length

Returns

(oratext*) public id

XmlEvGetPubId0()

Retrieves the NULL-terminated public id for one of the following events:

- XML_EVENT_PE_DECLARATION
- XML_EVENT_UE_DECLARATION
- XML_EVENT_NOTATION_DECLARATION

Syntax

```
oratext *XmlEvGetPubId0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returnsb

(oratext*) public id

XmlEvGetSysId()

Retrieves the system id for one of the following events; also, provides the length as an OUT len parameter:

- XML_EVENT_PE_DECLARATION
- XML_EVENT_UE_DECLARATION
- XML_EVENT_NOTATION_DECLARATION

Syntax

```
oratext *XmlEvGetSysId(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
evctx	IN	XML Event context
len	OUT	the length

Returns

(oratext*) system id

XmlEvGetSysId0()

Retrieves the NULL-terminated system id for one of the following events:

- XML_EVENT_PE_DECLARATION
- XML_EVENT_UE_DECLARATION
- XML_EVENT_NOTATION_DECLARATION

Syntax

```
oratext *XmlEvGetSysId0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) system id

XmlEvGetTagID()

Retrieves the ID for the tag's QNAME, for XML_EVENT_START_ELEMENT events. Invokes the user-supplied ID callback specified in XmlEvCreatePPCtx(); if the callback is not specified, returns 0.

Syntax

```
sb8 XmlEvGetTagID(
   xmlevctx *evctx)
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(sb8) the ID

XmlEvGetTagUriID()

Retrieves the ID for the tag's URI, for `XML_EVENT_START_ELEMENT` and `XML_EVENT_END_ELEMENT` events. Invokes the user-supplied ID callback specified in XmlEvCreatePPCtx(); if the callback is not specified, returns `0`.

Syntax

```
sb8 XmlEvGetTagUriID(
   xmlevctx *evctx)
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(sb8)` the ID

XmlEvGetText()

Retrieves the text for one of the following events, and also returns the length of the event as an `OUT len` parameter:

- `XML_EVENT_CHARACTERS`
- `XML_EVENT_CHARACTERS_CONT`
- `XML_EVENT_SPACE`
- `XML_EVENT_SPACE_CONT`
- `XML_EVENT_COMMENT`
- `XML_EVENT_COMMENT_CONT`
- `XML_EVENT_CDATA`
- `XML_EVENT_CDATA_CONT`

Syntax

```
oratext *XmlEvGetText(
   xmlevctx *evctx
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`len`	`OUT`	length of the event name

Returns

`(oratext*)` event text

XmlEvGetText0()

Retrieves the `NULL`-terminated text for one of the following events:

- `XML_EVENT_CHARACTERS`
- `XML_EVENT_CHARACTERS_CONT`
- `XML_EVENT_SPACE`
- `XML_EVENT_SPACE_CONT`
- `XML_EVENT_COMMENT`
- `XML_EVENT_COMMENT_CONT`
- `XML_EVENT_CDATA`
- `XML_EVENT_CDATA_CONT`

Syntax

```
oratext *XmlEvGetText0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(oratext*)` event text

XmlEvGetUENdata()

Retrieves the ndata for `XML_EVENT_UE_DECLARATION` event, and also returns the length of the event as an `OUT len` parameter.

Syntax

```
oratext *XmlEvGetUENdata(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`len`	`OUT`	length of the event name

Returns

`(oratext*)` ndata

XmlEvGetUENdata0()

Retrieves the `NULL`-terminated ndata for `XML_EVENT_UE_DECLARATION` event.

Syntax

```
oratext *XmlEvGetUENdata0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context

Returns

`(oratext*)` ndata

XmlEvGetURI()

Retrieves the URI tag for one of the following events, and also returns the length of the event as an `OUT len` parameter:

- `XML_EVENT_START_ELEMENT`
- `XML_EVENT_END_ELEMENT`

Syntax

```
oratext *XmlEvGetURI(
   xmlevctx *evctx,
   ub4 *len);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`len`	`OUT`	length of the event name

Returns

`(oratext*)` URI tag

XmlEvGetURI0()

Retrieves the `NULL`-terminated URI tag for one of the following events:

- `XML_EVENT_START_ELEMENT`

- XML_EVENT_END_ELEMENT

Syntax

```
oratext *XmlEvGetURI0(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) URI tag

XmlEvGetVersion()

Provides information about version specification in XML declaration for the XML_EVENT_START_DOCUMENT event.

Syntax

```
oratext *XmlEvGetVersion(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(oratext*) version string from the XML declaration.

XmlEvIsEncodingSpecified()

Provides information about encoding specification in XML declaration for the XML_EVENT_START_DOCUMENT event.

Syntax

```
boolean XmlEvIsEncodingSpecified(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

TRUE if encoding was specified in XML declaration, FALSE otherwise

XmlEvIsNamespaceAttr()

Determines if an attribute is a namespace atrribute for `XML_EVENT_START_ELEMENT` event.

Syntax

```
boolean XmlEvIsNamespaceAttr(
   xmlevctx *evctx,
   ub4 index);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Event context
`index`	`IN`	index of the attribute

Returns

`TRUE` if an attribute is a namespace attribute, `FALSE` otherwise

XmlEvIsStandalone()

Provides information about standalone specification in XML declaration for the `XML_EVENT_START_DOCUMENT` event.

Syntax

```
sword  XmlEvIsStandalone(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
`evctx`	`IN`	XML Events contextt

Returns

`(sword) -1` if standalone was not specified in the XML declaration, `0` if `FALSE` was specified for standalone, and `1` if `TRUE` was specified for standalone

XmlEvNext()

Gets the next event; advances the parser.

Syntax

```
xmlevtype XmlEvNext(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(xmlevtype) the event

XmlEvNextTag()

Advances the parser to the next tag event, such as XML_EVENT_START_ELEMENT, XML_EVENT_END_ELEMENT, and XML_EVENT_END_DOCUMENT.

Syntax

```
xmlevtype XmlEvNextTag(
   xmlevctx *evctx);
```

Parameter	In/Out	Description
evctx	IN	XML Event context

Returns

(xmlevtype) the event

XmlEvLoadPPDoc()

Loads a new document and sets it up for pull parsing. Prepares to start parsing the XML document from an input source in pull-parse mode. Input sources are the same as for XmlLoadDom()XmlLoadDom() and XmlLoadSax()XmlLoadSax() of Package XML APIs for C. The actual parsing is driven by multiple calls to XmlEvNext().

Syntax

```
xmlerr XmlEvLoadPPDoc(
   xmlctx *xctx,
   xmlevctx *evctx,
   oratext *inputType,
   void *input,
   ub4 inputLen,
   oratext *inputEncoding);
```

Parameter	In/Out	Description
xctx	IN	XML context
evctx	IN	XML Events contextt
inputType	IN	type of input, such as file, buffer, uri, stream, or stdio

Parameter	In/Out	Description
input	IN	the input
inputLen	IN	input length for buffer input type
inputEncoding	IN	input encoding

Returns

(xmlerr) the error code

XmlEvSchemaValidate()

Validates XML documents represented by events. Initializes the stream validator.

This is an opaque method. An alternate approach would be to use the transparent XmlEvCreateSVCtx()XmlEvCreateSVCtx() and XmlEvDestroySVCtx()XmlEvDestroySVCtx().

Syntax

```
xmlerr XmlEvSchemaValidate(
   xmlctx *xctx,
   xsdctx *sctx,
   xmlevctx *docEvCtx,
   oratext **errmsg);
```

Parameter	In/Out	Description
xctx	IN	XML context
sctx	IN	Schema context
docEvCtx	IN	Event context for the document that is validated
errmsg	OUT	The error message that corresponds to the error code

Returns

(xmlerr) the error code

5
Package Orastream for XML C APIs

`Orastream` APIs support handling of text and binary nodes that exceed 64K in an XML document.

The data types used by `Orastream` are found in Datatypes for C; they include oracheck, oraerr, oraprop_id, oramemctx, oraprop, oraprop_t, oraprop_v, orastream, and orastreamhdl.

The error codes for the `Orastream` interfaces are described in Table 5-1.

Table 5-1 Orastream Error Codes for XML C Implementation

Error Code	Description
`ORASTREAM_ERR_NULL_POINTER`	Null pointer encountered.
`ORASTREAM_ERR_BAD_STREAM`	Invalid stream object.
`ORASTREAM_ERR_WRONG_DIRECTION`	Stream object is defined for the opposite I/O direction.
`ORASTREAM_ERR_UNKNOWN_PROPERTY`	Unknown creation property.
`ORASTREAM_ERR_NO_DIRECTION`	The I/O direction of the stream is undefined.
`ORASTREAM_ERR_BI_DIRECTION`	The stream direction is incorrectly defined as using both I/O directions.
`ORASTREAM_ERR_NOT_OPEN`	The stream is not open.
`ORASTREAM_ERR_WRONG_MODE`	The stream is defined for the opposite char/byte mode.
`ORASTREAM_ERR_CANT_OPEN`	The stream cannot be opened.
`ORASTREAM_ERR_CANT_CLOSE`	The stream cannot be closed.

The `Orastream` methods, listed in the following table, support unidirectional streams used to move data piecewise. The direction and mode of the stream is determined by the paramters that initialize the stream in the OraStreamInit() method.

Table 5-2 Summary of OraStream Methods for XML C Implementation

Function	Summary
OraStreamClose()	Closes the stream.
OraStreamHandle()	Returns the handle to the stream.
OraStreamInit()	Initializes the stream.
OraStreamIsOpen()	Determines if the stream is open.
OraStreamOpen()	Opens the stream.
OraStreamRead()	Reads bytes from the stream.
OraStreamReadable()	Determines if the stream is readable.
OraStreamReadChar()	Reads characters from the stream.

Table 5-2 (Cont.) Summary of OraStream Methods for XML C Implementation

Function	Summary
OraStreamSid()	Sets the SID of a stream.
OraStreamTerm()	Destroys the stream.
OraStreamWrite()	Writes bytes to the stream.
OraStreamWritable()	Determines if the stream is writable.
OraStreamWriteChar()	Writes characters to the stream.

> **See Also:**
>
> *Oracle XML Developer's Kit Programmer's Guide*

OraStreamClose()

Closes the `orastream` object.

The function is used to close the given stream by calling the 'close' callback function of the stream.

Returns `ORAERR_OK` for success, or the error code for failure.

Syntax

```
oraerr OraStreamClose(
  orastream *stream);
```

Parameter	In/Out	Description
`stream`	`IN`	Stream that is closed

OraStreamHandle()

Returns the handle of the `orastream` object.

The handle contains the generic pointers and file descriptors.

Syntax

```
orastreamhdl *OraStreamHandle(
  orastream *stream);
```

Parameter	In/Out	Description
`stream`	`IN`	Stream whose handle is returned

OraStreamInit()

Creates and initializes a `orastream` object.

Syntax

```
orastream *OraStreamInit(
  void *sctx,
  void *sid,
  oraerr *err,
  list);
```

Parameter	In/Out	Description
`sctx`	`IN`	The input context; may be `NULL`
`sid`	`IN`	The user-defined stream context identifier
`err`	`OUT`	The error, if any. `ORAERR_OK` for success, or the error code for failure.

`list`	`IN`	`NULL`-terminated list of name-value pairs of arguments that specify the properties of the new `orastream` oject. These are:
		• The `open` property name is for the open function, and its value follows.
		`ORASTREAM_OPEN_F((*), sctx, sid, hdl, length)`
		• The `close` property name is for the close function, and its value follows.
		`ORASTREAM_CLOSE_F((*), sctx, sid, hdl)`
		• The `read` property name is for reading byte data from the stream to the buffer. Note that `nread` returns the number of bytes actually read.
		`ORASTREAM_READ_F((*), sctx, sid, hdl, dest, size, start, nread, eoi)`
		• The `write` property name is for writing byte data from the buffer to the stream. Note that `written` returns the number of bytes actually written.
		`ORASTREAM_WRITE_F((*), sctx, sid, hdl, src, size, written)`
		• The `read_char` property name is for reading character data from the stream to the buffer. Note that `nread` returns the number of characters actually read.
		`ORASTREAM_READ_F((*), sctx, sid, hdl, dest, size, start, nread, eoi)`
		• The `write_char` property name is for writing character data from the buffer to the stream. Note that `written` returns the number of characters actually written.
		`ORASTREAM_WRITE_F((*), sctx, sid, hdl, src, size, written)`

OraStreamIsOpen()

Determines if the `orastream` is open. Returns `TRUE` or `FALSE`.

Note that the stream must be open to perform read and write operations.

Syntax

```
boolean OraStreamIsOpen(
  orastream *stream);
```

Parameter	In/Out	Description
`stream`	`IN`	The stream that should be open for reads or writes.

OraStreamOpen()

Opens the `orastream` object.

The function opens the stream by calling the 'open' callback function of the stream.

Returns `ORAERR_OK` for success, or the error code for failure.

Syntax

```
oraerr OraStreamOpen(
  orastream *stream,
  ubig_ora *length)
```

Parameter	In/Out	Description
`stream`	`IN`	The stream that is open
`length`	`OUT`	Optional parameter; not used

OraStreamRead()

Reads `byte`s from the `orastream` object.

The function is used to read the data from the stream into the specified buffer. It also returns `TRUE` for the `eoi` parameter if the end of stream is reached.

Returns `ORAERR_OK` for success, or the error code for failure.

Syntax

```
oraerr OraStreamRead(
  orastream *stream,
  oratext *dest,
  ubig_ora size,
  oratext **start,
```

```
  ubig_ora *nread,
  ub1 *eoi);
```

Parameter	In/Out	Description
stream	IN	Stream that is being read
dest	IN	The destination buffer
size	IN	The size of the data to be read
start	OUT	Pointer to the start of data being read
nread	OUT	Number of bytes actually read from the stream
eoi	OUT	Returns TRUE if end of the stream is reached; FALSE otherwise

OraStreamReadable()

Determines if an existing orastream object is readable.

Returns TRUE or FALSE.

Syntax

```
boolean OraStreamReadable(
  orastream *stream);
```

Parameter	In/Out	Description
stream	IN	Stream that is checked for readability

OraStreamReadChar()

Reads chars from the orastream object.

The function is used to read the data from the stream into the specified buffer. It also returns TRUE for the eoi parameter if the end of stream is reached.

Returns ORAERR_OK for success, or the error code for failure.

Syntax

```
oraerr OraStreamReadChar(
  orastream *stream,
  oratext *dest,
  ubig_ora size,
  oratext **start,
  ubig_ora *nread,
  ub1 *eoi);
```

Parameter	In/Out	Description
stream	IN	Stream that is being read
dest	IN	The destination buffer
size	IN	The size of the data to be read
start	OUT	Pointer to the start of data being read
nread	OUT	Number of characters actually read from the stream
eoi	OUT	Returns TRUE if end of the stream is reached; FALSE otherwise

OraStreamSid()

Assigns an SID to an existing orastream object. Returns the old SID through the OUT parameter osid.

Returns ORAERR_OK for success, or the error code for failure.

Syntax

```
oraerr OraStreamSid(
  orastream *stream,
  void *sid,
  void **osid);
```

Parameter	In/Out	Description
stream	IN	The stream whose SID is changed
sid	IN	The new SID
osid	OUT	The previous SID of the stream

OraStreamTerm()

Destroys a orastream object and frees its associated memory.

Returns ORAERR_OK for success, or the error code for failure.

Syntax

```
oraerr OraStreamTerm(
  orastream *stream);
```

Parameter	In/Out	Description
stream	IN	Stream that is destroyed

OraStreamWrite()

Writes bytes to the orastream object.

The number of bytes actually read are stored by the OUT parameter nwrote.

Returns ORAERR_OK for success, or the error code for failure.

Syntax

```
oraerr OraStreamWrite(
  orastream *stream,
  oratext *src,
  ubig_ora size,
  ubig_ora *nwrote);
```

Parameter	In/Out	Description
stream	IN	Stream where the data is written
src	IN	Buffer from which the data is written
size	IN	Size of data to be written
nwrote	OUT	Number of bytes written to the stream

OraStreamWritable()

Determines if an existing orastream object is writable.

Returns TRUE or FALSE.

Syntax

```
boolean OraStreamWritable(
  orastream *stream);
```

Parameter	In/Out	Description
stream	IN	Stream that is checked for writability.

OraStreamWriteChar()

Writes chars to the orastream object.

The number of characters actually written are stored by the `OUT` parameter `nwrote`.

Returns `ORAERR_OK` for success, or the error code for failure.

Syntax

```
oraerr OraStreamWriteChar(
  orastream *stream,
  oratext *src,
  ubig_ora size,
  ubig_ora *nwrote);
```

Parameter	In/Out	Description
`stream`	`IN`	Stream where the data is written
`src`	`IN`	Buffer from which the data is written
`size`	`IN`	Size of data to be written
nwrote	`OUT`	Number of characters written to the stream

6
Package Range for XML C APIs

The following table summarizes the methods available through the `Range` interface for XML C APIs.

Table 6-1 Summary of Range Methods for XML C Implementation

Function	Summary
XmlDomCreateRange()	Create `Range` object.
XmlDomRangeClone()	Clone a range.
XmlDomRangeCloneContents()	Clone contents selected by a range.
XmlDomRangeCollapse()	Collapse range to either start point or end point.
XmlDomRangeCompareBoundaryPoints()	Compare boundary points of two ranges.
XmlDomRangeDeleteContents()	Delete content selected by a range.
XmlDomRangeDetach()	Detach a range.
XmlDomRangeExtractContents()	Extract contents selected by a range.
XmlDomRangeGetCollapsed()	Return whether the range is collapsed.
XmlDomRangeGetCommonAncestor()	Return deepest common ancestor node of two boundary points.
XmlDomRangeGetDetached()	Return whether the range is detached.
XmlDomRangeGetEndContainer()	Return range end container node.
XmlDomRangeGetEndOffset()	Return range end offset.
XmlDomRangeGetStartContainer()	Return range start container node.
XmlDomRangeGetStartOffset()	Return range start offset.
XmlDomRangeIsConsistent()	Return whether the range is consistent.
XmlDomRangeSelectNode()	Select a node as a range.
XmlDomRangeSelectNodeContents()	Define range to select node contents.
XmlDomRangeSetEnd()	Set the end point.
XmlDomRangeSetEndBefore()	Set the end point before a node.
XmlDomRangeSetStart()	Set the start point.
XmlDomRangeSetStartAfter()	Set the start point after a node.
XmlDomRangeSetStartBefore()	Set the start point before a node.

XmlDomCreateRange()

The only one method of `DocumentRange` interface, used to create a `Range` object.

Syntax

```
xmlrange* XmlDomCreateRange(
   xmlctx *xctx,
   xmlrange *range,
   xmldocnode *doc);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	existing NodeIterator, or NULL to allocate new
doc	IN	document to which the new Range is attached

Returns

(xmlrange *) original or new Range object.

XmlDomRangeClone()

Clone a Range. Clones the range without affecting the content selected by the original range. Returns NULL if an error.

Syntax

```
xmlrange* XmlDomRangeClone(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(xmlrange *) new range that clones the old one

XmlDomRangeCloneContents()

Clone contents selected by a range. Clones but does not delete contents selected by a range. Performs the range consistency check and sets retval to an error code if an error.

Syntax

```
xmlnode* XmlDomRangeCloneContents(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(xmlnode *) cloned contents

XmlDomRangeCollapse()

Collapses the range to either start point or end point. The point where it is collapsed to is assumed to be a valid point in the document which this range is attached to.

Syntax

```
xmlerr XmlDomRangeCollapse(
   xmlctx *xctx,
   xmlrange *range,
   boolean tostart);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
tostart	IN	indicates whether to collapse to start (TRUE) or to end (FALSE)

Returns

(xmlerr) numeric return code

XmlDomRangeCompareBoundaryPoints()

Compares two boundary points of two different ranges. Returns -1,0,1 depending on whether the corresponding boundary point of the range (range) is before, equal, or after the corresponding boundary point of the second range (srange). It returns ~(int)0 if two ranges are attached to two different documents or if one of them is detached.

Syntax

```
sb4 XmlDomRangeCompareBoundaryPoints(
   xmlctx *xctx,
   xmlrange *range,
   xmlcmphow how,
   xmlrange *srange,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
how	IN	xmlcmphow value; how to compare
srange	IN	range object with which to compare
xerr	OUT	numeric return code

Returns

(sb4) strcmp-like comparison result

XmlDomRangeDeleteContents()

Deletes content selected by a range. Performs the range consistency check and sets retval to an error code if an error.

Syntax

```
xmlerr XmlDomRangeDeleteContents(
   xmlctx *xctx,
   xmlrange *range);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object

Returns

(xmlerr) numeric return code

XmlDomRangeDetach()

Detaches the range from the document and places it (range) in invalid state.

Syntax

```
xmlerr XmlDomRangeDetach(
   xmlctx *xctx,
   xmlrange *range);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object

Returns

(xmlerr) numeric return code

XmlDomRangeExtractContents()

Extract contents selected by a range. Clones and deletes contents selected by a range. Performs the range consistency check and sets retval to an error code if an error.

Syntax

```
xmlnode* XmlDomRangeExtractContents(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(xmlnode *) extracted

XmlDomRangeGetCollapsed()

Returns TRUE if the range is collapsed and is not detached, otherwise returns FALSE.

Syntax

```
boolean XmlDomRangeGetCollapsed(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(boolean) TRUE if the range is collapsed, FALSE otherwise

XmlDomRangeGetCommonAncestor()

Returns deepest common ancestor node of two boundary points of the range if the range is not detached, otherwise returns NULL. It is assumed that the range is in a consistent state.

Syntax

```
xmlnode* XmlDomRangeGetCommonAncestor(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(xmlnode *) deepest common ancestor node [or NULL]

XmlDomRangeGetDetached()

Return whether the range is detached. Returns TRUE if the range is detached and is not NULL. Otherwise returns FALSE.

Syntax

```
ub1 XmlDomRangeGetDetached(
   xmlctx *xctx,
   xmlrange *range);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object

Returns

(ub1) TRUE if the range is detached, FALSE otherwise

XmlDomRangeGetEndContainer()

Returns range end container node if the range is not detached, otherwise returns NULL.

Syntax

```
xmlnode* XmlDomRangeGetEndContainer(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(xmlnode *) range end container node [or NULL]

XmlDomRangeGetEndOffset()

Returns range end offset if the range is not detached, otherwise returns ~(ub4)0 [the maximum ub4 value].

Syntax

```
ub4 XmlDomRangeGetEndOffset(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
range	IN	range object
xerr	OUT	numeric return code

Returns

(ub4) range end offset [or ub4 maximum]

XmlDomRangeGetStartContainer()

Returns range start container node if the range is valid and is not detached, otherwise returns NULL.

Syntax

```
xmlnode* XmlDomRangeGetStartContainer(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(xmlnode *) range start container node

XmlDomRangeGetStartOffset()

Returns range start offset if the range is not detached, otherwise returns ~(ub4)0 [the maximum ub4 value].

Syntax

```
ub4 XmlDomRangeGetStartOffset(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context

Parameter	In/Out	Description
range	IN	range object
xerr	OUT	numeric return code

Returns

(ub4) range start offset [or ub4 maximum]

XmlDomRangeIsConsistent()

Return whether the range is consistent. Returns TRUE if the range is consistent: both points are under the same root and the start point is before or equal to the end point. Otherwise returns FALSE.

Syntax

```
boolean XmlDomRangeIsConsistent(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
xerr	OUT	numeric return code

Returns

(ub1) TRUE if the range is consistent, FALSE otherwise

XmlDomRangeSelectNode()

Sets the range end point and start point so that the parent node of this node becomes the container node, and the offset is the offset of this node among the children of its parent. The range becomes collapsed. It is assumed that the node is a valid node of its document. If the range is detached, it is ignored, and the range becomes attached.

Syntax

```
xmlerr XmlDomRangeSelectNode(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
node	IN	XML node

Returns

(xmlerr) numeric return code

XmlDomRangeSelectNodeContents()

Sets the range start point to the start of the node contents and the end point to the end of the node contents. It is assumed that the node is a valid document node. If the range is detached, it is ignored, and the range becomes attached.

Syntax

```
xmlerr XmlDomRangeSelectNodeContents(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
node	IN	XML node

Returns

(xmlerr) numeric return code

XmlDomRangeSetEnd()

Sets the range end point. If it has a root container other than the current one for the range, the range is collapsed to the new position. If the end is set to be at a position before the start, the range is collapsed to that position. Returns xmlerr value. according to the description where this type is defined. It is assumed that the start point of the range is a valid start point.

Syntax

```
xmlerr XmlDomRangeSetEnd(
   xmlctx *xctx,
   xmlrange *range,
```

```
   xmlnode *node,
   ub4 offset);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
node	IN	XML node
offset	IN	ending offset

Returns

(xmlerr) numeric return code

XmlDomRangeSetEndBefore()

Sets the range end point before a node. If it has a root container other than the current one for the range, the range is collapsed to the new position. If the before node sets the end to be at a position before the start, the range is collapsed to new position. Returns xmlerr value according to the description where this type is defined. It is assumed that the start point of the range is a valid start point.

Syntax

```
xmlerr XmlDomRangeSetEndBefore(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
node	IN	XML node

Returns

(xmlerr) numeric return code

XmlDomRangeSetStart()

Sets the range start point. If it has a root container other than the current one for the range, the range is collapsed to the new position. If the start is set to be at a position after the end, the range is collapsed to that position. Returns xmlerr value according to the description where this type is defined. It is assumed that the end point of the range is a valid end point.

Syntax

```
xmlerr XmlDomRangeSetStart(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node,
   ub4 offset);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
node	IN	XML node
offset	IN	starting offset

Returns

(xmlerr) numeric return code

XmlDomRangeSetStartAfter()

Sets the range start point after a node. If it has a root container other than the current one, the range is collapsed to the new position. If the previous node sets the start after the end, the range is collapsed to a new position. It is assumed that the end point of the range is a valid end point.

Syntax

```
xmlerr XmlDomRangeSetStartAfter(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

Parameter	In/Out	Description
xctx	IN	XML context
range	IN	range object
node	IN	XML node

Returns

(xmlerr) numeric return code

XmlDomRangeSetStartBefore()

Sets the range start point before a node. If it has a root container other than the current one, the range is collapsed to the new position with offset `0`. If the previous node sets the start after the end, the range is collapsed to a new position. It is assumed that the end point of the range is a valid end point.

Syntax

```
xmlerr XmlDomRangeSetStartBefore(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`range`	`IN`	range object
`node`	`IN`	XML node

Returns

`(xmlerr)` numeric return code

7

Package SAX for XML C APIs

SAX is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list. To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to one of the `XmlLoadSax` calls. A pointer to a user-defined context structure is also provided, and will be passed to each SAX function.

For event-based schema validation APIs, refer to Package Event APIs for C.

The following table summarizes the methods available through the `SAX` interface for XML C APIs.

Table 7-1 Summary of SAX Methods for XML C Implementation

Function	Summary
XmlSaxAttributeDecl()	Receives SAX notification of an attribute's declaration.
XmlSaxCDATA()	Receives SAX notification of `CDATA`. Oracle extension.
XmlSaxCharacters()	Receives SAX notification of character data
XmlSaxComment()	Receives SAX notification of a comment.
XmlSaxElementDecl()	Receives SAX notification of an element's declaration. Oracle extension.
XmlSaxEndDocument()	Receives SAX end-of-document notification.
XmlSaxEndElement()	Receives SAX end-of-element notification.
XmlSaxNotationDecl()	Receives SAX notification of a notation declaration.
XmlSaxPI()	Receives SAX notification of a processing instruction.
XmlSaxParsedEntityDecl()	Receives SAX notification of a parsed entity declaration. Oracle extension.
XmlSaxStartDocument()	Receives SAX start-of-document notification.
XmlSaxStartElement()	Receives SAX start-of-element notification.
XmlSaxStartElementNS()	Receives SAX namespace-aware start-of-element notification.
XmlSaxUnparsedEntityDecl()	Receives SAX notification of an unparsed entity declaration.
XmlSaxWhitespace()	Receives SAX notification of ignorable (whitespace) data.
XmlSaxXmlDecl()	Receives SAX notification of an XML declaration. Oracle extension.

XmlSaxAttributeDecl()

This event marks an element declaration in the DTD. The element's name and content will be in the data encoding. Note that an attribute may be declared before the element it belongs to!

Syntax

```
xmlerr XmlSaxAttributeDecl(
   void *ctx,
```

```
   oratext *elem,
   oratext *attr,
   oratext *body);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
elem	IN	element for which the attribute is declared; data encoding
attr	IN	attribute's name; data encoding
body	IN	body of an attribute declaration

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxAttributeDecl()

XmlSaxBeginGen()

This call creates an XML generation context from the specified XML meta context. This context is uded by the SAX calls to generate an incremental XML document.

Syntax

```
void *XmlSaxBeginGen(
   XmlCtx *xctx);
```

Parameter	In/Out	Description
xctx	IN	the XML meta contextt

XmlSaxCDATA()

This event handles CDATA, as distinct from Text. If no XmlSaxCDATA callback is provided, the Text callback will be invoked. The data will be in the data encoding, and the returned length is in characters, not bytes. See also XmlSaxWhitespace, which receiving notification about ignorable (whitespace formatting) character data.

Syntax

```
xmlerr XmlSaxCDATA(
   void *ctx,
   oratext *ch,
   size_t len);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
ch	IN	pointer to CDATA; data encoding
len	IN	length of CDATA, in characters

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxWhitespace()

XmlSaxCharacters()

This event marks character data, either Text or CDATA. If an XmlSaxCDATA callback is provided, then CDATA will be send to that instead; with no XmlSaxCDATA callback, both Text and CDATA go to the XmlSaxCharacters callback. The data will be in the data encoding, and the returned length is in characters, not bytes. See also XmlSaxWhitespace, which receiving notification about ignorable (whitespace formatting) character data.

Syntax

```
xmlerr XmlSaxCharacters(
   void *ctx,
   oratext *ch,
   size_t len);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
ch	IN	pointer to data; data encoding
len	IN	length of data, in characters

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxWhitespace()

XmlSaxComment()

This event marks a comment in the XML document. The comment's data will be in the data encoding. Oracle extension, not in SAX standard.

Syntax

```
xmlerr XmlSaxComment(
   void *ctx,
   oratext *data);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
data	IN	comment's data; data encoding

Returns

(xmlerr) error code, XMLERR_OK [0] for success

XmlSaxElementDecl()

This event marks an element declaration in the DTD. The element's name and content will be in the data encoding.

Syntax

```
xmlerr XmlSaxElementDecl(
   void *ctx,
   oratext *name,
   oratext *content);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
name	IN	element's name
content	IN	element's context model

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxAttributeDecl()

XmlSaxEndDocument()

The last SAX event, called once for each document, indicating the end of the document. Matching event is `XmlSaxStartDocument`.

Syntax

```
xmlerr XmlSaxEndDocument(
   void *ctx);
```

Parameter	In/Out	Description
`ctx`	`IN`	user's SAX context

Returns

`(xmlerr)` error code, `XMLERR_OK [0]` for success

> **See Also:**
>
> XmlSaxStartDocument()

XmlSaxEndElement()

This event marks the close of an element; it matches the `XmlSaxStartElement` or `XmlSaxStartElementNS` events. The name is the `tagName` of the element (which may be a qualified name for namespace-aware elements) and is in the data encoding.

XmlSaxEndGen()

Completes the generation of an incrementally constructed XML document, and returns it to the application.

Syntax

```
orastream *XmlSaxEndGen(
   void *genCtx);
```

Parameter	In/Out	Description
`genCtx`	`IN`	the generation context used to create the XML document; see XmlSaxBeginGen()XmlSaxBeginGen()t

Return

`(orastream*)` the stream containing the generated XML document

XmlSaxNotationDecl()

The even marks the declaration of a notation in the DTD. The notation's name, public ID, and system ID will all be in the data encoding. Both IDs are optional and may be `NULL`.

XmlSaxPI()

This event marks a `ProcessingInstruction`. The `ProcessingInstruction`s target and data will be in the data encoding. There is always a target, but the data may be `NULL`.

Syntax

```
xmlerr XmlSaxPI(
   void *ctx,
   oratext *target,
   oratext *data);
```

Parameter	In/Out	Description
`ctx`	`IN`	user's SAX context
`target`	`IN`	PI's target; data encoding
`data`	`IN`	PI's data as data encoding, or `NULL`

Returns

`(xmlerr)` error code, `XMLERR_OK [0]` for success

XmlSaxParsedEntityDecl()

Marks an parsed entity declaration in the DTD. The parsed entity's name, public ID, system ID, and notation name will all be in the data encoding.

Syntax

```
xmlerr XmlSaxParsedEntityDecl(
   void *ctx,
   oratext *name,
   oratext *value,
   oratext *pubId,
   oratext *sysId,
   boolean general);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
name	IN	entity's name; data encoding
value	IN	entity's value; data encoding
pubId	IN	entity's public ID as data encoding, or NULL
sysId	IN	entity's system ID; data encoding
general	IN	TRUE if general entity, FALSE if parameter entity

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxUnparsedEntityDecl()

XmlSaxStartDocument()

The first SAX event, called once for each document, indicating the start of the document. Matching event is XmlSaxEndDocument.

Syntax

```
xmlerr XmlSaxStartDocument(
   void *ctx);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxEndDocument()

XmlSaxStartElement()

This event marks the start of an element. Note this is the original SAX 1 non-namespace-aware version; `XmlSaxStartElementNS` is the SAX 2 namespace-aware version. If both are registered, only the NS version will be called. The element's name will be in the data encoding, as are all the attribute parts. See the functions in the `NamedNodeMap` interface for operating on the attributes map. The matching function is `XmlSaxEndElement` (there is no namespace aware version of this function).

Syntax

```
xmlerr XmlSaxStartElement(
   void *ctx,
   oratext *name,
   xmlnodelist *attrs);
```

Parameter	In/Out	Description
`ctx`	`IN`	user's SAX context
`name`	`IN`	element's name; data encoding
`attrs`	`IN`	`NamedNodeMap` of element's attributes

Returns

`(xmlerr)` error code, `XMLERR_OK [0]` for success

> **See Also:**
>
> XmlSaxEndElement(), XmlDomGetNodeMapLength() and XmlDomRemoveNamedItem() in Package DOM APIs for C

XmlSaxStartElementNS()

This event marks the start of an element. Note this is the new SAX 2 namespace-aware version; XmlSaxStartElement is the SAX 1 non-namespace-aware version. If both are registered, only the NS version will be called. The element's qualified name, local name, and namespace URI will be in the data encoding, as are all the attribute parts. See the functions in the `NamedNodeMap` interface for operating on the attributes map. The matching function is `XmlSaxEndElement` (there is no namespace aware version of this function).

Syntax

```
xmlerr XmlSaxStartElementNS(
   void *ctx,
   oratext *qname,
   oratext *local,
```

```
   oratext *nsp,
   xmlnodelist *attrs);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
qname	IN	element's qualified name; data encoding
local	IN	element's namespace local name; data encoding
nsp	IN	element's namespace URI; data encoding
attrs	IN	NodeList of element's attributes, or NULL

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxStartElement(), XmlSaxEndElement(), XmlDomGetNodeMapLength() and XmlDomRemoveNamedItem() in Package DOM APIs for C

XmlSaxUnparsedEntityDecl()

Marks an unparsed entity declaration in the DTD, see XmlSaxParsedEntityDecl for the parsed entity version. The unparsed entity's name, public ID, system ID, and notation name will all be in the data encoding.

Syntax

```
xmlerr XmlSaxUnparsedEntityDecl(
   void *ctx,
   oratext *name,
   oratext *pubId,
   oratext *sysId,
   oratext *note);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
name	IN	entity's name; data encoding
pubId	IN	entity's public ID as data encoding, or NULL
sysId	IN	entity's system ID; data encoding

Parameter	In/Out	Description
note	IN	entity's notation name; data encoding

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxParsedEntityDecl()

XmlSaxWhitespace()

This event marks ignorable whitespace data such as newlines, and indentation between lines. The matching function is XmlSaxCharacters, which receives notification of normal character data. The data is in the data encoding, and the returned length is in characters, not bytes.

Syntax

```
xmlerr XmlSaxWhitespace(
   void *ctx,
   oratext *ch,
   size_t len);
```

Parameter	In/Out	Description
ctx	IN	user's SAX context
ch	IN	pointer to data; data encoding
len	IN	length of data, in characters

Returns

(xmlerr) error code, XMLERR_OK [0] for success

> **See Also:**
>
> XmlSaxCharacters()

XmlSaxXmlDecl()

This event marks an XML declaration. The `XmlSaxStartDocument` event is always first; if this callback is registered and an `XMLDecl` exists, it will be the second event. The encoding flag says whether an encoding was specified. Since the document's own encoding specification may be overridden (or wrong), and the input will be converted to the data encoding anyway, the actual encoding specified in the document is not provided. For the standalone flag, `-1` will be returned if it was not specified, otherwise `0` for `FALSE`, 1 for `TRUE`.

Syntax

```
xmlerr XmlSaxXmlDecl(
   void *ctx,
   oratext *version,
   boolean encoding,
   sword standalone);
```

Parameter	In/Out	Description
`ctx`	`IN`	user's SAX context
`version`	`IN`	version string from XMLDecl; data encoding
`encoding`	`IN`	whether encoding was specified
`standalone`	`IN`	value of the standalone document; `< 0` if not specified

Returns

`(xmlerr)` error code, `XMLERR_OK [0]` for success

8
Package Schema for XML C APIs

This C implementation of the XML schema validator follows the W3C XML Schema specification, rev REC-xmlschema-1-20010502. It implements the required behavior of a schema validator for multiple schema documents to be assembled into a schema. This resulting schema can be used to validate a specific instance document.

For event-based schema validation, see the methods documented in Package Event APIs for C.

The following table summarizes the methods available through the `Schema` interface for XML C APIs.

Table 8-1 Summary of Schema Methods for XML C Implementation

Function	Summary
XmlSchemaClean()	Cleans up loaded schemas in a schema context and recycle the schema context.
XmlSchemaCreate()	Creates and returns a schema context.
XmlSchemaDestroy()	Destroys a schema context.
XmlSchemaErrorWhere()	Returns the location where an error occurred.
XmlSchemaLoad()	Loads a schema document.
XmlSchemaLoadedList()	Returns the size and/or list of loaded schema documents.
XmlSchemaSetErrorHandler()	Sets an error message handler and its associated context in a schema context
XmlSchemaSetValidateOptions()	Sets option(s) to be used in the next validation session.
XmlSchemaTargetNamespace()	Returns target namespace of a given schema document.
XmlSchemaUnload()	Unloads a schema document.
XmlSchemaValidate()	Validates an element node against a schema.
XmlSchemaVersion()	Returns the version of this schema implementation.

XmlSchemaCreate()

Return a schema context to be used in other validator APIs. This needs to be paired with an `XmlSchemaDestroy`.

Syntax

```
xsdctx *XmlSchemaCreate(
   xmlctx *xctx,
   xmlerr *err,
   list);
```

Parameter	In/Out	Description
xctx	IN	XML context
err	OUT	returned error code
list	IN	NULL-terminated list of variable arguments

Returns

(xsdctx *) schema context

> **See Also:**
>
> XmlSchemaDestroy(), XmlCreate() in Package XML APIs for C

XmlSchemaDestroy()

Destroy a schema context and free up all its resources.

Syntax

```
void XmlSchemaDestroy(
   xsdctx *sctx);
```

Parameter	In/Out	Description
sctx	IN	schema context to be freed

> **See Also:**
>
> XmlSchemaCreate()

XmlSchemaErrorWhere()

Returns the location (line#, path) where an error occurred.

Syntax

```
xmlerr XmlSchemaErrorWhere(
   xsdctx *sctx,
   ub4 *line,
   oratext **path);
```

Parameter	In/Out	Description
sctx	IN	schema context
line	IN/OUT	line number where error occurred
path	IN/OUT	URL or filespace where error occurred

Returns

(xmlerr) error code

> **See Also:**
>
> XmlSchemaSetErrorHandler()

XmlSchemaLoad()

Load up a schema document to be used in the next validation session. Schema documents can be incrementally loaded into a schema context as long as every loaded schema document is valid. When the last loaded schema turns out to be invalid, you need to clean up the schema context by calling XmlSchemaClean()XmlSchemaClean() and reload everything all over again including the last schema with appropriate correction.

Given a schema document, this function converts the DOM representation into an internal schema representation. The schema document can be provided as a URI or directly a DOM representation. In the URI case, this function reads the input stream and builds a DOM representation of the schema before converting it into internal representation. In the DOM case, the application can provide a DOM representation of the schema, which will be used to create the internal schema representation.

Syntax

```
xmlerr XmlSchemaLoad(
   xsdctx *sctx,
   oratext *uri,
   list);
```

Parameter	In/Out	Description
sxctx	IN	schema context
uri	IN	URL of schema document; compiler encoding
list	IN	NULL-terminated list of variable arguments

Returns

`(xmlerr)` numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSchemaUnload(), XmlSchemaLoadedList()

XmlSchemaLoadedList()

Return only the size of loaded schema documents if `list` is `NULL`. If `list` is not `NULL`, a list of URL pointers are returned in the user-provided pointer buffer. Note that its user's responsibility to provide a buffer with big enough size.

Syntax

```
ub4 XmlSchemaLoadedList(
   xsdctx *sctx,
   oratext **list);
```

Parameter	In/Out	Description
`sctx`	`IN`	schema context
`list`	`IN`	address of pointer buffer

Returns

`(ub4)` list size

> **See Also:**
>
> XmlSchemaLoad(), XmlSchemaUnload()

XmlSchemaSetErrorHandler()

Sets an error message handler and its associated context in a schema context. To retrieve useful location information on errors, the address of the schema context must be provided in the error handler context.

Syntax

```
xmlerr XmlSchemaSetErrorHandler(
   xsdctx *sctx,
   XML_ERRMSG_F(
      (*errhdl),
      ectx,
      msg,
```

```
    err),
  void *errctx);
```

Parameter	In/Out	Description
sctx	IN	schema context
errhdl	IN	error message handler
errctx	IN	error handler context

Returns

(xmlerr) error code

> **See Also:**
>
> XmlSchemaCreate(), XmlSchemaErrorWhere(), and XML_ERRMSG_F() in Package Callback APIs for C

XmlSchemaSetValidateOptions()

Set options to be used in the next validation session. Previously set options will remain effective until they are overwritten or reset.

Syntax

```
xmlerr XmlSchemaSetValidateOptions(
   xsdctx *sctx,
   list);
```

Parameter	In/Out	Description
sctx	IN	schema context
list	IN	NULL-terminated list of variable argument

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSchemaValidate()

XmlSchemaTargetNamespace()

Return target namespace of a given schema document identified by its URI. All currently loaded schema documents can be queried. Currently loaded schema documents include the ones loaded through `XmlSchemaLoad`s and the ones loaded through `schemaLocation` or `noNamespaceSchemaLocation` hints.

Syntax

```
oratext *XmlSchemaTargetNamespace(
   xsdctx *sctx,
   oratext *uri);
```

Parameter	In/Out	Description
`sctx`	`IN`	XML context
`uri`	`IN`	URL of the schema document to be queried

Returns

`(oratext *)` target namespace string; `NULL` if given document not

> **See Also:**
>
> XmlSchemaLoadedList()

XmlSchemaUnload()

Unload a schema document from the validator. All previously loaded schema documents will remain loaded until they are unloaded. To unload all loaded schema documents, set URI to be `NULL` (this is equivalent to `XmlSchemaClean`). Note that all children schemas associated with the given schema are also unloaded. In this implementation, it only support the following scenarios:

- load, load, ...
- load, load, load, unload, unload, unload, clean, and then repeat.

It doesn't not support: load, load, unload, load, ...

Syntax

```
xmlerr XmlSchemaUnload(
   xsdctx *sctx,
   oratext *uri,
   list);
```

Parameter	In/Out	Description
sctx	IN	schema context
uri	IN	URL of the schema document; compiler encoding
list	IN	NULL-terminated list of variable argument

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSchemaLoad(), XmlSchemaLoadedList()

XmlSchemaValidate()

Validates an element node against a schema. Schemas used in current session consists of all schema documents specified through XmlSchemaLoad and provided as hint(s) through schemaLocation or noNamespaceSchemaLocation in the instance document. After the invocation of this routine, all loaded schema documents remain loaded and can be queried by XmlSchemaLoadedList. However, they will remain inactive. In the next validation session, inactive schema documents can be activated by specifying them through XmlSchemaLoad or providing them as hint(s) through schemaLocation or noNamespaceSchemaLocation in the new instance document. To unload a schema document and all its descendants (documents included or imported in a nested manner), use XmlSchemaUnload.

Syntax

```
xmlerr XmlSchemaValidate(
   xsdctx *sctx,
   xmlctx *xctx,
   xmlelemnode *elem);
```

Parameter	In/Out	Description
sctx	IN	schema context
xctx	IN	XML top-level context
elem	IN	element node in the doc, to be validated

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSchemaSetValidateOptions()

XmlSchemaVersion()

Return the version of this schema implementation.

Syntax

```
oratext *XmlSchemaVersion();
```

Returns

`(oratext *)` version string [compiler encoding]

XmlSchemaClean()

Clean up loaded schemas in a schema context and recycle the schema context.

Syntax

```
void XmlSchemaClean(
   xsdctx *sctx);
```

Parameter	In/Out	Description
`sctx`	`IN`	schema context to be cleaned

> **See Also:**
>
> XmlSchemaCreate(), XmlSchemaDestroy()

9
Package SOAP for XML C APIs

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

Attachments are not allowed in Soap 1.1. In Soap 1.2, body may not have other elements if Fault is present.

The structure of a SOAP message is illustrated in the following section of code.

```
[SOAP message (XML document)
   [SOAP envelope
      [SOAP header?
            element*
      ]
      [SOAP body
           (element* | Fault)?
      ]
   ]
]
```

The following table summarizes the methods available through the SOAP package for XML C APIs.

Table 9-1 Summary of SOAP Methods for XML C Implementation

Function	Summary
XmlSoapAddBodyElement()	Adds an element to a SOAP message body.
XmlSoapAddFaultReason()	Adds additional Reason to Fault.
XmlSoapAddFaultSubDetail()	Adds additional child to Fault Detail.
XmlSoapAddHeaderElement()	Adds an element to a SOAP header.
XmlSoapCall()	Sends a SOAP message then waits for a reply.
XmlSoapCreateConnection()	Creates a SOAP connection object.
XmlSoapCreateCtx()	Creates and returns a SOAP context.
XmlSoapCreateMsg()	Creates and returns an empty SOAP message.
XmlSoapDestroyConnection()	Destroys a SOAP connection object.
XmlSoapDestroyCtx()	Destroys a SOAP context.
XmlSoapDestroyMsg()	Destroys a SOAP message created with XmlSoapCreateMsg.
XmlSoapError()	Gets a human readable error code.
XmlSoapGetBody()	Return a SOAP message's envelope body.
XmlSoapGetBodyElement()	Gets an element from a SOAP body.
XmlSoapGetEnvelope()	Returns a SOAP part's envelope.

Table 9-1 (Cont.) Summary of SOAP Methods for XML C Implementation

Function	Summary
XmlSoapGetFault()	Returns Fault code, reason, and details.
XmlSoapGetHeader()	Returns a SOAP message's envelope header.
XmlSoapGetHeaderElement()	Gets an element from a SOAP header.
XmlSoapGetMustUnderstand()	Gets mustUnderstand attribute from SOAP header element.
XmlSoapGetReasonLang()	Gets the language of a reason with the specified index.
XmlSoapGetReasonNum()	Determines the number of reasons in Fault element.
XmlSoapGetRelay()	Gets Relay attribute from SOAP header element.
XmlSoapGetRole()	Gets role from SOAP header element.
XmlSoapHasFault()	Determines if SOAP message contains Fault object.
XmlSoapSetFault()	Sets Fault in SOAP message.
XmlSoapSetMustUnderstand()	Sets mustUnderstand attribute for SOAP header element.
XmlSoapSetRelay()	Sets Relay attribute for a SOAP header element.
XmlSoapSetRole()	Sets role for SOAP header element.

XmlSoapAddBodyElement()

Adds an element to a SOAP message body. Sets the numeric error code.

Syntax

```
xmlelemnode *XmlSoapAddBodyElement(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   oratext *qname,
   oratext *uri,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN/OUT	SOAP message
qname	IN	QName of element to add
uri	IN	Namespace URI of element to add
xerr	IN/OUT	Error code

Returns

(xmlelemnode *) created element

> **See Also:**
>
> XmlSoapAddHeaderElement()

XmlSoapAddFaultReason()

Add additional Reason to Fault. The same reason text may be provided in different languages. When the fault is created, the primary language/reason is added at that time; use this function to add additional translations of the reason.

Syntax

```
xmlerr XmlSoapAddFaultReason(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   ratext *reason,
    oratext *lang);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`msg`	`IN/OUT`	SOAP message
`reason`	`IN`	Human-readable fault Reason
`lang`	`IN`	Language of reason

Returns

`(xmlerr)` numeric error code, `XMLERR_OK [0]` on success

> **See Also:**
>
> XmlSoapSetFault()

XmlSoapAddFaultSubDetail()

Adds an additional child to Fault Detail. `XmlSoapSetFault` allows for creation of a Deatail element with only one child. Extra children could be added with this function.

Syntax

```
xmlerr XmlSoapAddFaultSubDetail(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   xmlelemnode *sub);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN/OUT	SOAP message
sub	IN	subdetail tree

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapGetReasonLang()

XmlSoapAddHeaderElement()

Adds an element to a SOAP header.

Syntax

```
xmlelemnode *XmlSoapAddHeaderElement(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   oratext *qname,
   oratext *uri,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN/OUT	SOAP message
qname	IN	QName of element to add
uri	IN	Namespace URI of element to add
xerr	IN/OUT	error code

Returns

(xmlelemnode *) created element

> **See Also:**
>
> XmlSoapAddBodyElement(), XmlSoapGetHeaderElement()

XmlSoapCall()

Send a SOAP message over a connection and wait for the response; the message reply (an XML document) is parsed and returned as a SOAP message (equivalent to a DOM).

The message buffer is first used to serialize the outgoing message; if it's too small (overflow occurs), xerr gets XMLERR_SAVE_OVERFLOW and `NULL` is returned. The same buffer is then re-used to receive the replied SOAP message.

Opening the connection object is expected to cause an active SOAP handler to appear on the end-point; how this happens is up to the user. For HTTP, the URL should invoke a cgi-bin, or detect the application/soap+xml content-type.

Syntax

```
xmldocnode *XmlSoapCall(
   xmlsoapctx *ctx,
   xmlsoapcon *con,
   xmldocnode *msg,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`con`	`IN`	SOAP connection object
`msg`	`IN`	SOAP message to send
`xerr`	`IN`	numeric code of failure

Returns

`(xmldocnode *)` returned message, or `NULL` on failure with xerr set

> **See Also:**
>
> XmlSoapCreateMsg(), XmlSoapCreateConnection(), XmlSoapDestroyConnection()

XmlSoapCreateConnection()

Create a SOAP connection object, specifying the binding (transport) and endpoint. The binding is an enum of type xmlsoapbind, and the endpoint depends on the binding.

Currently only HTTP binding is supported, and the endpoint is a URL. That URL should be active, i.e. a cgi-bin script or some mechanism to trigger SOAP message processing based on the Content-type of the incoming message ("application/soap+xml").

To control the HTTP access method (GET or POST), use the web-method property named `XMLSOAP_PROP_WEB_METHOD` which can have possible values `XMLSOAP_WEB_METHOD_GET` and `XMLSOAP_WEB_METHOD_POST`.

(conbuf, consiz) is the connection buffer used with LPU; for sending, it contains only the HTTP header, but on reception it holds the entire reply, including HTTP header and the full SOAP body. If no buffer is provided, one will be allocated for you. If size if zero, the default size (64K) will be used.

(msgbuf, msgsiz) is the message buffer used to form SOAP messages for sending. It needs to be large enough to contain the largest message which will be sent. If no buffer is specified, one will be allocated for you. If the size is zero, the default size (64K) will be used.

Two buffers are needed for sending since the SOAP message needs to be formed first in order to determine it's size; then, the HTTP header can be formed, since the Content-Length is now known.

Syntax

```
xmlsoapcon *XmlSoapCreateConnection(
   xmlsoapctx *ctx,
   xmlerr *xerr,
   xmlsoapbind bind,
   void *endp,
   oratext *conbuf,
   ubig_ora consiz,
   oratext *msgbuf,
   ubig_ora msgsiz,
   ...);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`xerr`	`OUT`	numeric error code on failure
`bind`	`IN`	connection binding
`endp`	`IN`	connection endpoint
`conbuf`	`IN/OUT`	connection buffer (or `NULL` to have one allocated)
`consiz`	`IN`	size of connection buffer (or 0 for default size)
`msgbuf`	`IN/OUT`	message buffer (or `NULL` to have one allocated)
`msgsiz`	`IN`	size of message buffer (or 0 for default size)

Parameter	In/Out	Description
...	IN	additional HTTP headers to set, followed by NULL

Returns

(xmlsoapcon *) connect object, or NULL on error with xerr set

> **See Also:**
>
> XmlSoapDestroyConnection(), XmlSoapCall()

XmlSoapCreateCtx()

Creates and returns a SOAP context. This context must be passed to all XmlSoap APIs. Note the name provided should be unique and is used to identify the context when debugging. Options are specified as (name, value) pairs, ending with a NULL, same as for XmlCreate. If no options are desired, the NULL is still needed. Options are: debug_level (enables SOAP debug output to stderr), numeric level (the higher the level, the more detailed extensive the output), 0 for no debug (this is the default setting).

Syntax

```
xmlsoapctx *XmlSoapCreateCtx(
   xmlctx *xctx,
   xmlerr *xerr,
   oratext *name,
   ...);
```

Parameter	In/Out	Description
xctx	IN	XML context
xerr	OUT	error return code on failure
name	IN	name of context; used for debugging
...	IN	options, as (name, value) pairs, followed by NULL

Returns

(xmlsoapctx *) SOAP context, or NULL on failure (w/xerr set)

> **See Also:**
>
> XmlSoapDestroyCtx()

XmlSoapCreateMsg()

Creates and returns an empty SOAP message. The SOAP message will consist of an Envelope. The Envelope contains an empty Header and Body. A SOAP message is an XML document represented by a DOM, and is no different from any other XML document. All DOM operations are valid on the document, but be sure not to harm the overall structure. Changes should be restricted to creating and modifying elements inside the Header and Body.

Syntax

```
xmldocnode *XmlSoapCreateMsg(
   xmlsoapctx *ctx,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
xerr	OUT	error retrun code on failure

Returns

`(xmldocnode *)` SOAP message, or `NULL` on failure (w/xerr set)

> **See Also:**
>
> XmlSoapDestroyMsg()

XmlSoapDestroyConnection()

Destroys a SOAP connection object made with XmlSoapCreateConnection and frees all allocated resources.

Syntax

```
xmlerr XmlSoapDestroyConnection(
   xmlsoapctx *ctx,
   xmlsoapcon *con);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
con	IN	SOAP connection

Returns

`(xmlerr)` numeric error code, `XMLERR_OK [0]` on success

> **See Also:**
>
> XmlSoapCreateConnection(), XmlSoapCall()

XmlSoapDestroyCtx()

Destroys a SOAP context created with XmlSoapCreateCtx. All memory allocated will be freed, and all connections closed.

Syntax

```
xmlerr XmlSoapDestroyCtx(
   xmlsoapctx *ctx);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context

Returns

`(xmlerr)` numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapCreateCtx()

XmlSoapDestroyMsg()

Destroys a SOAP message created with XmlSoapCreateMsg; this is the same as calling XmlFreeDocument.

Syntax

```
xmlerr XmlSoapDestroyMsg(
   xmlsoapctx *ctx,
   xmldocnode *msg);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP connection
`msg`	`IN`	SOAP message

Returns

`(xmlerr)` numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapCreateMsg()

XmlSoapError()

Retrives human readable representation of the error code. Optionally, retrieves the information about the error code of the underlying layer.

Syntax

```
oratext *XmlSoapError(
   xmlsoapctx *ctx,
   xmlsoapcon *con,
   xmlerr err,
   uword *suberr,
   oratext **submsg);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`con`	`IN`	Connection about which additional info is requested
`err`	`IN`	Error code for which human readable information will be returned.
`suberr`	`OUT`	Error code from `con`
`submsg`	`OUT`	Human readable information about `con` error

Returns

`(oratext *)` error code

XmlSoapGetBody()

Returns a SOAP message's envelope body.

Syntax

```
xmlelemnode *XmlSoapGetBody(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message
xmlerr	IN/OUT	error code

Returns

(xmlelemnode *) SOAP Body

> **See Also:**
>
> XmlSoapGetHeader()

XmlSoapGetBodyElement()

Gets an element from a SOAP body.

Syntax

```
xmlelemnode *XmlSoapGetBodyElement(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   oratext *uri,
    oratext *local,
    xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message
uri	IN	Namespace URI of element to get
local	IN	Local name of element to get
xerr	IN/OUT	error code

Returns

(xmlelemnode *) named element, or NULL on error

> **See Also:**
>
> XmlSoapAddBodyElement()

XmlSoapGetEnvelope()

Returns a SOAP part's envelope

Syntax

```
xmlelemnode *XmlSoapGetEnvelope(
   mlsoapctx *ctx,
   xmldocnode *msg,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message
xerr	IN/OUT	error code

Returns

(xmlelemnode *) SOAP envelope

XmlSoapGetFault()

Returns Fault code, reason, and details. Fetches the Fault information and returns through user variables. NULL may be supplied for any part which is not needed. For lang, if the pointed-to variable is NULL, it will be set to the default language (that of the first reason).

Syntax

```
xmlerr XmlSoapGetFault(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   oratext **code,
   oratext **reason,
   oratext **lang,
   oratext **node,
   oratext **role,
   xmlelemnode **detail);
```

Parameter	In/Out	Description
ctx	IN	SOAP context

Parameter	In/Out	Description
msg	IN/OUT	SOAP message
code	OUT	Code (1.2), faultcode (1.1)
reason	OUT	Human-readable fault Reason (1.2), faultreason (1.1)
lang	IN	Desired language for reason (1.2), not used (NULL in 1.1)
node	OUT	Fault node
role	OUT	Role: next, none, or ulitmate receiver. Not used in 1.1
detail	OUT	User-defined details

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapSetFault()

XmlSoapGetHeader()

Returns a SOAP message's envelope header.

Syntax

```
xmlelemnode *XmlSoapGetHeader(
   xmlsoapctx *ctx,
  xmldocnode *msg,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message
xerr	IN/OUT	error code

Returns

(xmlelemnode *) SOAP header

> **See Also:**
>
> XmlSoapGetBody()

XmlSoapGetHeaderElement()

Gets an element from a SOAP header. Sets a numeric error code.

Syntax

```
xmlelemnode *XmlSoapGetHeaderElement(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   oratext *uri,
   oratext *local,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`msg`	`IN`	SOAP message
`uri`	`IN`	Namespace URI of element to get
`local`	`IN`	Local name of element to get
`xerr`	`IN/OUT`	Error code

Returns

`(xmlelemnode *)` named element, or `NULL` on error

> **See Also:**
>
> XmlSoapAddHeaderElement(), XmlSoapGetBodyElement()

XmlSoapGetMustUnderstand()

Gets `mustUnderstand` attribute from SOAP header element. The absence of this attribute is not an error and treated as value FALSE. To indicate the absence of an attribute, the error code XMLERR_SOAP_NO_MUST_UNDERSTAND is returned in this case, XMLERR_OK (0) is returned if the attribute is present. Other appropriate error codes might be returned. User supplied `mustUnderstand` value is set accordingly.

Syntax

```
xmlerr XmlSoapGetMustUnderstand(
   xmlsoapctx *ctx,
   xmlelemnode *elem,
   boolean *mustUnderstand);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
elem	IN	SOAP header element
mustUnderstand	OUT	mustUnderstand value, TRUE|FALSE

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapAddBodyElement(), XmlSoapSetMustUnderstand()

XmlSoapGetReasonLang()

Gets the language of a reason with a particular index.

Syntax

```
xmlerr XmlSoapGetReasonLang(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   ub4 index,
   oratext **lang);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message
indx	IN	Index of fault reason
lang	IN	Reason language

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapGetFault(), XmlSoapHasFault(), XmlSoapGetReasonNum()

XmlSoapGetReasonNum()

Determines the number of reasons in Fault element. Returns 0 if Fault is not present.

Syntax

```
ub4 XmlSoapGetReasonNum(
   xmlsoapctx *ctx,
   xmldocnode *msg);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message

Returns

(ub4 *) #num reasons

> **See Also:**
>
> XmlSoapGetFault(), XmlSoapHasFault()

XmlSoapGetRelay()

Gets Relay attribute from SOAP header element.

Syntax

```
xmlerr XmlSoapGetRelay(
   xmlsoapctx *ctx,
   xmlelemnode *elem,
   boolean *Relay);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
elem	IN	SOAP header element
Relay	OUT	Relay value

Returns

`xmlerr` numeric error code, `XMLERR_OK` on success

> **See Also:**
>
> XmlSoapAddBodyElement(), XmlSoapSetRelay()

XmlSoapGetRole()

Gets role from SOAP header element. If the element has no role, `XMLERR_SOAP_NO_ROLE` is returned, otherwise `XMLERR_OK (0)` is returned and the user's role is set accordingly. if the element has no role, then according to the standard, the user's role is set to `XMLSOAP_ROLE_ULT`.

Syntax

```
xmlerr XmlSoapGetRole(
   xmlsoapctx *ctx,
   xmlelemnode *elem,
   xmlsoaprole *role);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`elem`	`IN`	SOAP header element
`role`	`OUT`	Role value

Returns

`(xmlerr)` numeric error code, `XMLERR_OK [0]` on success

> **See Also:**
>
> XmlSoapSetMustUnderstand(), XmlSoapSetRole()

XmlSoapHasFault()

Determines if SOAP message contains Fault object.

Syntax

```
boolean XmlSoapHasFault(
   xmlsoapctx *ctx,
```

```
   xmldocnode *msg,
   xmlerr *xerr);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN	SOAP message
xerr	IN/OUT	Error code

Returns

(boolean) TRUE if there's a Fault, FALSE if not

> **See Also:**
>
> XmlSoapGetFault()

XmlSoapSetFault()

Sets Fault in SOAP message.

- In version 1.2, only one Fault is allowed for each message, and it must be the only child of the Body. If the Body has children, they are first removed and freed. The Fault is then added with children code - "env:Code" (required), reason - "env:Reason" (required), node - "env:Node" (optional), role - "env:role"(optional), and detail - "env:Detail" (optional). The primary-language reason should be added first; calls to XmlSoapGetFault which pass a NULL language will pick this reason. Detail is the user-defined subtree to be spliced into the Fault.
- In version 1.1, only one Fault is allowed per message. If the Body already has Fault, it is first removed and freed. The Fault is then added with children code - "faultcode" (required), reason - "faultstring" (required), node - "faultactor" (optional), and detail - "detail" (optional). Detail is the user-defined subtree to be spliced into the Fault. role and lang are not used in ver 1.1

Syntax

```
xmlerr XmlSoapSetFault(
   xmlsoapctx *ctx,
   xmldocnode *msg,
   oratext *node,
   oratext *code,
   oratext *reason,
   oratext *lang,
   oratext *role,
   xmlelemnode *detail);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
msg	IN/OUT	SOAP message
node	IN	URI of SOAP node which faulted, Node (1.2), faultactor(1.1)
code	IN	Code (1.2), faultcode (1.1)
reason	IN	Human-readable fault Reason (1.2), faultreason (1.1)
lang	IN	Language of reason (1.2), unused (1.1)
role	IN	URI representing role, Role (1.2), unused (1.1)
detail	IN	detail elements (user-defined)

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlSoapAddFaultReason()

XmlSoapSetMustUnderstand()

Sets mustUnderstand attribute for SOAP header element. According to the standard, if the value is FALSE, the attribute is not set.

Syntax

```
xmlerr XmlSoapSetMustUnderstand(
   xmlsoapctx *ctx,
   xmlelemnode *elem,
   boolean mustUnderstand);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
elem	IN/OUT	SOAP header element
mustUnderstand	IN	mustUnderstand value, TRUE|FALSE

Returns

`(xmlerr)` numeric error code, `XMLERR_OK [0]` on success

> **See Also:**
>
> XmlSoapSetRole()

XmlSoapSetRelay()

Sets Relay attribute for a SOAP header element. If the value is FALSE, the attribute is not set.

Syntax

```
xmlerr XmlSoapSetRelay(
   xmlsoapctx *ctx,
   xmlelemnode *elem,   boolean Relay);
```

Parameter	In/Out	Description
`ctx`	`IN`	SOAP context
`elem`	`IN/OUT`	SOAP header element
`Relay`	`IN`	Relay; `TRUE|FALSE`

Returns

`(xmlerr)` numeric error code, `XMLERR_OK [0]` on success

> **See Also:**
>
> XmlSoapGetRelay()

XmlSoapSetRole()

Sets role for SOAP header element. If the role specified is XMLSOAP_ROLE_ULT, then according to the standard the attribute is not set.

Syntax

```
xmlerr XmlSoapSetRole(
   xmlsoapctx *ctx,
   xmlelemnode *elem,
   xmlsoaprole role);
```

Parameter	In/Out	Description
ctx	IN	SOAP context
elem	IN/OUT	SOAP header element
role	IN	Role value

Returns

xmlerr numeric error code, XMLERR_OK on success

> **See Also:**
>
> XmlSoapSetMustUnderstand()

10

Package Traversal for XML C APIs

Package Traversal contains APIs that implement the rules and behavior of traversal. For convenience, we grouped the APIs under four general interfaces types:

- DocumentTraversal Interface
- NodeFilter Interface
- NodeIterator Interface
- TreeWalker Interface

DocumentTraversal Interface for Traversal XML C APIs

The following table summarizes the methods available through the `DocumentTraversal` interface of `Traversal` for XML C APIs.

Table 10-1 Summary of DocumentTraversal Traversal Methods for XML C Implementation

Function	Summary
XmlDomCreateNodeIter()	Create node iterator object.
XmlDomCreateTreeWalker()	Create a tree walker object.

XmlDomCreateNodeIter()

One of two methods of `DocumentTraversal` interface, used to create a `NodeIterator` object. This method is identical to XmlDomCreateTreeWalker() except for the type of object returned.

The `whatToShow` argument is a mask of flag bits, one for each node type. The value `XMLDOM_SHOW_ALL` passes all node types through, otherwise only the types whose bits are set will be passed.

`Entity` reference expansion is controlled by the `entrefExpansion` flag. If `TRUE`, entity references are replaced with their final content; if `FALSE`, entity references are left as nodes.

Syntax

```
xmliter* XmlDomCreateNodeIter(
   xmlctx *xctx,
   xmliter *iter,
   xmlnode *root,
   xmlshowbits whatToShow,
   XMLDOM_ACCEPT_NODE_F(
      (*nodeFilter),
      xctx,
      node),
   boolean entrefExpand);
```

Parameter	In/Out	Description
xctx	IN	XML context
iter	IN	existing NodeIterator to set, NULL to create
xerr	IN	root node for NodeIterator
whatToShow	IN	mask of XMLDOM_SHOW_*XXX* flag bits
nodeFilter	IN	node filter to be used, NULL if none
xerr	IN	whether to expand entity reference nodes

Returns

(xmliter *) original or new NodeIterator object

> **See Also:**
>
> XmlDomCreateTreeWalker()

XmlDomCreateTreeWalker()

One of two methods of DocumentTraversal interface, used to create a TreeWalker object. This method is identical to XmlDomCreateNodeIter() except for the type of object returned.

The whatToShow argument is a mask of flag bits, one for each node type. The value XMLDOM_SHOW_ALL passes all node types through, otherwise only the types whose bits are set will be passed.

Entity reference expansion is controlled by the entrefExpansion flag. If TRUE, entity references are replaced with their final content; if FALSE, entity references are left as nodes.

Syntax

```
xmlwalk* XmlDomCreateTreeWalker(
   xmlctx *xctx,
   xmlwalk* walker,
   xmlnode *root,
   xmlshowbits whatToShow,
   XMLDOM_ACCEPT_NODE_F(
      (*nodeFilter),
      xctx,
      node),
   boolean entrefExpansion);
```

Parameter	In/Out	Description
xctx	IN	XML context
walker	IN	existing TreeWalker to set, NULL to create
xerr	IN	root node for TreeWalker
whatToShow	IN	mask of XMLDOM_SHOW_*XXX* flag bits
nodeFilter	IN	node filter to be used, NULL if none
xerr	IN	whether to expand entity reference nodes

Returns

(xmlwalk *) new TreeWalker object

> **See Also:**
>
> XmlDomCreateNodeIter()

NodeFilter Interface for Traversal XML C APIs

The following table summarizes the methods available through the NodeFilter interface of Traversal for XML C APIs.

Table 10-2 Summary of NodeFilter Traversal Methods for XML C Implementation

Function	Summary
XMLDOM_ACCEPT_NODE_F()	Determines the filtering action based on node adn filter.

XMLDOM_ACCEPT_NODE_F()

Sole method of NodeFilter interface. Given a node and a filter, determines the filtering action to perform.

This function pointer is passed to the node iterator/tree walker methods, as needed.

Values for xmlerr are:

- XMLERR_OK Accept the node. Navigation methods defined for NodeIterator or TreeWalker will return this node.
- XMLERR_FILTER_REJECT Reject the node. Navigation methods defined for NodeIterator or TreeWalker will not return this node. For TreeWalker, the children of this node will also be rejected. NodeIterators treat this as a synonym for XMLDOM_FILTER_SKIP

- `XMLERR_FILTER_SKIP` Skip this single node. Navigation methods defined for `NodeIterator` or `TreeWalker` will not return this node. For both `NodeIterator` and `TreeWalker`, the children of this node will still be considered.

Syntax

```
#define XMLDOM_ACCEPT_NODE_F(func, xctx, node)
xmlerr func(
   xmlctx *xctx,
   xmlnode *node);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`node`	`IN`	node to test

Returns

`(xmlerr)` filtering result

NodeIterator Interface for Traversal XML C APIs

The following table summarizes the methods available through the `NodeIterator` interface of `Traversal` for XML C APIs.

Table 10-3 Summary of NodeIterator Traversal Methods for XML C Implementation

Function	Summary
XmlDomIterDetach()	Detach a node iterator (deactivate it).
XmlDomIterNextNode()	Returns next node for iterator.
XmlDomIterPrevNode()	Returns previous node for iterator.

XmlDomIterDetach()

Detaches the `NodeIterator` from the set which it iterated over, releasing any resources and placing the iterator in the `INVALID` state. After detach has been invoked, calls to `XmlDomIterNextNode` or `XmlDomIterPrevNode` will raise the exception `XMLERR_ITER_DETACHED`.

Syntax

```
xmlerr XmlDomIterDetach(
   xmlctx *xctx,
   xmliter *iter);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
iter	IN	node iterator object

See Also:

XmlDomIterNextNode(), XmlDomIterPrevNode()

XmlDomIterNextNode()

Returns the next node in the set and advances the position of the iterator in the set. After a node iterator is created, the first call to XmlDomIterNextNode returns the first node in the set. It assumed that the reference node (current iterator position) is never deleted. Otherwise, changes in the underlying DOM tree do not invalidate the iterator.

Syntax

```
xmlnode* XmlDomIterNextNode(
   xmlctx *xctx,
   xmliter *iter,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
iter	IN	node iterator object
xerr	OUT	numeric return error code

Returns

(xmlnode *) next node in set being iterated over [or NULL]

See Also:

XmlDomIterPrevNode(), XmlDomIterDetach()

XmlDomIterPrevNode()

Returns the previous node in the set and moves the position of the iterator backward in the set.

Syntax

```
xmlnode* XmlDomIterPrevNode(
   xmlctx *xctx,
   xmliter *iter,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
iter	IN	node iterator object
xerr	OUT	numeric return error code

Returns

`(xmlnode *)` previous node in set being iterated over [or `NULL`]

> **See Also:**
>
> XmlDomIterNextNode(), XmlDomIterDetach()

TreeWalker Interface for Traversal XML C APIs

Table 10-4 summarizes the methods available through the `TreeWalker` interface of `Traversal` for XML C APIs.

Table 10-4 Summary of TreeWalker Traversal Methods for XML C Implementation

Function	Summary
XmlDomWalkerFirstChild()	Return first visible child of current node.
XmlDomWalkerGetCurrentNode()	Return current node.
XmlDomWalkerGetRoot()	Return root node.
XmlDomWalkerLastChild()	Return last visible child of current node.
XmlDomWalkerNextNode()	Return next visible node.
XmlDomWalkerNextSibling()	Return next sibling node.
XmlDomWalkerParentNode()	Return parent node.
XmlDomWalkerPrevNode()	Return previous node.
XmlDomWalkerPrevSibling()	Return previous sibling node.
XmlDomWalkerSetCurrentNode()	Set current node.
XmlDomWalkerSetRoot()	Set the root node.

XmlDomWalkerFirstChild()

Moves the `TreeWalker` to the first visible child of the current node, and returns the new node. If the current node has no visible children, returns `NULL`, and retains the current node.

Syntax

```
xmlnode* XmlDomWalkerFirstChild(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`walker`	`IN`	`TreeWalker` object
`xerr`	`OUT`	numeric return error code

Returns

`(xmlnode *)` first visible child [or `NULL`]

> **See Also:**
>
> XmlDomWalkerLastChild()

XmlDomWalkerGetCurrentNode()

Return (get) current node, or `NULL` on error.

Syntax

```
xmlnode* XmlDomWalkerGetCurrentNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`walker`	`IN`	`TreeWalker` object
`xerr`	`OUT`	numeric return error code

Returns

`(xmlnode *)` current node

XmlDomWalkerGetRoot()

Return (get) root node, or `NULL` on error. Since the current node can be removed from under the root node together with a subtree where it belongs to, the current root node in a walker might have no relation to the current node any more. The `TreeWalker` iterations are based on the current node. However, the root node defines the space of an iteration. This function checks if the root node is still in the root node (ancestor) relation to the current node. If so, it returns this root node. Otherwise, it finds the root of the tree where the current node belongs to, and sets and returns this root as the root node of the walker. It returns `NULL` if the walker is a `NULL` pointer.

Syntax

```
xmlnode* XmlDomWalkerGetRoot(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`walker`	`IN`	TreeWalker object
`xerr`	`OUT`	numeric return error code

Returns

`(xmlnode *)` root node

XmlDomWalkerLastChild()

Moves the `TreeWalker` to the last visible child of the current node, and returns the new node. If the current node has no visible children, returns `NULL`, and retains the current node.

Syntax

```
xmlnode* XmlDomWalkerLastChild(
   xmlctx *xctx,
   xmlwalk *walker,
    xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

Parameter	In/Out	Description
walker	IN	TreeWalker object
xerr	OUT	numeric return error code

Returns

(xmlnode *) last visible children [or NULL]

XmlDomWalkerNextNode()

Moves the TreeWalker to the next visible node in document order relative to the current node, and returns the new node. If the current node has no next node, or if the search for the next node attempts to step upward from the TreeWalker's root node, returns NULL, and retains the current node.

Syntax

```
xmlnode* XmlDomWalkerNextNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
walker	IN	TreeWalker object
xerr	OUT	numeric return error code

Returns

(xmlnode *) next node [or NULL]

> **See Also:**
>
> XmlDomWalkerPrevNode(), XmlDomWalkerNextSibling(), XmlDomWalkerPrevSibling()

XmlDomWalkerNextSibling()

Moves the TreeWalker to the next sibling of the current node, and returns the new node. If the current node has no visible next sibling, returns NULL, and retains the current node.

Syntax

```
xmlnode* XmlDomWalkerNextSibling(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
walker	IN	TreeWalker object
xerr	OUT	numeric return error code

Returns

(xmlnode *) next sibling [or NULL]

> **See Also:**
>
> XmlDomWalkerNextNode(), XmlDomWalkerPrevNode(), XmlDomWalkerPrevSibling()

XmlDomWalkerParentNode()

Moves to and returns the closest visible ancestor node of the current node. If the search for the parent node attempts to step upward from the TreeWalker's root node, or if it fails to find a visible ancestor node, this method retains the current position and returns null.

Syntax

```
xmlnode* XmlDomWalkerParentNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
walker	IN	TreeWalker object
xerr	OUT	numeric return error code

Returns

(xmlnode *) parent node [or NULL]

XmlDomWalkerPrevNode()

Moves the `TreeWalker` to the previous visible node in document order relative to the current node, and returns the new node. If the current node has no previous node, or if the search for the previous node attempts to step upward from the `TreeWalker`'s root node, returns `NULL`, and retains the current node.

Syntax

```
xmlnode* XmlDomWalkerPrevNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`walker`	`IN`	`TreeWalker` object
`xerr`	`OUT`	numeric return error code

Returns

`(xmlnode *)` previous node [or `NULL`]

> **See Also:**
>
> XmlDomWalkerNextNode(), XmlDomWalkerNextSibling(), XmlDomWalkerPrevSibling()

XmlDomWalkerPrevSibling()

Moves the `TreeWalker` to the previous sibling of the current node, and returns the new node. If the current node has no visible previous sibling, returns `NULL`, and retains the current node.

Syntax

```
xmlnode* XmlDomWalkerPrevSibling(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`walker`	`IN`	`TreeWalker` object

Parameter	In/Out	Description
xerr	OUT	numeric return error code

Returns

(xmlnode *) previous sibling [or NULL]

> **See Also:**
>
> XmlDomWalkerNextNode(), XmlDomWalkerPrevNode(), XmlDomWalkerNextSibling()

XmlDomWalkerSetCurrentNode()

Sets and returns new current node. It also checks if the root node is an ancestor of the new current node. If not it does not set the current node, returns NULL, and sets retval to XMLDOM_WALKER_BAD_NEW_CUR. Returns NULL if an error.

Syntax

```
xmlnode* XmlDomWalkerSetCurrentNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlnode *node,
   xmlerr *xerr);
```

Parameter	In/Out	Description
xctx	IN	XML context
walker	IN	TreeWalker object
node	IN	new current node
xerr	OUT	numeric return error code

Returns

(xmlnode *) new current node

XmlDomWalkerSetRoot()

Set the root node. Returns new root node if it is an ancestor of the current node. If not it signals an error and checks if the current root node is an ancestor of the current node. If yes it returns it. Otherwise it sets the root node to and returns the root of the tree where the current node belongs to. It returns NULL if the walker or the root node parameter is a NULL pointer.

Syntax

```
xmlnode* XmlDomWalkerSetRoot(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlnode *node,
   xmlerr *xerr);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`walker`	`IN`	`TreeWalker` object
`node`	`IN`	new root node
`xerr`	`OUT`	numeric return error code

Returns

`(xmlnode *)` new root node

11
Package XML for XML C APIs

This C implementation of the XML processor (or parser) follows the W3C XML specification (rev REC-xml-19980210) and implements the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The following table summarizes the methods available through the `XML` package for XML C APIs.

Table 11-1 Summary of XML Methods for XML C Implementation

Function	Summary
XmlAccess()	Set access method callbacks for URL.
XmlCreate()	Create an XML Developer's Toolkit `xmlctx`.
XmlCreateDTD()	Create DTD.
XmlCreateDocument()	Create Document (node).
XmlDestroy()	Destroy an `xmlctx`.
XmlDiff()	Compares two XML documents.
XmlFreeDocument()	Free a document (releases all resources).
XmlGetEncoding()	Returns data encoding in use by XML context.
XmlHasFeature()	Determine if DOM feature is implemented.
XmlIsSimple()	Returns single-byte (simple) characterset flag.
XmlIsUnicode()	Returns `XmlIsUnicode` (simple) characterset flag.
XmlLoadDom()	Load (parse) an XML document and produce a DOM.
XmlLoadSax()	Load (parse) an XML document from and produce SAX events.
XmlLoadSaxVA()	Load (parse) an XML document from and produce SAX events [`varargs`].
XmlSaveDom()	Saves (serializes, formats) an XML document.
XmlVersion()	Returns version string for XDK.

XmlAccess()

Sets the open/read/close callbacks used to load data for a specific URL access method. Overrides the built-in data loading functions for HTTP, FTP, and so on, or provides functions to handle new types, such as `UNKNOWN`.

Syntax

```
xmlerr XmlAccess(
   xmlctx *xctx,
   xmlurlacc access,
   void *userctx,
   XML_ACCESS_OPEN_F(
      (*openf),
```

```
      ctx,
      uri,
      parts,
      length,
      uh),
   XML_ACCESS_READ_F(
      (*readf),
      ctx,
      uh,
      data,
      nraw,
      eoi),
   XML_ACCESS_CLOSE_F(
      (*closef),
      ctx,
      uh));
```

Parameter	In/Out	Description
xctx	IN	XML context
access	IN	URL access method
userctx	IN	user-defined context passed to callbacks
openf	IN	open-access callback function
readf	IN	read-access callback function
closef	IN	close-access callback function

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlLoadDom(), XmlLoadSax()

XmlCreate()

Create an XML Developer's Toolkit xmlctx.

Syntax

```
xmlctx *XmlCreate(
   xmlerr *err,
   oratext *name,
   list);
```

Parameter	In/Out	Description
err	OUT	returned error code
access	IN	name of context, for debugging

Parameter	In/Out	Description

list	IN	NULL-terminated list of variable arguments. Properties common to all xmlctx's, both XDK and XMLType, are:
		• data_encoding is the data encoding in which XML data will be presented through DOM and SAX. Default is UTF-8 and UTF-E on EBCDIC platforms. Single-byte encodings are substantially faster than multibyte encodings; Unicode (UTF-16) uses more memory but has better performance than multibyte. If the data_encoding parameter is set to UTF-16, the APIs process wide-CHAR arrays, not oratext byte arrays.
		• default_input_encoding is the default input encoding). If the encoding of an input document cannot be automatically determined through other methods, this encoding will be the default.
		• error_language is the language (and optional encoding) in which error messages are created. Default is American with UTF-8 encoding. To specify only the language, give the name of the language ("American"). To also specify the encoding, add the period and the Oracle name of the encoding ("American.WE8ISO8859P1").
		• error_handler is the function pointer; see XML_ERRMSG_F. By default, errors output the formatted message to stderr. If an error handler is provided, message will be passed to it, and not printed.
		• error_context is user-defined context for error handler, a context pointer to be passed to the error handler function. It is user-defined; it is just specified here and passed along when an error occurs.
		• input_encoding is the name of a forced input encoding for input documents. Use it to override a document's XMLDecl, and always interpret it in the given encoding. It should be not necessary in normal use, as existing BOMs and XMLDecls should be correct.
		• memory_alloc is a low-level memory allocation function, if not using malloc. If used, the matching free function must also be given. See XML_ALLOC_F.
		• memory_free is a low-level memory freeing function, if not using free. Matches the memory_alloc function.
		• memory_context is a user-defined memory context passed to the alloc and free functions. Its definition and use is entirely up to the user; it is just set here and passed to the callbacks.
		The XDK has additional properties:
		• input_buffer_size is the basic I/O buffer size. Default is 256K; the range is 4K to 4MB. Depending on the encoding, 1, 2 or 3 of these buffers may be needed. Note that size is in characters, not bytes. If the buffer holds Unicode data, it will be twice as large.
		• memory_block_size is the size of chunk the high-level memory package will request from the low-level allocator; it is the basic unit of memory allocation. Default is 64K; the range is 16K to 256K.
		These optional parameters should be used in the following manner:
		`xmlctx *XmlCreate(`
		`  xmlerr *err,`
		`  oratext *name,`
		`  ("data_encoding", dataEncoding),`
		`  ("default_data_encoding", defaultDataEncoding),`
		`  ("error_language", errorLanguage),`
		`  ("error_handler", errorHandler),`
		`  ("error_context", errorContext)`
		`  ("input_encoding", inputEncoding),`
		`  ("memory_alloc", memAlloc),`

Parameter	In/Out	Description
		("memory_free", *memFree*), ("memory_context", *memContext*), ("input_buffer_seize", *inputBufSize*), ("memory_block_size", *memBlockSize*));

Returns

`(xmlctx *)` created xmlctx [or `NULL` on error with err set]

> **See Also:**
>
> XmlDestroy(), XML_ERRMSG_F() in Package Callback APIs for C

XmlCreateDTD()

Create DTD.

Syntax

```
xmldocnode* XmlCreateDTD(
   xmlctx *xctx
   oratext *qname,
   oratext *pubid,
   oratext *sysid,
   xmlerr *err);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`qname`	`IN`	qualified name
`pubid`	`IN`	external subset public identifier
`sysid`	`IN`	external subset system identifier
`err`	`OUT`	returned error code

Returns

`(xmldtdnode *)` new DTD node

XmlCreateDocument()

Creates the initial top-level `DOCUMENT` node and its supporting infrastructure. If a qualified name is provided, a an element with that name is created and set as the document's root element.

Syntax

```
xmldocnode* XmlCreateDocument(
   xmlctx *xctx,
   oratext *uri,
   oratext *qname,
   xmldtdnode *dtd,
   xmlerr *err);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`uri`	`IN`	namespace URI of root element to create, or `NULL`
`qname`	`IN`	qualified name of root element, or `NULL` if none
`dtd`	`IN`	associated DTD node
`err`	`OUT`	returned error code

Returns

`(xmldocnode *)` new `Document` object.

XmlDestroy()

Destroys an XML context.

Syntax

```
void XmlDestroy(
   xmlctx *xctx);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context

> **See Also:**
>
> XmlCreate()

XmlDiff()

Compares two XML documents, specified either as DOM Trees, files, URIs, orastreams, and so on, and returns its document node. If input documents are not supplied as DOM trees, DOM trees will be created for them.

If the inputs are DOMs, that memory will not be freed when the call completes.

Data(DOM) encoding of both the documents must be the same as the data encoding in the XML context. The DOM for the diff will be created in the data encoding specified by the XML context.

Syntax

```
xmldocnode *XmlDiff(
   xmlctx *xctx,
   xmlerr *err,
   ub4  flags,
   xmldfsrct firstSourceType,
   void *firstSource,
   void *firstSourceExtra,
   xmldfsrct secondSourceType,
   void *secondSource,
   void *secondSourceExtra,
   uword hashLevel);
```

Parameter	In/Out	Description
xctx	IN	XML context
err	OUT	numeric error code, XMLERR_OK [0] on success
flags	IN	Comparison options. By default, global algorithm and snapshot model are used. • XMLDF_FL_DEFAULTS(=0) chooses defaults • XMLDF_FL_ALGORITHM_GLOBAL is the global algorithm; it will generate the minimal diff using INSERT, APPEND, DELETE and UPDATE, and needs more memory and time than XMLDF_FL_ALGORITHM_LOCAL • XMLDF_FL_ALGORITHM_LOCAL is the local algorithm; it may not generate the minimal diff, but it is faster and uses less space than XMLDF_FL_ALGORITHM_GLOBAL • XMLDF_FL_DISABLE_UPDATE disables update operations with global algorithms • XMLDF_FL_OUTPUT_SNAPSHOT uses the snapshot model
firstSourceType	IN	Source type for the first document. If 0, assumed to be a DOM document node.
firstSource	IN	Pointer to the first document source
firstSourceExtra	IN	An additional pointer to the first document source; used for the buffer length pointer.
secondSourceType	IN	Source type for the second document. If 0, assumed to be a DOM document node.

Parameter	In/Out	Description
secondSource	IN	Pointer to the second document source
secondSourceExtra	IN	An additional pointer to the second document source; used for the buffer length pointer.
hashLevel	IN	1-based depth (counting from the root), where hashing should be used for subtrees. Values less than or equal to 1 indicate no hashing. This value must be specified programmatically. The hash value for every element node is associated with the entire subtree rooted at that node. During the computation of the diff, there is no further drilling down into the tree beyond hash level depth. • If hashing is used with XMLDF_FL_ALGORITHM_GLOBAL, it will speed up diff computation significantly, but may reduce the quality of the diff. • With XMLDF_FL_ALGORITHM_LOCAL, it improves the quality of the diff

XmlFreeDocument()

Destroys a document created by XmlCreateDocument or through one of the Load functions. Releases all resources associated with the document, which is then invalid.

Syntax

```
void XmlFreeDocument(
   xmlctx *xctx,
   xmldocnode *doc);
```

Parameter	In/Out	Description
xctx	IN	XML context
doc	IN	document to free

> **See Also:**
>
> XmlCreateDocument(), XmlLoadDom()

XmlGetEncoding()

Returns data encoding in use by XML context. Ordinarily, the data encoding is chosen by the user, so this function is not needed. However, if the data encoding is not specified, and allowed to default, this function can be used to return the name of that default encoding.

Syntax

```
oratext *XmlGetEncoding(
   xmlctx *xctx);
```

Parameter	In/Out	Description
xctx	IN	XML context

Returns

(oratext *) name of data encoding

> **See Also:**
>
> XmlIsSimple(), XmlIsUnicode()

XmlHasFeature()

Determine if a DOM feature is implemented. Returns TRUE if the feature is implemented in the specified version, FALSE otherwise.

In level 1, the legal values for package are 'HTML' and 'XML' (case-insensitive), and the version is the string "1.0". If the version is not specified, supporting any version of the feature will cause the method to return TRUE.

- DOM 1.0 features are "XML" and "HTML".
- DOM 2.0 features are "Core", "XML", "HTML", "Views", "StyleSheets", "CSS", "CSS2", "Events", "UIEvents", "MouseEvents", "MutationEvents", "HTMLEvents", "Range", "Traversal"

Syntax

```
boolean XmlHasFeature(
   xmlctx *xctx,
   oratext *feature,
   oratext *version);
```

Parameter	In/Out	Description
xctx	IN	XML context
feature	IN	package name of the feature to test
version	IN	version number of the package name to test

Returns

(boolean) feature is implemented?

XmlIsSimple()

Returns a flag saying whether the context's data encoding is "simple", single-byte for each character, like ASCII or EBCDIC.

Syntax

```
boolean XmlIsSimple(
   xmlctx *xctx);
```

Parameter	In/Out	Description
xctx	IN	XML context

Returns

(boolean) TRUE of data encoding is "simple", FALSE otherwise

> **See Also:**
>
> XmlGetEncoding(), XmlIsUnicode()

XmlIsUnicode()

Returns a flag saying whether the context's data encoding is Unicode, UTF-16, with two-byte for each character.

Syntax

```
boolean XmlIsUnicode(
   xmlctx *xctx);
```

Parameter	In/Out	Description
xctx	IN	XML context

Returns

(boolean) TRUE of data encoding is Unicode, FALSE otherwise

> **See Also:**
>
> XmlGetEncoding(), XmlIsSimple()

XmlLoadDom()

Loads (parses) an XML document from an input source and creates a DOM. The root document node is returned on success, or `NULL` on failure (with err set).

The function takes two fixed arguments, the xmlctx and an error return code, then zero or more (property, value) pairs, then `NULL`.

`SOURCE` Input source is set by one of the following mutually exclusive properties (choose one):

- ("`uri`", document URI) [compiler encoding]
- ("`file`", document filesystem path) [compiler encoding]
- ("`buffer`", address of buffer, "`buffer_length`", # bytes in buffer)
- ("`stream`", address of stream object, "`stream_context`", pointer to stream object's context)
- ("`stdio`", `FILE*` stream)

`PROPERTIES` Additional properties:

- ("`dtd`", DTD node) DTD for document
- ("`base_uri`", document base URI) for documents loaded from other sources than a URI, sets the effective base URI. the document's base URI is needed in order to resolve relative URI include, import, and so on.
- ("`input_encoding`", encoding name) forced input encoding [name]
- ("`default_input_encoding`", `encoding_name`) default input encoding to assume if document is not self-describing (no BOM, protocol header, `XMLDecl`, and so on)
- ("`schema_location`", `string`) `schemaLocation` of schema for this document. used to figure optimal layout when loading documents into a database
- ("`validate`", boolean) when `TRUE`, turns on DTD validation; by default, only well-formedness is checked. note that schema validation is a separate beast.
- ("`discard_whitespace`", boolean) when `TRUE`, formatting whitespace between elements (newlines and indentation) in input documents is discarded. by default, ALL input characters are preserved.
- ("`dtd_only`", boolean) when `TRUE`, parses an external DTD, not a complete XML document.
- ("`stop_on_warning`", boolean) when `TRUE`, warnings are treated the same as errors and cause parsing, validation, and so on, to stop immediately. by default, warnings are issued but the game continues.
- ("`warn_duplicate_entity`", boolean) when `TRUE`, entities which are declared more than once will cause warnings to be issued. the default is to accept the first declaration and silently ignore the rest.
- ("`no_expand_char_ref`", boolean) when `TRUE`, causes character references to be left unexpanded in the DOM data. ordinarily, character references are replaced by the character they represent. however, when a document is saved those characters entities do not reappear. to way to ensure they remain through load and save is to not expand them.

- ("no_check_chars", boolean) when TRUE, omits the test of XML [2] Char production: all input characters will be accepted as valid

Syntax

```
xmldocnode *XmlLoadDom(
   xmlctx *xctx,
   xmlerr *err,
   list);
```

Parameter	In/Out	Description
xctx	IN	XML context
err	OUT	returned error code
list	IN	NULL-terminated list of variable arguments

Returns

(xmldocnode *) document node on success [NULL on failure with err set]

> **See Also:**
>
> XmlSaveDom()

XmlLoadSax()

Loads (parses) an XML document from an input source and generates a set of SAX events (as user callbacks). Input sources and basic set of properties is the same as for XmlLoadDom.

Syntax

```
xmlerr XmlLoadSax(
   xmlctx *xctx,
   xmlsaxcb *saxcb,
   void *saxctx,
   list);
```

Parameter	In/Out	Description
xctx	IN	XML context
saxcb	IN	SAX callback structure
saxctx	IN	context passed to SAX callbacks
list	IN	NULL-terminated list of variable arguments

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

XmlLoadSaxVA()

Loads (parses) an XML document from an input source and generates a set of SAX events (as user callbacks). Input sources and basic set of properties is the same as for XmlLoadDom.

Syntax

```
xmlerr XmlLoadSaxVA(
   xmlctx *xctx,
   xmlsaxcb *saxcb,
   void *saxctx,
   va_list va);
```

Parameter	In/Out	Description
xctx	IN	XML context
saxcb	IN	SAX callback structure
saxctx	IN	context passed to SAX callbacks
va	IN	NULL-terminated list of variable arguments

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

XmlSaveDom()

Serializes document or subtree to the given destination and returns the number of bytes written; if no destination is provided, just returns formatted size but does not output.

If an output encoding is specified, the document will be re-encoded on output; otherwise, it will be in its existing encoding.

The top level is indented step*level spaces, the next level step*(level+1) spaces, and so on.

When saving to a buffer, if the buffer overflows, 0 is returned and err is set to XMLERR_SAVE_OVERFLOW.

DESTINATION Output destination is set by one of the following mutually exclusive properties (choose one):

- ("uri", document URI) POST, PUT? [compiler encoding]
- ("file", document filesystem path) [compiler encoding]
- ("buffer", address of buffer, "buffer_length", # bytes in buffer)

- ("stream", address of stream object, "stream_context", pointer to stream object's context)

PROPERTIES Additional properties:

- ("output_encoding", encoding name) name of final encoding for document. unless specified, saved document will be in same encoding as xmlctx.
- ("indent_step", unsigned) spaces to indent each level of output. default is 4, 0 means no indentation.
- ("indent_level", unsigned) initial indentation level. default is 0, which means no indentation, flush left.
- ("xmldecl", boolean) include an XMLDecl in the output document. ordinarily an XMLDecl is output for a compete document (root node is DOC).
- ("bom", boolean) input a BOM in the output document. usually the BOM is only needed for certain encodings (UTF-16), and optional for others (UTF-8). causes optional BOMs to be output.
- ("prune", boolean) prunes the output like the unix 'find' command; does not not descend to children, just prints the one node given.

Syntax

```
ubig_ora XmlSaveDom(
   xmlctx *xctx,
   xmlerr *err,
   xmlnode *root,
   list);
```

Parameter	In/Out	Description
xctx	IN	XML context
err	OUT	error code on failure
root	IN	root node or subtree to save
list	IN	NULL-terminated list of variable arguments

Returns

(ubig_ora) number of bytes written to destination

> **See Also:**
>
> XmlLoadDom()

XmlVersion()

Returns the version string for the XDK

Syntax

```
oratext *XmlVersion();
```

Returns

`(oratext *)` version string

12

Package XmlDiff for XML C APIs

The methods of the package `XmlDiff` allow you to compare and modify XML documents. The XmlDiff() and XmlPatch() methods are generally equivalent to UNIX commands `diff` and `patch`, and in addition are optimized for, and aware of, XML.

The following table summarizes the methods available through the `XmlDiff` package for XML C APIs.

Table 12-1 Summary of XmlDiff Methods for XML C Implementation

Function	Summary
XmlDiff()	Determines the changes between two XML documents.
XmlHash()	Computes a hash value for an XML document or a node in DOM.
XmlPatch()	Applies changes on input XML document.

> **See Also:**
>
> *Oracle XML Developer's Kit Programmer's Guide*

XmlDiff()

Determines the changes between two XML documents.

XmlDiff() captures the `diff` between two documents in an XML format that conforms to the `Xdiff` XML schema; you can customize this output.

These input documents can be specified either as DOM Trees, files, URI, `orastream`, and so on. DOM trees for both the inputs will be created if they are not supplied as DOM trees. The DOM for the diff document is created, and the doc node is returned.

If the caller supplies inputs as DOMs, the memory for the DOMs will not be freed.

Data (DOM) encoding of both documents must be the same as the data encoding in `xctx`. The diff DOM will be created in the data encoding specified in `xctx`.

There are four algorithms that can be run in XmlDiff(): global, local, global with hashing, and local with hashing. The `diff` may be different in the four cases.

The global algorithm will generate minimal `diff` using insert, append, delete and update operations. It needs more memory and time than the local algorithm. The local algorithm may not generate minimal diff, but is faster and uses less space than the global algorithm.

Hashing can be used with both global and local algorithms. If hashing is used with the global algorithm, it will speed up diff computation significantly, but may reduce the quality of diff. With local algorithm, it improves the quality of the `diff`.

You must specify a depth at which to use hashing. In hashing, the hash value for every element node is associated with a digest for the entire subtree rooted at that node. The tree is not investigated beyond the specified hash level depth while computing the `diff`.

The output of the global algorithm with or without hashing meets 'operations-in-docorder' requirement (the nodes must appear in same order as a preorder traversal of the document tree), but the output of the local algorithm does not.

The namespace prefixes XmlDiff() will use in the `xdiff` document may be same as those in either the first or second `doc`, depending on which prefix was seen first while processing. The NS URI will be bound to the prefix in the output appropriately. If this NS does not have a prefix in both `doc`s, a new prefix will be generated and bound to the NS in `xdiff doc`.

Syntax

```
xmldocnode *XmlDiff(
   xmlctx *xctx,
   xmlerr *err,
   ub4  flags,
   xmldfsrct firstSourceType,
   void *firstSource,
   void *firstSourceExtra,
   xmldfsrct secondSourceType,
   void *secondSource,
   void *secondSourceExtra,
   uword hashLevel,
   oraprop *properties);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`xmlerr`	`OUT`	numeric error code, `XMLERR_OK` on success
`flags`	`IN`	The following options are available: • `XMLDF_FL_DEFAULTS(=0)` chooses defaults • `XMLDF_FL_ALGORITHM_GLOBAL` is the global algorithm • `XMLDF_FL_ALGORITHM_LOCAL` is the local algorithm • `XMLDF_FL_DISABLE_UPDATE` indicates a disable update operation, with the global algorithm By default, global algorithm is used.
`firstSourceType`	`IN`	Type of source for first document; if zero, `firstSource` is assumed to be a DOM doc node.
`firstSource`	`IN`	Pointer to the source for the first document
`firstSourceExtra`	`IN`	An additional pointer to the source for the first document; used for buffer length pointer

Parameter	In/Out	Description
secondSourceType	IN	Type of source for second document; if zero, secondSource is assumed to be a DOM doc node.
secondSource	IN	Pointer to the source for the second document
secondSourceExtra	IN	An additional pointer to the source for the second document; used for buffer length pointer
hashLevel	IN	The depth (counting from 1 for the root) at which to use hashing for sub trees; <=1 means not to use hashing
properties	IN	Used for Output Builder

Returns

(xmldocnode) Doc node for the diff document, or NULL on error

XmlHash()

Computes a hash value for an XML document or a node in DOM.

If the hash values for two XML subtrees are equal, the corresponding subtrees are equal to a very high probability. Computes the hash value using the Message Digest algorithm 5 (MD5), a widely-used cryptographic hash function with a 128-bit hash value, so there is a very small probability that two different inputs might map to same MD5 digest.

The source can be specified as a file, a URL, and so on. It can also be a Document node in DOM, or any other DOM node, and must be specified using the inputSource parameter. If inputSource is a non-Document DOM node, inputSourceExtra must point to the Document node for the DOM.

Syntax

```
xmlerr XmlHash(
   xmlctx *xctx,
   xmlhasht *digest,
   ub4 flags,
   xmldfsrct iputSourceType,
   void *inputSource,
   void *inputSourceExtra,
   oraprop *properties);
```

Parameter	In/Out	Description
xctx	IN	XML context
digest	OUT	The hash value for the XML sub-tree
flags	IN	Not used
inputSourceType	IN	Type of source for the input document; if zero, inputSource is assumed to be a DOM doc node

Parameter	In/Out	Description
inputSource	IN	Pointer to the source for the input document
inputSourceExtra	IN	An additional pointer to the source for the input document; if used for a node pointer in a DOM, inputSource must be a document node.
properties	IN	Not used

Returns

(xmlerr) numeric error code, XMLERR_OK on success

XmlPatch()

XmlPatch() applies Xdiff schema-conforming changes to an input document. The input document and the diff document can be specified either as a DOM tree, file, URI, or buffer.

DOMs are built for both the input and diff document if they are not supplied as DOMs.

Data(DOM) encoding of both input and diff documents must be the same as the data encoding in xctx. The patched DOM will be in the data encoding specified in xctx.

Only the simple XPath is supported in the snapshot model. The XPath should identify a node with a posistion predicate in abbreviated syntax, such as /a[1]/b[2]. The XPaths generated by XmlDiff() meet this requirement. Also, 'operations-in-docorder' condition must be TRUE; the nodes must appear in same order as a preorder traversal of the document tree. Global (with or without hashing) meets this requirement. Local does not.

The programming interface should specify the output model used in the diff doc. The oracle-xmldif should be the first child of the top level xdiff element. It should also use flags to specify if operations are in document order (TRUE or FALSE), and wether the output model is a snapshot or current.

Syntax

```
xmldocnode *XmlPatch(
   xmlctx *xctx,
   xmlerr *err,
   ub4  flags,
   xmldfsrct inputSourceType,
   void *inputSource,
   void *inputSourceExtra,
   xmldfsrct diffSourceType,
   void *diffSource,
   void *diffSourceExtra,
   oraprop *properties);
```

Parameter	In/Out	Description
xctx	IN	XML context
xmlerr	OUT	numeric error code, XMLERR_OK on success
flags	IN	The following option is available: • XMLDF_FL_DEFAULTS(=0) chooses defaults
inputSourceType	IN	Type of source for the input document; if zero, inputSource is assumed to be a DOM doc node.
inputSource	IN	Pointer to the source for the input document
inputSourceExtra	IN	An additional pointer to the source for the input document; used for buffer length pointer
diffSourceType	IN	Type of source for diff document; if zero, secondSource is assumed to be a DOM doc node.
diffsSource	IN	Pointer to the source for the diff document
diffSourceExtra	IN	An additional pointer to the source for the diff document; used for buffer length pointer
properties	IN	Not used

Returns

(xmldocnode) Doc node for the pathed DOM, or NULL on error

13
Package XPath for XML C APIs

XPath methods process XPath related types and interfaces.

The following table summarizes the methods available through the `XPath` package for XML C APIs.

Table 13-1 Summary of XPath Methods for XML C Implementation

Function	Summary
XmlXPathCreateCtx()	Create an XPath context.
XmlXPathDestroyCtx()	Destroy an XPath context.
XmlXPathEval()	Evaluate XPath expression.
XmlXPathGetObjectBoolean()	Get boolean value of XPath object.
XmlXPathGetObjectFragment()	Get fragment value of XPath object.
XmlXPathGetObjectNSetNode()	Get node from nodeset type XPath object.
XmlXPathGetObjectNSetNum()	Get number of nodes in nodeset type XPath object.
XmlXPathGetObjectNumber()	Get number from XPath object.
XmlXPathGetObjectString()	Get string from XPath object.
XmlXPathGetObjectType()	Get XPath object type.
XmlXPathParse()	Parse XPath expression.

XmlXPathCreateCtx()

Create an `XPath` context

Syntax

```
xpctx* XmlXPathCreateCtx(
   xmlctx *xsl,
   oratext *baseuri,
   xmlnode *ctxnode,
   ub4 ctxpos,
   ub4 ctxsize);
```

Parameter	In/Out	Description
`xsl`	`IN`	XSL stylesheet as `xmldoc` object
`baseuri`	`IN`	base URI used by document, if any
`ctxnode`	`IN`	current context position

Parameter	In/Out	Description
ctxpos	IN	current context size
ctxsize	IN	current context node

Returns

(xpctx *) XPath context or NULL on error

XmlXPathDestroyCtx()

Destroy an XPath context.

Syntax

```
void XmlXPathDestroyCtx(
   xpctx *xslxpctx);
```

Parameter	In/Out	Description
xslxpctx	IN	XPath context object

XmlXPathEval()

Evaluate XPath expression.

Syntax

```
xpobj *XmlXPathEval(
   xpctx *xctx,
   xpexpr *exprtree,
   xmlerr *err);
```

Parameter	In/Out	Description
xctx	IN	XPath context
exprtree	IN	parsed XPath expression tree
err	OUT	error code

Returns

(xpobj *) result XPath object or NULL on error

XmlXPathGetObjectBoolean()

Get boolean value of `XPath` object

Syntax

```
boolean XmlXPathGetObjectBoolean(
   xpobj *obj);
```

Parameter	In/Out	Description
`obj`	`IN`	XPath object

Returns

`(boolean)` truth value

> **See Also:**
>
> XmlXPathGetObjectType(), XmlXPathGetObjectNSetNum(), XmlXPathGetObjectNSetNode(), XmlXPathGetObjectNumber(), XmlXPathGetObjectBoolean()

XmlXPathGetObjectFragment()

Get boolean value of `XPath` object

Syntax

```
xmlnode* XmlXPathGetObjectFragment(
   xpobj *obj);
```

Parameter	In/Out	Description
`obj`	`IN`	`XPath` object

Returns

`(boolean)` truth value

> **See Also:**
>
> XmlXPathGetObjectType(), XmlXPathGetObjectNSetNum(), XmlXPathGetObjectNSetNode(), XmlXPathGetObjectNumber(), XmlXPathGetObjectBoolean()

XmlXPathGetObjectNSetNode()

Get node from nodeset-type XPath object

Syntax

```
xmlnode *XmlXPathGetObjectNSetNode(
   xpobj *obj,
   ub4 i);
```

Parameter	In/Out	Description
obj	IN	XPath object
i	IN	node index in nodeset

Returns

(xmlnode *) The object type or values.

> **See Also:**
>
> XmlXPathGetObjectType(), XmlXPathGetObjectNSetNum(), XmlXPathGetObjectString(), XmlXPathGetObjectNumber(), XmlXPathGetObjectBoolean()

XmlXPathGetObjectNSetNum()

Get number of nodes in nodeset-type XPath object

Syntax

```
ub4 XmlXPathGetObjectNSetNum(
   xpobj *obj);
```

Parameter	In/Out	Description
obj	IN	XPath object

Returns

(ub4) number of nodes in nodeset

> **See Also:**
>
> XmlXPathGetObjectType(), XmlXPathGetObjectNSetNode(), XmlXPathGetObjectString(), XmlXPathGetObjectNumber(), XmlXPathGetObjectBoolean()

XmlXPathGetObjectNumber()

Get number from XPath object

Syntax

```
double XmlXPathGetObjectNumber(
   xpobj *obj);
```

Parameter	In/Out	Description
obj	IN	XPath object

Returns

(double) number

> **See Also:**
>
> XmlXPathGetObjectType(), XmlXPathGetObjectNSetNum(), XmlXPathGetObjectNSetNode(), XmlXPathGetObjectString(), XmlXPathGetObjectBoolean()

XmlXPathGetObjectString()

Get string from XPath object

Syntax

```
oratext *XmlXPathGetObjectString(
   xpobj *obj);
```

Parameter	In/Out	Description
obj	IN	XPath object

Returns

(oratext *) string

> **See Also:**
>
> XmlXPathGetObjectType(), XmlXPathGetObjectNSetNum(), XmlXPathGetObjectNSetNode(), XmlXPathGetObjectNumber(), XmlXPathGetObjectBoolean()

XmlXPathGetObjectType()

Get `XPath` object type

Syntax

```
xmlxslobjtype XmlXPathGetObjectType(
   xpobj *obj);
```

Parameter	In/Out	Description
`obj`	`IN`	XPath object

Returns

`(xmlxslobjtype)` type-code for object

> **See Also:**
>
> XmlXPathGetObjectNSetNum(), XmlXPathGetObjectNSetNode(), XmlXPathGetObjectString(), XmlXPathGetObjectNumber(), XmlXPathGetObjectBoolean()

XmlXPathParse()

Parse `XPath` expression.

Syntax

```
xpexpr* XmlXPathParse(
   xpctx *xctx,
   oratext *expr,
   xmlerr * err);
```

Parameter	In/Out	Description
`xctx`	`IN`	XPath context object
`expr`	`IN`	XPath expression

Parameter	In/Out	Description
`err`	`OUT`	error code

Returns

`(xpexpr *) XPath` expression parse tree or `NULL` on error

14

Package XPointer for XML C APIs

Package `XPointer` methods implement XML pointers for XML C APIs. They are grouped into three interface types:

- XPointer Interface
- XPtrLoc Interface
- XPtrLocSet Interface

XPointer Interface for XPointer XML C APIs

The following table summarizes the methods available through the `XPointer` interface of `XPointer` for XML C APIs.

Table 14-1 Summary of XPointer XPointer Methods for XML C Implementation

Function	Summary
XmlXPointerEval()	Evaluates XPointer string.

XmlXPointerEval()

Parses and evaluates xpointer string and calculates locations in the document.

Syntax

```
xmlxptrlocset* XmlXPointerEval(
   xmldocnode* doc,
   oratext* xptrstr);
```

Parameter	In/Out	Description
`doc`	`IN`	document node of the corresponding DOM tree
`xptrstr`	`IN`	xpointer string

Returns

`(xmlxptrlocset *)` calculated location set

XPtrLoc Interface for XPointer XML C APIs

The following table summarizes the methods available through the `XPtrLoc` interface of `XPointer` for XML C API.

Table 14-2 Summary of XPtrLoc XPointer Methods for XML C Implementation

Function	Summary
XmlXPtrLocGetNode()	Returns Xml node from `XPtrLoc`.
XmlXPtrLocGetPoint()	Returns Xml point from `XPtrLoc`.
XmlXPtrLocGetRange()	Returns Xml range from `XPtrLoc`.
XmlXPtrLocGetType()	Returns type of `XPtrLoc`.
XmlXPtrLocToString()	Returns string for a location.

XmlXPtrLocGetNode()

Returns node from location

Syntax

```
xmlnode* XmlXPtrLocGetNode(
   xmlxptrloc* loc);
```

Parameter	In/Out	Description
`loc`	`IN`	location

Returns

`(xmlnode *) Node` from location

XmlXPtrLocGetPoint()

Returns point from location

Syntax

```
xmlpoint* XmlXPtrLocGetPoint(
   xmlxptrloc* loc);
```

Parameter	In/Out	Description
`loc`	`IN`	location

Returns

`(xmlpoint *)` Point from location

XmlXPtrLocGetRange()

Returns range from location.

Syntax

```
xmlrange* XmlXPtrLocGetRange(
   xmlxptrloc* loc);
```

Parameter	In/Out	Description
loc	IN	location

Returns

(xmlrange *) Range from location

XmlXPtrLocGetType()

Returns type of location

Syntax

```
xmlxptrloctype XmlXPtrLocGetType(
   xmlxptrloc* loc);
```

Parameter	In/Out	Description
loc	IN	location

Returns

(xmlxptrloctype) Type of location

XmlXPtrLocToString()

Returns string for a location:

- node name: name of the container node

- names of container nodes: "not a location" otherwise

Syntax

```
oratext* XmlXPtrLocToString(
   xmlxptrloc* loc);
```

Parameter	In/Out	Description
loc	IN	location

Returns

(oratext *) string

XPtrLocSet Interface for XPointer XML C APIs

The following table summarizes the methods available through the `XPtrLocSet` interface of `XPointer` for XML C APIs.

Table 14-3 Summary of XPtrLocSet XPointer Methods for XML C Implementation

Function	Summary
XmlXPtrLocSetFree()	Free a location set
XmlXPtrLocSetGetItem()	Returns location with `idx` position in `XPtrLocSet`
XmlXPtrLocSetGetLength()	Returns length of `XPtrLocSet`.

XmlXPtrLocSetFree()

It is user's responsibility to call this function on every location set returned by `XPointer` or `XPtrLocSet` interfaces

Syntax

```
void XmlXPtrLocSetFree(
   xmlxptrlocset* locset);
```

Parameter	In/Out	Description
`locset`	`IN`	location set

XmlXPtrLocSetGetItem()

Returns location with idx position in the location set. First position is `1`.

Syntax

```
xmlxptrloc* XmlXPtrLocSetGetItem(
   xmlxptrlocset* locset,
   ub4 idx);
```

Parameter	In/Out	Description
`locset`	`IN`	location set
`idx`	`IN`	location index

Returns

`(xmlxptrloc *)` location with the position `idx`

XmlXPtrLocSetGetLength()

Returns the number of locations in the location set

Syntax

```
ub4 XmlXPtrLocSetGetLength(
   xmlxptrlocset* locset);
```

Parameter	In/Out	Description
`locset`	`IN`	location set

Returns

`(ub4)` number of nodes in `locset`

15
Package XSLT for XML C APIs

Package XSLT implements types and methods related to XSL processing.

The following table summarizes the methods available through the `XSLT` package for XML C APIs.

Table 15-1 Summary of XSLT Methods for XML C Implementation

Function	Summary
XmlXslCreate()	Create an XSL context.
XmlXslDestroy()	Destroy an XSL context.
XmlXslGetBaseURI()	Get the XSL processor base URI.
XmlXslGetOutput()	Get the XSL result fragment.
XmlXslGetStylesheetDom()	Get the XSL stylesheet document.
XmlXslGetTextParam()	Get the XSL text parameter value.
XmlXslProcess()	Perform XSL processing on an instance document.
XmlXslResetAllParams()	Reset XSL processor parameters.
XmlXslSetOutputDom()	Set the XSL context output DOM.
XmlXslSetOutputEncoding()	Set the XSL context output encoding.
XmlXslSetOutputMethod()	Set the XSL context output method.
XmlXslSetOutputSax()	Set the XSL context output SAX.
XmlXslSetOutputStream()	Set the XSL context output stream.
XmlXslSetTextParam()	Set the XSL context output text parameter.

XmlXslCreate()

Create an XSLT context

Syntax

```
xslctx *XmlXslCreate(
   xmlctx *ctx,
   xmldocnode *xsl,
   oratext *baseuri,
   xmlerr *err);
```

Parameter	In/Out	Description
`ctx`	`IN`	XSL context object
`xsl`	`IN`	XSL stylesheet document object

Parameter	In/Out	Description
baseuri	IN	base URI for including and importing documents
err	IN/OUT	returned error code

Returns

(xslctx *) XSLT context

> **See Also:**
>
> XmlXslDestroy()

XmlXslDestroy()

Destroy an XSL context

Syntax

```
xmlerr XmlXslDestroy(
   xslctx *ctx);
```

Parameter	In/Out	Description
ctx	IN	XSL context

Returns

(xmlerr) error code

> **See Also:**
>
> XmlXslCreate()

XmlXslGetBaseURI()

Get the XSL processor base URI

Syntax

```
oratext *XmlXslGetBaseURI(
   xslctx *ctx);
```

Parameter	In/Out	Description
ctx	IN	XSL context object

Returns

(oratext *) base URI

XmlXslGetOutput()

Get the XSL result fragment

Syntax

```
xmlfragnode *XmlXslGetOutput(
   xslctx *ctx);
```

Parameter	In/Out	Description
ctx	IN	XSL context object

Returns

(xmlfragnode *) result fragment

XmlXslGetStylesheetDom()

Get the XSL stylesheet document

Syntax

```
xmldocnode *XmlXslGetStylesheetDom(
   xslctx *ctx);
```

Parameter	In/Out	Description
ctx	IN	XSL context object

Returns

(xmldocnode *) stylesheet document

XmlXslGetTextParam()

Get the XSL text parameter value

Syntax

```
oratext *XmlXslGetTextParam(
   xslctx *ctx,
   oratext *name);
```

Parameter	In/Out	Description
ctx	IN	XML context object
name	IN	name of the top-level parameter value

Returns

(oratext *) parameter value

> **See Also:**
>
> XmlXslSetTextParam()

XmlXslProcess()

Do XSL processing on an instance document

Syntax

```
xmlerr XmlXslProcess(
   xslctx *ctx,
   xmldocnode *xml,
   boolean normalize);
```

Parameter	In/Out	Description
ctx	IN	XSL context object
xml	IN	instance document to process
normalize	IN	if TRUE, force the XSL processor to normalize the document

Returns

(xmlerr) error code

XmlXslResetAllParams()

Reset all the top level parameters added

Syntax

```
xmlerr XmlXslResetAllParams(
   xslctx *ctx);
```

Parameter	In/Out	Description
ctx	IN	XSL context object

Returns

(xmlerr) error code, XMLERR_SUCC [0] on success.

> **See Also:**
>
> XmlXslSetTextParam()

XmlXslSetOutputDom()

Set the xslctx output DOM

XmlXslSetOutputEncoding()

Set the xslctx output encoding

Syntax

```
xmlerr XmlXslSetOutputEncoding(
   xslctx *ctx,
   oratext* encoding);
```

Parameter	In/Out	Description
ctx	IN	XML context object
encoding	IN	output encoding

Returns

(xmlerr) error code, XMLERR_SUCC [0] on success.

XmlXslSetOutputMethod()

Set the xslctx output method

Syntax

```
xmlerr XmlXslSetOutputMethod(
   xslctx *ctx,
   xmlxslomethod method);
```

Parameter	In/Out	Description
ctx	IN	XML context object
encoding	IN	XSL output method

Returns

(xmlerr) error code, XMLERR_SUCC [0] on success.

XmlXslSetOutputSax()

Set the xslctx output SAX

Syntax

```
xmlerr XmlXslSetOutputSax(
   xslctx *ctx,
   xmlsaxcb* saxcb,
   void *saxctx);
```

Parameter	In/Out	Description
ctx	IN	XSL context object
saxcb	IN	SAX callback object
saxctx	IN	SAX callback context

Returns

(xmlerr) error code, XMLERR_SUCC [0] on success.

XmlXslSetOutputStream()

Syntax

```
xmlerr XmlXslSetOutputStream(
   xslctx *ctx,
   xmlostream *stream);
```

Parameter	In/Out	Description
ctx	IN	XSL context object
stream	IN	output stream object

Returns

`(xmlxsl)` error code, `XMLXSL_SUCC [0]` on success.

XmlXslSetTextParam()

Set the `xslctx` output text parameter.

Syntax

```
xmlerr XmlXslSetTextParam(
   xslctx *ctx,
   oratext *name,
   oratext *value);
```

Parameter	In/Out	Description
`ctx`	`IN`	XSL context object
`name`	`IN`	name of top level parameter
`value`	`IN`	value of top level parameter

Returns

`(xmlerr)` error code, `XMLERR_SUCC [0]` on success.

> **See Also:**
>
> XmlXslGetTextParam()

16

Package XSLTVM for XML C APIs

Package XSLTVM implements the XSL Transformation (XSLT) language for XML C APIs, as specified in W3C Recommendation 16 November 1999. For convenience, we grouped XSLTVM methods into two interface types.

- XSLTC Interface
- XSLTVM Interface

XSLTC Interface of XSLTVM for XML C APIs

The following table summarizes the methods available through the `XSLTC` interface of `XSLTVM` for XML C APIs.

Table 16-1 Summary of XSLTC XSLTVM Methods for XML C Implementation

Function	Summary
XmlXvmCompileBuffer()	Compile an XSLT stylesheet from buffer into bytecode.
XmlXvmCompileDom()	Compile an XSLT stylesheet from DOM into bytecode.
XmlXvmCompileFile()	Compile an XSLT stylesheet from file into bytecode.
XmlXvmCompileURI()	Compile XSLT stylesheet from URI into byte code.
XmlXvmCompileXPath()	Compile an XPath expression.
XmlXvmCreateComp()	Create an XSLT compiler.
XmlXvmDestroyComp()	Destroy an XSLT compiler object.
XmlXvmGetBytecodeLength()	Returns the bytecode length.

XmlXvmCompileBuffer()

Compile an XSLT stylesheet from buffer into bytecode. Compiler flags could be one or more of the following:

- `XMLXVM_DEBUG` forces compiler to include debug information into the bytecode
- `XMLXVM_STRIPSPACE` is equivalent to `<xsl:strip-space elements="*"/>`.

The generated bytecode resides in a compiler buffer which is freed when next stylesheet is compiled or when compiler object is deleted. Hence, if the bytecode is to be reused it should be copied into another location.

XmlXvmCompileDom()

Compile an XSLT stylesheet from DOM into bytecode. Compiler flags could be one or more of the following:

- `XMLXVM_DEBUG` forces compiler to include debug information into the bytecode
- `XMLXVM_STRIPSPACE` is equivalent to `<xsl:strip-space elements="*"/>`.

The generated bytecode resides in a compiler buffer which is freed when next stylesheet is compiled or when compiler object is deleted. Hence, if the bytecode is to be reused it should be copied into another location.

Syntax

```
ub2 *XmlXvmCompileDom(
   xmlxvmcomp *comp,
   xmldocnode *root,
   xmlxvmflag flags,
   xmlerr *error);
```

Parameter	In/Out	Description
`comp`	`IN`	compiler object
`rooot`	`IN`	root element of the stylesheet DOM
`flags`	`IN`	flags for the current compilation
`error`	`OUT`	returned error code

Returns

`(ub2 *)` bytecode or `NULL` on error

> **See Also:**
>
> XmlXvmCompileFile(), XmlXvmCompileBuffer(), XmlXvmCompileURI()

XmlXvmCompileFile()

Compile XSLT stylesheet from file into bytecode. Compiler flags could be one or more of the following:

- `XMLXVM_DEBUG` forces compiler to include debug information into the bytecode
- `XMLXVM_STRIPSPACE` is equivalent to `<xsl:strip-space elements="*"/>`.

The generated bytecode resides in a compiler buffer which is freed when next stylesheet is compiled or when compiler object is deleted. Hence, if the bytecode is to be reused it should be copied into another location.

Syntax

```
ub2 *XmlXvmCompileFile(
   xmlxvmcomp *comp,
   oratext *path,
   oratext *baseURI,
```

```
   xmlxvmflag flags,
   xmlerr *error);
```

Parameter	In/Out	Description
comp	IN	compiler object
path	IN	path of XSL stylesheet file
baseuri	IN	base URI of the document
flags	IN	flags for the current compilation
error	OUT	returned error code

Returns

(ub2 *) bytecode or NULL on error

> **See Also:**
>
> XmlXvmCompileURI(), XmlXvmCompileBuffer(), XmlXvmCompileDom()

XmlXvmCompileURI()

Compile XSLT stylesheet from URI into bytecode. Compiler flags could be one or more of the following:

- XMLXVM_DEBUG forces compiler to include debug information into the bytecode
- XMLXVM_STRIPSPACE is equivalent to <xsl:strip-space elements="*"/>.

The generated bytecode resides in a compiler buffer which is freed when next stylesheet is compiled or when compiler object is deleted. Hence, if the bytecode is to be reused it should be copied into another location.

Syntax

```
ub2 *XmlXvmCompileURI(
   xmlxvmcomp *comp,
   oratext *uri,
   xmlxvmflag flags,
   xmlerr *error);
```

Parameter	In/Out	Description
comp	IN	compiler object
uri	IN	URI of the file that contains the XSL stylesheet

Parameter	In/Out	Description
flags	IN	flags for the current compilation
error	OUT	returned error code

Returns

(ub2 *) bytecode or NULL on error

> **See Also:**
>
> XmlXvmCompileFile(), XmlXvmCompileBuffer(), XmlXvmCompileDom()

XmlXvmCompileXPath()

Compiles an XPath expression. The optional pfxmap is used to map namespace prefixes to URIs in the XPath expression. It is an array of prefix, URI values, ending in NULL, and so on.

Syntax

```
ub2 *XmlXvmCompileXPath(
   xmlxvmcomp *comp,
   oratext *xpath,
   oratext **pfxmap,
   xmlerr *error);
```

Parameter	In/Out	Description
comp	IN	compiler object
xpath	IN	XPath expression
pfxmap	IN	array of prefix-URI mappings
error	OUT	returned error code

Returns

(ub2 *) XPath expression bytecode or NULL on error

XmlXvmCreateComp()

Create an XSLT compiler object. The XSLT compiler is used to compile XSLT stylesheets into bytecode.

Syntax

```
xmlxvmcomp *XmlXvmCreateComp(
   xmlctx *xctx);
```

Parameter	In/Out	Description
xctx	IN	XML context

Returns

`(xmlxvmcomp *)` XSLT compiler object, or `NULL` on error

> **See Also:**
>
> XmlXvmDestroyComp()

XmlXvmDestroyComp()

Destroys an XSLT compiler object

Syntax

```
void XmlXvmDestroyComp(
   xmlxvmcomp *comp);
```

Parameter	In/Out	Description
comp	IN	XSLT compiler object

> **See Also:**
>
> XmlXvmCreateComp()

XmlXvmGetBytecodeLength()

The bytecode length is needed when the bytecode is to be copied or when it is set into XSLTVM.

Syntax

```
ub4 XmlXvmGetBytecodeLength(
   ub2 *bytecode,
   xmlerr *error);
```

Parameter	In/Out	Description
bytecode	IN	bytecode buffer
error	OUT	returned error code

Returns

(ub4) The bytecode length in bytes.

XSLTVM Interface of XSLTVM for XML C APIs

The following table summarizes the methods available through the XSLTVM interface of XSLTVM for XML C APIs.

Table 16-2 Summary of XSLTVM XSLTVM Methods for XML C Implementation

Function	Summary
XMLXVM_DEBUG_F()	XMLXSLTVM debug function.
XmlXvmCreate()	Create an XSLT virtual machine.
XmlXvmDestroy()	Destroys an XSLT virtual machine.
XmlXvmEvaluateXPath()	Evaluate already-compiled XPath expression.
XmlXvmGetObjectBoolean()	Get boolean value of XPath object.
XmlXvmGetObjectNSetNode()	Get node from nodeset type XPathobject.
XmlXvmGetObjectNSetNum()	Get number of nodes in nodeset type XPathobject.
XmlXvmGetObjectNumber()	Get number from XPath object.
XmlXvmGetObjectString()	Get string from XPath object.
XmlXvmGetObjectType()	Get XPath object type.
XmlXvmGetOutputDom()	Returns the output DOM.
XmlXvmResetParams()	Resets the stylesheet top level text parameters.
XmlXvmSetBaseURI()	Sets the base URI for the XLTVM.
XmlXvmSetBytecodeBuffer()	Set the compiled bytecode.
XmlXvmSetBytecodeFile()	Set the compiled byte code from file.
XmlXvmSetBytecodeURI()	Set the compiled bytecode.
XmlXvmSetDebugFunc()	Set a callback function for debugging.
XmlXvmSetOutputDom()	Sets the XSLTVM to output document node.
XmlXvmSetOutputEncoding()	Sets the encoding for the XSLTVM output.
XmlXvmSetOutputSax()	Sets XSLTVM to output SAX.
XmlXvmSetOutputStream()	Set the XSLTVM output to a user-defined stream.
XmlXvmSetTextParam()	Set the stylesheet top-level text parameter.
XmlXvmTransformBuffer()	Run compiled XSLT stylesheet on XML document in memory.

Table 16-2 (Cont.) Summary of XSLTVM XSLTVM Methods for XML C Implementation

Function	Summary
XmlXvmTransformDom()	Run compiled XSLT stylesheet on XML document as DOM.
XmlXvmTransformFile()	Run compiled XSLT stylesheet on XML document in file.
XmlXvmTransformURI()	Run compiled XSLT stylesheet on XML document from URI.

XMLXVM_DEBUG_F()

Debug callback function for XSLT VM.

Syntax

```
#define XMLXVM_DEBUG_F(func, line, file, obj, n)
void func(
   ub2 line,
   oratext *file,
   xvmobj *obj,
   ub4 n)
```

Parameter	In/Out	Description
line	IN	source stylesheet line number
file	IN	stylesheet filename
obj	IN	current VM object
n	IN	index of current node

> **See Also:**
>
> XmlXvmSetDebugFunc()

XmlXvmCreate()

Create an XSLT virtual machine. Zero or more of the following XSLTVM properties could be set by using this API:

- `"VMStack", size` sets the size[Kbyte] of the main VM stack; default size is 4K.
- `"NodeStack", size` sets the size[Kbyte] of the node-stack; default size is 16K.
- `"StringStack", size` sets the size[Kbyte] of the string-stack; default size is 64K.

If the stack size is not specified the default size is used. The explicit stack size setting is needed when XSLTVM terminates with "Stack Overflow" message or when smaller memory footprints are required.

Syntax

```
xmlxvm *XmlXvmCreate(
   xmlctx *xctx,
   list);
```

Parameter	In/Out	Description
`xctx`	`IN`	XML context
`list`	`IN`	`NULL`-terminated list of properties to set; can be empty

Returns

`(xmlxvm *)` XSLT virtual machine object, or `NULL` on error

> **See Also:**
>
> XmlXvmDestroy()

XmlXvmDestroy()

Destroys an XSLT virtual machine

Syntax

```
void XmlXvmDestroy(
   xmlxvm *vm);
```

Parameter	In/Out	Description
`vm`	`IN`	VM object

> **See Also:**
>
> XmlXvmCreate()

XmlXvmEvaluateXPath()

Evaluate already-compiled XPath expression

Syntax

```
xvmobj *XmlXvmEvaluateXPath(
   xmlxvm *vm,
   ub2 *bytecode,
   ub4 ctxpos,
   ub4 ctxsize,
   xmlnode *ctxnode);
```

Parameter	In/Out	Description
vm	IN	XSLTVM object
bytecode	IN	XPath expression bytecode
ctxpos	IN	current context position
ctxsize	IN	current context size
ctxnode	IN	current context node

Returns

(xvmobj *) XPath object

XmlXvmGetObjectBoolean()

Get boolean value of XPath object

Syntax

```
boolean XmlXvmGetObjectBoolean(
   xvmobj *obj);
```

Parameter	In/Out	Description
obj	IN	object

Returns

(boolean) value of an XPath object

> **See Also:**
>
> XmlXvmGetObjectType(), XmlXvmGetObjectNSetNum(), XmlXvmGetObjectNSetNode(), XmlXvmGetObjectNumber(), XmlXvmGetObjectBoolean()

XmlXvmGetObjectNSetNode()

Get node from nodeset-type XPath object

Syntax

```
xmlnode *XmlXvmGetObjectNSetNode(
   xvmobj *obj,
   ub4 i);
```

Parameter	In/Out	Description
obj	IN	object
i	IN	node index in nodeset

Returns

(xmlnode *) The object type or values.

> **See Also:**
>
> XmlXvmGetObjectType(), XmlXvmGetObjectNSetNum(), XmlXvmGetObjectString(), XmlXvmGetObjectNumber(), XmlXvmGetObjectBoolean()

XmlXvmGetObjectNSetNum()

Get number of nodes in nodeset-type XPath object

Syntax

```
ub4 XmlXvmGetObjectNSetNum(
   xvmobj *obj);
```

Parameter	In/Out	Description
obj	IN	object

Returns

(ub4) number of nodes in nodeset

> **See Also:**
>
> XmlXvmGetObjectType(), XmlXvmGetObjectNSetNode(), XmlXvmGetObjectString(), XmlXvmGetObjectNumber(), XmlXvmGetObjectBoolean()

XmlXvmGetObjectNumber()

Get number from XPath object.

Syntax

```
double XmlXvmGetObjectNumber(
   xvmobj *obj);
```

Parameter	In/Out	Description
obj	IN	object

Returns

(double) number

> **See Also:**
>
> XmlXvmGetObjectType(), XmlXvmGetObjectNSetNum(), XmlXvmGetObjectNSetNode(), XmlXvmGetObjectString(), XmlXvmGetObjectBoolean()

XmlXvmGetObjectString()

Get string from XPath object.

Syntax

```
oratext *XmlXvmGetObjectString(
   xvmobj *obj);
```

Parameter	In/Out	Description
obj	IN	object

Returns

(oratext *) string

> **See Also:**
>
> XmlXvmGetObjectType(), XmlXvmGetObjectNSetNum(), XmlXvmGetObjectNSetNode(), XmlXvmGetObjectNumber(), XmlXvmGetObjectBoolean()

XmlXvmGetObjectType()

Get XPath object type

Syntax

```
xmlxvmobjtype XmlXvmGetObjectType(
   xvmobj *obj);
```

Parameter	In/Out	Description
obj	IN	object

Returns

(xmlxvmobjtype) type-code for object

> **See Also:**
>
> XmlXvmGetObjectNSetNum(), XmlXvmGetObjectNSetNode(), XmlXvmGetObjectString(), XmlXvmGetObjectNumber(), XmlXvmGetObjectBoolean()

XmlXvmGetOutputDom()

Returns the root node of the result DOM tree (if any). XmlXvmSetOutputDom has to be used before transformation to set the VM to output a DOM tree (the default VM output is a stream).

Syntax

```
xmlfragnode *XmlXvmGetOutputDom(
   xmlxvm *vm);
```

Parameter	In/Out	Description
vm	IN	VM object

Returns

(xmlfragnode *) output DOM, or NULL in a case of SAX or Stream output.

> **See Also:**
>
> XmlXvmSetOutputDom()

XmlXvmResetParams()

Resets the stylesheet top-level parameters with their default values.

Syntax

```
void XmlXvmResetParams(
   xmlxvm *vm);
```

Parameter	In/Out	Description
vm	IN	VM object

XmlXvmSetBaseURI()

Sets the base URI for the XSLTVM. The baseuri is used by VM to the compose the path XML documents to be loaded for transformation using `document` or `XmlXvmTransformFile`.

Syntax

```
xmlerr XmlXvmSetBaseURI(
   xmlxvm *vm,
   oratext *baseuri);
```

Parameter	In/Out	Description
vm	IN	VM object
baseuri	IN	VM base URI for reading and writing documents

Returns

`(xmlerr)` error code.

XmlXvmSetBytecodeBuffer()

Set the compiled bytecode from buffer. Any previously set bytecode is replaced. An XML transformation can't be performed if the stylesheet bytecode is not set. The VM doesn't copy the bytecode into internal buffer, hence the it shouldn't be freed before VM finishes using it.

Syntax

```
xmlerr XmlXvmSetBytecodeBuffer(
   xmlxvm *vm,
   ub2 *buffer,
   size_t buflen);
```

Parameter	In/Out	Description
vm	IN	XSLT VM context
buffer	IN	user's buffer
buflen	IN	size of buffer, in bytes

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlXvmSetBytecodeFile(), XmlXvmSetBytecodeURI()

XmlXvmSetBytecodeFile()

Set the compiled bytecode from file. Any previously set bytecode is replaced. An XML transformation can't be performed if the stylesheet bytecode is not set.

Syntax

```
xmlerr XmlXvmSetBytecodeFile(
   xmlxvm *vm,
   oratext *path);
```

Parameter	In/Out	Description
vm	IN	XSLT VM context
path	IN	path of bytecode file

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlXvmSetBytecodeURI(), XmlXvmSetBytecodeBuffer()

XmlXvmSetBytecodeURI()

Set the compiled bytecode from URI. Any previously set bytecode is replaced. An XML transformation can't be performed if the stylesheet bytecode is not set.

Syntax

```
xmlerr XmlXvmSetBytecodeURI(
   xmlxvm *vm,
   oratext *uri);
```

Parameter	In/Out	Description
vm	IN	XSLT VM context
uri	IN	path of bytecode file

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

> **See Also:**
>
> XmlXvmSetBytecodeFile(), XmlXvmSetBytecodeBuffer()

XmlXvmSetDebugFunc()

The user callback function is invoked by VM every time the execution reaches a new line in the XSLT stylesheet. The VM passes to the user the stylesheet file name, the line number, the current context nodes-set and the current node index in the node-set. IMPORTANT - the stylesheet has to be compiled with flag XMLXVM_DEBUG.

Syntax

```
#define XMLXVM_DEBUG_FUNC(func)
void func (ub2 line, oratext *filename, xvmobj *obj, ub4 n)
xmlerr XmlXvmSetDebugFunc(
   xmlxvm *vm,
   XMLXVM_DEBUG_FUNC(debugcallback));
```

Parameter	In/Out	Description
vm	IN	XSLT VM context
func	IN	callback function

Returns

(xmlerr) numeric error code, XMLERR_OK [0] on success

XmlXvmSetOutputDom()

Sets the XSLTVM to output DOM. If xmldocnode==NULL, then the result DOM tree belongs to the VM object and is deleted when a new transformation is performed or when the VM object

is deleted. If the result DOM tree is to be used for longer period of time then an `xmldocnode` has to be created and set to the VM object.

Syntax

```
xmlerr XmlXvmSetOutputDom(
   xmlxvm *vm,
   xmldocnode *doc);
```

Parameter	In/Out	Description
`vm`	`IN`	VM object
`doc`	`IN`	empty document

Returns

`(xmlerr)` error code

XmlXvmSetOutputEncoding()

Sets the encoding for the XSLTVM stream output. If the input (data) encoding is different from the one set by this APIs then encoding conversion is performed. This APIs overrides the encoding set in the XSLT stylesheet (if any).

Syntax

```
xmlerr XmlXvmSetOutputEncoding(
   xmlxvm *vm,
   oratext *encoding);
```

Parameter	In/Out	Description
`vm`	`IN`	VM object
`encoding`	`IN`	output encoding

Returns

`(xmlerr)` error code.

XmlXvmSetOutputSax()

Set XSLTVM to output SAX. If the SAX callback interface object is provided the VM outputs the result document in a form of SAX events using the user specified callback functions.

Syntax

```
xmlerr XmlXvmSetOutputSax(
   xmlxvm *vm,
   xmlsaxcb *saxcb,
   void *saxctx);
```

Parameter	In/Out	Description
vm	IN	VM object
saxcb	IN	SAX callback object
saxctx	IN	SAX context

Returns

(xmlerr) error code

XmlXvmSetOutputStream()

Set the XSLTVM output to a user-defined stream. The default XSLTVM output is a stream. This APIs overrides the default stream with user specified APIs for writing.

Syntax

```
xmlerr XmlXvmSetOutputStream(
   xmlxvm *vm,
   xmlostream *ostream);
```

Parameter	In/Out	Description
vm	IN	VM object
ostream	IN	stream object

Returns

(xmlerr) error code.

XmlXvmSetTextParam()

Set the stylesheet top-level text parameter. The parameter value set in the XSLT stylesheet is overwritten. Since the top-level parameters are reset with stylesheet values after each transformation, this APIs has to be called again.

Syntax

```
xmlerr XmlXvmSetTextParam(
   xmlxvm *vm,
   oratext *name,
   oratext *value);
```

Parameter	In/Out	Description
vm	IN	VM object

Parameter	In/Out	Description
name	IN	name of top-level parameter
value	IN	value of top-level parameter

Returns

(xmlerr) error code, XMLERR_SUCC [0] on success.

XmlXvmTransformBuffer()

Run compiled XSLT stylesheet on XML document in memory. The compiled XSLT stylesheet (bytecode) should be set using XmlXvmSetBytecodeXXX prior to this call.

Syntax

```
xmlerr XmlXvmTransformBuffer(
   xmlxvm *vm,
   oratext *buffer,
   ub4 length,
   oratext *baseURI);
```

Parameter	In/Out	Description
vm	IN	VM object
buffer	IN	NULL-terminated buffer that contains the XML document
length	IN	length of the XML document
baseURI	IN	base URI of XML document

Returns

(xmlerr) error code.

> **See Also:**
>
> XmlXvmTransformFile(), XmlXvmTransformURI(), XmlXvmTransformDom()

XmlXvmTransformDom()

Run compiled XSLT stylesheet on XML document as DOM. The compiled XSLT stylesheet (bytecode) should be set using XmlXvmSetBytecodeXXX prior to this call.

Syntax

```
xmlerr XmlXvmTransformDom(
   xmlxvm *vm,
   xmldocnode *root);
```

Parameter	In/Out	Description
vm	IN	VM object
root	IN	root element of XML document's DOM

Returns

(xmlerr) error code.

> **See Also:**
>
> XmlXvmTransformFile(), XmlXvmTransformURI(), XmlXvmTransformBuffer()

XmlXvmTransformFile()

Run compiled XSLT stylesheet on XML document in file. The compiled XSLT stylesheet (bytecode) should be set using XmlXvmSetBytecodeXXX prior to this call.

Syntax

```
xmlerr XmlXvmTransformFile(
   xmlxvm *vm,
   oratext *path,
   oratext *baseURI);
```

Parameter	In/Out	Description
vm	IN	VM object
path	IN	path of XML document to transform
baseURI	IN	base URI of XML document

Returns

(xmlerr) error code

> **See Also:**
>
> XmlXvmTransformURI(), XmlXvmTransformBuffer(), XmlXvmTransformDom()

XmlXvmTransformURI()

Run compiled XSLT stylesheet on XML document from URI. The compiled XSLT stylesheet (bytecode) should be set using `XmlXvmSetBytecodeXXX` prior to this call.

Syntax

```
xmlerr XmlXvmTransformURI(
   xmlxvm *vm,
   oratext *uri);
```

Parameter	In/Out	Description
`vm`	`IN`	VM object
`uri`	`IN`	URI of XML document to transform

Returns

`(xmlerr)` error code.

> **See Also:**
>
> XmlXvmTransformFile(), XmlXvmTransformBuffer(), XmlXvmTransformDom()

17
JSON DOM Interfaces

The JSON Interface for DOM C APIs are grouped into the following sections:

- JSON DOM Functions
- Constants

JSON DOM Functions

The following table summarizes the methods available through the `JSON` interface of DOM for C APIs.

Table 17-1 Summary of DOM Methods for JSON C Implementation

Function	Summary
JsonDomGetRoot	Returns the root node of a DOM, which is one of the three types of nodes.
JsonDomSetRoot	Sets the specified node as the root.
JsonDomGetNumFields	Returns the number of fields within the object.
JsonDomGetArraySize	Returns the number of elements within the array.
JsonDomGetElements	Returns a set of elements from an array in a batch interface.
JsonDomGetNodeType	Returns the type of the node.
JsonDomGetScalarInfoOci	Returns the value information from the scalar node.
JsonDomCreateObject	Creates a new orphan DOM object node.
JsonDomCreateOCIInterval	Creates a new DOM scalar node of OCIInterval type.
JsonDomCreateArray	Creates a new orphan DOM array node.
JsonDomCreateString	Creates a new orphan DOM scalar node of string type.
JsonDomCreateBinary	Creates a new orphan DOM scalar node of binary type.
JsonDomCreateFloat	Creates a new orphan DOM scalar node of float type.
JsonDomCreateDouble	Creates a new orphan DOM scalar node of double type.
JsonDomCreateBoolean	Creates a new orphan DOM scalar node of boolean type.
JsonDomCreateOCINumber	Creates a new orphan DOM scalar node of `NUMBER` type from an OCINumber input.
JsonDomCreateOCIDate	Creates a new orphan DOM scalar node of date type.
JsonDomCreateOCIDateTime	Create a new orphan DOM scalar node of type `DATE`, `TIMESTAMP`, or `TIMESTAMP WITH TIME ZONE` from an OCIDateTime type.

Table 17-1 (Cont.) Summary of DOM Methods for JSON C Implementation

Function	Summary
JsonDomCreateNull	Creates a new orphan DOM scalar node of null type.
JsonDomSetField	Sets the value of the specified field to the specified object within DOM.
JsonDomAppendElement	Append an orphan node as a new element in an array, at the last position within the array.
JsonDomFreeNode	Free an orphan node and all its descendants.
JsonDomGetError	Returns an error code for a DOM operation that failed.
JsonDomClearError	Clear the error code for the DOM.
JsonDomGetErrorMessage	Returns a rendered error message for the current DOM error code.

JsonDomGetRoot()

Returns the root node of a DOM, which is one of the three types of nodes (although rare, a scalar with no children is a valid JSON document per the RFC).

Syntax

```
JsonDomNode *JsonDomGetRoot(
      JsonDomDoc    *jdoc
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container

Returns

`(JsonDomNode *)` root node for the DOM (or NULL if none is set).

> **Note:**
>
> - The orphan nodes cannot be accessed by descending the tree starting from the root.
> - Check the error code by calling `JsonDomGetGetError()` before proceeding.

JsonDomSetRoot()

Sets the specified node as the root.

The new node may be a NULL. The new root must be an orphan node, that is, it cannot be a node that exists within the tree descending from the old root, if any.

Orphan nodes are created by constructors, or by copying node(s) from another DOM document, or by unlinking them from the DOM tree.

Syntax

```
void  JsonDomSetRoot(
    JsonDomDoc     *jdoc,
    JsonDomNode    *root
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
root	IN	Orphan node within the DOM

Returns

(ub4) A count of the number of fields within the object.

> **Note:**
>
> - The old root and all descendants are freed.
> - Check the error code by calling JsonDomGetGetError() before proceeding.

JsonDomGetNumFields()

Returns the number of fields within the object.

Syntax

```
ub4 JsonDomGetNumFields(
    JsonDomDoc     *jdoc,
    JsonDomObject  *obj
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
obj	IN	Object node within the DOM

Returns

(ub4) A count of the number of fields within this object.

JsonDomGetArraySize()

Returns the number of elements within the array.

Orphan nodes are created by constructors, or by copying node(s) from another DOM document, or by unlinking them from the DOM tree.

Syntax

```
ub4 JsonDomGetArraySize(
    JsonDomDoc *jdoc,
    JsonDomArray *ary
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container
`ary`	`IN`	Array node within the DOM

Returns

`(ub4)` A count of the number of elements within this array.

> **Note:**
>
> - Duplicate field names are not allowed in most DOMs.
> - Check the error code by calling `JsonDomGetGetError()` before proceeding.

JsonDomGetElements()

Returns a set of elements from an array in a batch interface.

This allows a caller to use a fixed-sized array to iterate over the elements, while still retrieving them in bulk.

Syntax

```
ub4 JsonDomGetElements(
  JsonDomDoc      *jdoc,
  JsonDomArray    *ary,
  ub4              startPos,
  ub4              fetchSz,
  JsonDomNode     *ndary[]
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
ary	IN	Array node within the DOM
startPos	IN	Starting position (0-based)
fetchSz	IN	Number of elements to retrieve
ndary	OUT	Returns nodes from the array

Returns

(ub4) The number of elements returned in the output node array.

> **Note:**
>
> - The first element at start position 0. The iteration is backed directly by the DOM array, so this interface won't work properly if the array undergoes modification during the iteration.
> - Check the error code by calling JsonDomGetGetError() before proceeding.

JsonDomGetNodeType()

Returns the type of a node.

The possible return values are:

- JZNDOM_SCALAR
- JZNDOM_ARRAY
- JZNDOM_OBJECT

The node may be safely cast to the subclass type. For example, if the return is JZNDOM_SCALAR, the node may be cast to (JsonDomScalar *).

Syntax

```
jznnodetype JsonDomGetNodeType(
  JsonDomDoc     *jdoc,
  JsonDomNode    *node
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container

Parameter	In/Out	Description
node	IN	Node within the DOM

Returns

(jznnodetype) the type of node (Object, Array, Scalar).

> **Note:**
>
> Check the error code by calling JsonDomGetGetError() before proceeding.

JsonDomGetScalarInfoOci()

Returns a scalar value in the val structure that must be applied by the caller, often a pointer to a local variable in the calling code.

The return structure is a discriminated union. The value's scalar subtype (for example, JZNVAL_NUMBER) is available as the first member, and the scalar's actual value can be read from the appropriate union member.

Syntax

```
jsonerr  JsonDomGetScalarInfoOci(
  JsonDomDoc       *jdoc,
  JsonDomScalar    *nd,
  jznScalarVal     *val,
  JsonOCIVal       *aux
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
nd	IN	Scalar node within the DOM
val	OUT	Output scalar value
aux	OUT	Auxiliary output structure

Returns

(jsonerr) An error code. JZNERR_OK (0) indicates no error.

Note:

- Oracle native types are unpickled and returned in structured form in auxiliary union of type `JsonOCIVal*`.
- Check the error code by calling `JsonDomGetGetError()` before proceeding.

JsonDomCreateObject()

Creates a new DOM object node.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomObject *JsonDomCreateObject(
  JsonDomDoc    *jdoc
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container

Returns

`(JsonDomObject *)` new orphaned DOM object node.

JsonDomCreateOCIInterval()

Creates a new DOM scalar node of OCIInterval type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomScalar *JsonDomCreateOCIInterval(
  JsonDomDoc     *jdoc,
  OCIInterval    *oinv
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container
`oinv`	`IN`	OCIInterval Pointer

Returns

`(JsonDomScalar *)` new orphaned DOM scalar node.

JsonDomCreateArray()

Creates a new DOM array node.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomArray *JsonDomCreateArray(
  JsonDomDoc *jdoc
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container

Returns

`(JsonDomArray *)` new orphaned DOM array node.

JsonDomCreateString()

Creates a new DOM scalar node of string type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node. The scalar is set to the value of the string argument `sval`, which is of length `slen`.

Syntax

```
JsonDomScalar *JsonDomCreateString(
  JsonDomDoc      *jdoc,
  oratext         *sval,
  ub4              slen
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
sval	IN	String buffer pointer
slen	IN	Length of the string buffer in bytes

Returns

`(JsonDomScalar *)` new orphaned DOM scalar node

> **Note:**
>
> The string is copied into the DOM's memory space.

JsonDomCreateBinary()

Creates a new DOM scalar node of binary type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node. The scalar is set to the contents of the byte array `bval` of length `blen`.

Syntax

```
JsonDomScalar *JsonDomCreateBinary(
  JsonDomDoc      *jdoc,
  oratext         *bval,
  ub4              blen
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container
`bval`	`IN`	Binary buffer pointer
`blen`	`IN`	Length of the binary buffer

Returns

`(JsonDomScalar *)` new orphaned DOM scalar node

> **Note:**
>
> The binary value is copied into the DOM's memory space.

JsonDomCreateFloat()

Creates a new DOM scalar node of float type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node. The scalar is set to the float value `fval`.

Syntax

```
JsonDomScalar *JsonDomCreateFloat(
  JsonDomDoc      *jdoc,
  float            fval
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
fval	IN	Float value

Returns

(JsonDomScalar *) new orphaned DOM scalar node

JsonDomCreateDouble()

Creates a new DOM scalar node of double type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node. The scalar value is set to the double value dval.

Syntax

```
JsonDomScalar *JsonDomCreateDouble(
  JsonDomDoc      *jdoc,
  double           dval
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
dval	IN	Double value

Returns

(JsonDomScalar *) new orphaned DOM scalar node

JsonDomCreateBoolean()

Creates a new DOM scalar node of boolean type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomScalar *JsonDomCreateBoolean(
  JsonDomDoc      *jdoc,
  boolean          bval
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
bval	IN	Boolean value

Returns

(JsonDomScalar *) new orphaned DOM scalar node

JsonDomCreateOCINumber()

Creates a new DOM scalar node of NUMBER type from an OCINumber.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomScalar *JsonDomCreateOCINumber(
  JsonDomDoc      *jdoc,
  OCINumber       *onum
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
onum	IN	OCINumber pointer

Returns

(JsonDomScalar *) new orphaned DOM scalar node

JsonDomCreateOCIDate()

Creates a new DOM scalar node of DATE type from an OCIDate.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomScalar *JsonDomCreateOCIDate(
  JsonDomDoc      *jdoc,
  OCIDate         *odate
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
odate	IN	OCIDate pointer

Returns

`(JsonDomScalar *)` new orphaned DOM scalar node

JsonDomCreateOCIDateTime()

Creates a new DOM scalar node of `DATE`, `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` type from an OCIDateTime.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomScalar *JsonDomCreateOCIDateTime(
  JsonDomDoc      *jdoc,
  OCIDateTime     *odtime
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
odtime	IN	OCIDateTime pointer

Returns

`(JsonDomScalar *)` new orphaned DOM scalar node

JsonDomCreateNull()

Creates a new DOM scalar node of `JSON null` type.

The new node is an orphan which is not attached to the DOM tree that descends from the root node.

Syntax

```
JsonDomScalar *JsonDomCreateNull(
  JsonDomDoc      *jdoc
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container

Returns

(JsonDomScalar *) new orphaned DOM scalar node

JsonDomSetField()

Sets the value of the specified field to the specified object within DOM.

The field is created if it doesn't exist, otherwise the old value is replaced.

Syntax

```
jsonerr JsonDomSetField(
  JsonDomDoc       *jdoc,
  JsonDomObject    *obj,
  oratext          *name,
  ub2               namelen,
  JsonDomNode      *node);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container
obj	IN	Object within the DOM
name	IN	String with field name to create or replace
namelen	IN	Length of field name string
node	IN	Orphan node to set as field value

Returns

(jsonerr) An error code. JZNERR_OK (0) indicates no error.

> **Note:**
>
> The old field value, if any, is freed along with all descendants.

JsonDomAppendElement()

Appends an orphan node as a new element in an array, at the last position within the array.

This function is equivalent to calling `JsonDomAddElement` and giving the array size as the insertion position.

Syntax

```
boolean JsonDomAppendElement(
  JsonDomDoc      *jdoc,
  JsonDomArray    *arr,
  JsonDomNode     *node
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container
`arr`	`IN`	Array within the DOM
`node`	`IN`	Orphan node to set an element value

Returns

`(boolean)`. `TRUE` if the element was appended, else `FALSE`.

> **Note:**
>
> - Check the error code by calling `JsonDomGetGetError()` before proceeding.

JsonDomFreeNode()

Free an orphan node and all descendants.

Syntax

```
boolean JsonDomFreeNode(
  JsonDomDoc     *jdoc,
  JsonDomNode    *node
);
```

Parameter	In/Out	Description
`jdoc`	`IN`	JSON DOM Container

Parameter	In/Out	Description
node	IN	Node to free within the DOM

Returns

(boolean). TRUE if freed, FALSE if not an orphan.

> **Note:**
>
> The old field value, if any, is freed along with all descendants.

JsonDomGetError()

Returns an error code for a DOM operation that failed.

Syntax

```
jsonerr JsonDomGetError(
  JsonDomDoc     *jdoc
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container

Returns

(jsonerr) An error code. JZNERR_OK (0) indicates no error.

JsonDomGetErrorMessage()

Returns a rendered error message for the current DOM error code.

Syntax

```
oratext *JsonDomGetErrorMessage(
  JsonDomDoc    *jdoc,
  jsonerr       *err,
  boolean        clear
);
```

Returns an empty string if there is no current error. Returns the current error code in the optional <err> argument. If <clear> is TRUE, the internal error in the document is cleared.

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container

Parameter	In/Out	Description
err	OUT	Error code or JZNERR_OK (0) for none
clear	IN	TRUE to clear error after reading

Returns

(oratext *) error message string, NULL on memory error.

JsonDomClearError()

Clear the error code for the DOM.

Syntax

```
void JsonDomClearError(
  JsonDomDoc     *jdoc
);
```

Parameter	In/Out	Description
jdoc	IN	JSON DOM Container

Returns

(void)

Constants

The below tables list the different types of constants in JSON.

DOM Node Type Constants

Constant	Description
JZNDOM_OBJECT	Indicates that the DOM node is of object type
JZNDOM_ARRAY	Indicates that the DOM node is of array type
JZNDOM_SCALAR	Indicates that the DOM node is of scalar type

Scalar Value Type Constants

Constant	JSON Value Type
JZNVAL_STRING	String

Constant	JSON Value Type
JZNVAL_ORA_NUMBER	Number
JZNVAL_TRUE	Boolean true
JZNVAL_FALSE	Boolean false
JZNVAL_BINARY	Binary
JZNVAL_DOUBLE	Double
JZNVAL_FLOAT	Float
JZNVAL_ORA_DATE	Date
JZNVAL_ORA_TIMESTAMP	Timestamp
JZNVAL_ORA_DAYSECOND_DUR	Day-second Interval
JZNVAL_ORA_YEARMONTH_DUR	Year-month interval
JZNVAL_ORA_TIMESTAMPTZ	Timestamp with time zone
JZNVAL_NULL	JSON null

Note:

The constants JZNVAL_DOUBLE and JZNVAL_FLOAT map to IEEE FLOAT and DOUBLE respectively, which are most commonly used in programming languages.

JSON Validation Constants

Constant	Description
JZN_ALLOW_NONE	Strict conformance to JSON specification.
JZN_ALLOW_UNQUOTED_NAMES	Object field names can appear without being enclosed in double quotes. Example: {a: 123}
JZN_ALLOW_SINGLE_QUOTES	Field names and string items can optionally be enclosed in single quote marks. Example: {'a': 'cat'}
JZN_ALLOW_UNQUOTED_UNICODE	Allow non-ASCII characters to appear in unquoted field names.

Constant	Description
JZN_ALLOW_NUMERIC_QUIRKS	Allow common numeric formats that are not strictly conforming to JSON specification. Example: +.123
JZN_ALLOW_CONTROLS_WHITESPACE	Allow ASCII control characters to appear where insignificant whitespace is allowed (normally restricted to tab, newline, carriage-return). Example: a form-feed character
JZN_ALLOW_TRAILING_COMMAS	Allow a trailing comma to appear in a container without a following item or field/value pair. Example: [1,2,3,]
JZN_ALLOW_MIXEDCASE_KEYWORDS	Allow keywords to appear in any case. Normally they are required to be in lower case. Examples: NULL, True
JZN_ALLOW_SCALAR_DOCUMENTS	Allow a single scalar item to be considered a valid JSON document. The older RFC required that a document be one of the two container types. The newer RFC allows any item, which includes scalars.
JZN_ALLOW_LEADING_BOM	Allow a byte-order-mark to appear as the first character of a JSON document, even if the document is encoded in UTF-8. Normally it's restricted to UTF-16 documents only.
JZN_ALLOW_ALL	Turns on all of the above flags.

Note:

The JSON validation constants are bit flags and they can be used in combination.

JSON Encoding Constants

Constant	Description
JZN_INPUT_UTF8	Input is UTF-8 encoding
JZN_INPUT_DETECT	Encoding of the input needs to be detected
JZN_INPUT_CONVERT	Input requires character set conversion

JSON Printing Constants

Constant	Description
JZNU_PRINT_PRETTY	Render textual JSON pretty printed

Constant	Description
JZNU_PRINT_ASCII	Render textual JSON with native ASCII, with escape
JZNU_PRINT_NUMFORMAT	Render textual JSON with numbers printed canonically

A

Mapping of APIs used before Oracle Database 10*g* Release 1

Here are the mappings between XML C APIs that were available in Oracle Database Oracle9*i* Release, and the current Unified XML C APIs that became available in subsequent Oracle Database.

> **See Also:**
>
> Format Models in *Oracle XML Developer's Kit Programmer's Guide*

C Package Changes

Pre-existing C APIs were available through the `oraxml` package. It had the following characteristics:

- Specification is limited to a one-to-one mapping between the xml context (`xmlctx`) and an xml document. Only one document can be accessed by DOM at any one time, however the data of multiple documents can be concurrent.
- The APIs are not always consistent, and don't always follow the declarations of the `xmlctx`.

In contrast, the new unified C APIs solve these problems:

- Multiple independent documents share the `xmlctx`.
- All APIs conform to the declarations of the `xmlctx`.
- Each document can be accessed simultaneously by DOM until explicitly destroyed by an `XmlDestroy()` call.

Initializing and Parsing Sequence Changes

The initialization and parsing of documents has changed in the Unified C API.

Example A-1 Initializing and Parsing Sequence for the Pre-Unified C API, One Document at a Time

The following pseudo-code demonstrates how to initialize and parse documents, one at a time, using the old C APIs. Contrast this with Example A-2.

```
#include <oraxml.h>
uword err;
xmlctx *ctx = xmlinit(&err, options);
for (;;)
{
   err = xmlparse(ctx, URI, options);
```

```
   ...
   /* DOM operations */
   ...
   /* recycle memory from document */
   xmlclean(ctx);
}
xmlterm(ctx);
```

Example A-2 Initializing and Parsing Sequence for the Unified C API, One Document at a Time

The following pseudo-code demonstrates how to initialize and parse documents, one at a time, using the new C APIs. Contrast this with Example A-1.

```
#include <xml.h>
xmlerr err;
xmldocnode *doc;
xmlctx *xctx = XmlCreate(&err, options, NULL);
for (;;)
{
   doc = XmlLoadDom(xctx, &err, "URI", URI, NULL);
   ...
   /* DOM operations */
   ...
   XmlFreeDocument(xctx, doc);
}
XmlDestroy(xctx);
```

Example A-3 Initializing and Parsing Sequence for the Pre-Unified C API, Multiple Documents and Simultaneous DOM Access

The following pseudo-code demonstrates how to initialize and parse multiple documents with simultaneous DOM access using the old C APIs. Contrast this with Example A-4.

```
xmlctx *ctx1 = xmlinitenc(&err, options);
xmlctx *ctx2 = xmlinitenc(&err, options);
err = xmlparse(ctx1, URI_1, options);
err = xmlparse(ctx2, URI_2, options);
...
/* DOM operations for both documents */
...
xmlterm(ctx1);
xmlterm(ctx2);
```

Example A-4 Initializing and Parsing Sequence for the Unified C API, Multiple Documents and Simultaneous DOM Access

The following pseudo-code example demonstrates how to initialize and parse multiple documents with simultaneous DOM access using the new C APIs. Contrast this with Example A-3.

```
xmldocnode *doc1;
xmldocnode *doc2;
xmlctx *xctx = XmlCreate(&err, options, NULL);
doc1 = XmlLoadDom(xctx, &err, "URI", URI_1, NULL);
doc2 = XmlLoadDom(xctx, &err, "URI", URI_2, NULL);
...
/* DOM operations for both documents*/
...
XmlFreeDocument(xctx, doc1);
```

```
XmlFreeDocument(xctx, doc2);
...
XmlDestroy(xctx);
```

Datatype Mapping between oraxml and xml Packages

Table A-1 outlines the changes made to datatypes for the new C API.

Table A-1 Datatypes Supported by oraxml Package versus xml Package

oraxml Supported Datatype	xml Supported Datatype
uword	xmlerr
xmlacctype	xmlurlacc
xmlattrnode	xmlattrnode
xmlcdatanode	xmlcdatanode
xmlcommentnode	xmlcommentnode
xmlctx	xmlctx
xmldocnode	xmldocnode
xmldomimp	Obsolete.Use xmlctx.
xmldtdnode	xmldtdnode
xmlelemnode	xmlelemnode
xmlentnode	xmlentnode
xmlentrefnode	xmlentrefnode
xmlflags	ub4
xmlfragnode	xmlfragnode
xmlihdl	xmlurlhdl
xmlmemcb	Use individual function pointers.
xmlnode	xmlnode

Table A-1 (Cont.) Datatypes Supported by oraxml Package versus xml Package

oraxml Supported Datatype	xml Supported Datatype
xmlnodes	xmlnodelist, xmlnamedmap
xmlnotenode	xmlnotenode
xmlntype	xmlnodetype
xmlpflags	ub4
xmlpinode	xmlpinode
xmlsaxcb	xmlsaxcb
xmlstream	xmlistream, xmliostream
xmltextnode	xmltextnode
xpctx	xpctx
xpexpr	xpexpr
xpnset	Obsolete.Use`XmlXPathGetObjectNSetNum()` and `XmlXPathGetObjectNSetNode()`.
xpnsetele	Obsolete.Use`XmlXPathGetObjectNSetNum()` and `XmlXPathGetObjectNSetNode()`.
xpobj	xpobj
xpobjtyp	xmlxslobjtype
xslctx	xslctx
xsloutputmethod	xmlxsloutputmethod

Method Mapping between oraxml and xml Packages

Table A-2 outlines the changes made to the methods of the new C API.

Table A-2 Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
appendChild()	XmlDomAppendChild()

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
appendData()	XmlDomAppendData()
cloneNode()	XmlDomCloneNode()
createAttribute()	XmlDomCreateAttr()
createAttributeNS()	XmlDomCreateAttrNS()
createCDATASection()	XmlDomCreateCDATA()
createComment()	XmlDomCreateComment()
createDocument()	XmlCreateDocument()
createDocumentFragment()	XmlDomCreateFragment()
createDocumentNS()	XmlCreateDocument()
createDocumentType()	XmlCreateDTD()
createElement()	XmlDomCreateElem()
createElementNS()	XmlDomCreateElemNS()
createEntityReference()	XmlDomCreateEntityRef()
createProcessingInstruction()	XmlDomCreatePI()
createTextNode()	XmlDomCreateText()
deleteData()	XmlDomDeleteData()
freeElements()	XmlDomFreeNodeList()
getAttribute()	XmlDomGetAttr()
getAttributeIndex()	XmlDomGetAttrs(), XmlDomGetNodeMapItem()
getAttributeNode()	XmlDomGetAttrNode()

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
`getAttributes()`	`XmlDomGetAttrs()`
`getAttrLocal()`	`XmlDomGetAttrLocal()`, `XmlDomGetAttrLocalLen()`
`getAttrName()`	`XmlDomGetAttrName()`
`getAttrNamespace()`	`XmlDomGetAttrURI()`, `XmlDomGetAttrURILen()`
`getAttrPrefix()`	`XmlDomGetAttrPrefix()`
`getAttrQualifiedName()`	`XmlDomGetAttrName()`
`getAttrSpecified()`	`XmlDomGetAttrSpecified()`
`getAttrValue()`	`XmlDomGetAttrValue()`
`getCharData()`	`XmlDomGetCharData()`
`getChildNode()`	`XmlDomGetChildNode()`
`getChildNodes()`	`XmlDomGetChildNodes()`
`getContentModel()`	`XmlDomGetContentModel()`
`getDocType()`	`XmlDomGetDTD()`
`getDocTypeEntities()`	`XmlDomGetDTDEntities()`
`getDocTypeName()`	`XmlDomGetDTDName()`
`getDocTypeNotations()`	`XmlDomGetDTDNotations()`
`getDocument()`	Obsolete; document returned by `XmlLoadDom`***`xxx`***`()` calls
`getDocumentElement()`	`XmlDomGetDoctElem()`
`getElementByID()`	`XmlDomGetElemByID()`
`getElementsByTagName()`	`XmlDomGetElemsByTag()`
`getElementsByTagNameNS()`	`XmlDomGetElemsByTag()`

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
`getEncoding()`	`XmlDomGetEncoding()`
`getEntityNotation()`	`XmlDomGetEntityNotation()`
`getEntityPubID()`	`XmlDomGetEntityPubID()`
`getEntitySysID()`	`XmlDomGetEntitySysID()`
`getFirstChild()`	`XmlDomGetFirstChild()`
`getImplementation()`	Obsolete; use `xmlctx` instead of `DOMImplementation`
`getLastChild()`	`XmlDomGetLastChild()`
`getNamedItem()`	`XmlDomGetNamedItem()`
`getNextSibling()`	`XmlDomGetNextSibling()`
`getNodeLocal()`	`XmlDomGetNodeLocal()`, `XmlDomGetNodeLocalLen()`
`getNodeMapLength()`	`XmlDomGetNodeMapLength()`
`getNodeName()`	`XmlDomGetNodeName()`, `XmlDomGetNodeNameLen()`
`getNodeNameSpace()`	`XmlDomGetNodeURI()`, `XmlDomGetNodeURILen()`
`getNodePrefix()`	`XmlDomGetNodePrefix()`
`getNodeQualifiedName()`	`XmlDomGetNodedName()`, `XmlDomGetNodedNameLen()`
`getNodeType()`	`XmlDomGetNodeType()`
`getNodeValue()`	`XmlDomGetNodeValue()`, `XmlDomGetNodeValueLen()`
`getNotationPubID()`	`XmlDomGetNotationPubID()`
`getNotationSysID()`	`XmlDomGetNotationSysID()`
`getOwnerDocument()`	`XmlDomGetOwnerDocument()`
`getParentNode()`	`XmlDomGetParentNode()`

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
getPIData()	XmlDomGetPIData()
getPITarget()	XmlDomGetPITarget()
getPreviousSibling()	XmlDomGetPrevSibling()
getTagName()	XmlDomGetTagName()
hasAttributes()	XmlDomHasAttrs()
hasChildNodes()	XmlDomHasChildNodes()
hasFeature()	XmlHasFeature()
importNode()	XmlDomImportNode()
insertBefore()	XmlDomInsertBefore()
insertData()	XmlDomInsertData()
isSingleChar()	XmlIsSimple()
isStandalone()	XmlDomGetDecl()
isUnicode()	XmlDomIsUnicode()
nodeValid()	XmlDomValidate()
normalize()	XmlDomNormalize()
numAttributes()	XmlDomNumAttrs()
numChildNodes()	XmlDomNumChildNodes()
prefixToURI()	XmlDomPrefixToURI()
printBuffer()	XmlSaveDomBuffer()
printBufferEnc()	XmlSaveDomBuffer()
printCallback()	XmlSaveDomStream()

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
printCallbackEnc()	XmlSaveDomStream()
printSize()	XmlSaveDomSize()
printSizeEnc()	XmlSaveDomSize()
printStream()	XmlSaveDomStdio()
printStreamEnc()	XmlSaveDomStdio()
removeAttribute()	XmlDomRemoveAttr()
removeAttributeNode()	XmlDomRemoveAttrNode()
removeChild()	XmlDomRemoveChild()
removeNamedItem()	XmlDomRemoveNamedItem()
replaceChild()	XmlDomReplaceChild()
replaceData()	XmlDomReplaceData()
saveString2()	XmlDomSaveString2()
saveString()	XmlDomSaveString()
setAttribute()	XmlDomSetAttr()
setAttributeNode()	XmlDomSetAttrNode()
setAttrValue()	XmlDomSetAttrValue()
setCharData()	XmlDomSetCharData()
setNamedItem()	XmlDomSetNamedItem()
setNodeValue()	XmlDomSetNodeValue(), XmlDomSetNodeValueLen()
setPIData()	XmlDomSetPIData()
splitText()	XmlDomSplitText()

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
substringData()	XmlDomSubstringData()
xmlaccess()	XmlAccess()
xmlinit()	XmlCreate()
xmlinitenc()	XmlCreate()
xmlparse()	XmlLoadDomURI()
xmlparsebuf()	XmlLoadDomBuffer()
xmlparsedtd()	Obsolete; use XML_LOAD_FLAG_DTD_ONLY flag in XmlLoad*XXX*() calls.
xmlparsefile()	XmlLoadDomFile()
xmlparsestream()	XmlLoadDomStream()
xmlterm()	XmlDestroy()
xpevalxpathexpr()	XmlXPathEval()
xpfreexpathctx()	XmlXPathDeleteCtx()
xpgetbooleanval()	XmlXPathGetObjectBoolean()
xpgetfirstnsetelem()	XmlXPathGetObjectNSetNum()
xpgetnextnsetelem()	XmlXPathGetObjectNSetNum()
xpgetnsetelemnode()	XmlXPathGetObjectNSetNum()
xpgetnsetval()	XmlXPathGetObjectNSetNum()
xpgetnumval()	XmlXPathGetObjectNumber()
xpgetrtfragval()	XmlXPathGetObjectFragment()
xpgetstrval()	XmlXPathGetObjectString()
xpgetxpobjtyp()	XmlXPathGetObjectType()

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
xpmakexpathctx()	XmlXPathCreateCtx()
xpparsexpathexpr()	XmlXPathParse()
xslgetbaseuri()	XmlXslGetBaseURI()
xslgetoutputdomctx()	XmlXslGetOutputDom()
xslgetoutputsax()	Unnecessary
xslgetoutputstream()	Unnecessary
xslgetresultdocfrag()	XmlXslGetOutputFragment()
xslgettextparam()	XmlXslGetTextParam()
xslgetxslctx()	Unnecessary
xslinit()	XmlXslCreateCtx()
xslprocess()	XmlXslProcess()
xslprocessex()	XmlXslProcess()
xslprocessxml()	XmlXslProcess()
xslprocessxmldocfrag()	XmlXslProcess()
xslresetallparams()	XmlXslResetAllParams()
xslsetoutputdomctx()	XmlXslSetOutputDom()
xslsetoutputencoding()	XmlXslSetOutputEncoding()
xslsetoutputmethod()	XmlXslSetOutputMethod()
xslsetoutputsax()	XmlXslSetOutputSax()
xslsetoutputsaxctx()	XmlXslSetOutputSax()
xslsetoutputstream()	XmlXslSetOutputStream()

Table A-2 (Cont.) Methods of the oraxml Package versus the xml Package

Package oraxml Method	Package xml Method(s)
`xslsettextparam()`	`XmlXslSetTextParam()`
`xslterm()`	`XmlXslDeleteCtx()`

Index

C

C package methods
- JsonDomClearError in package DOM, *17-16*
- JsonDomCreateArray in package DOM, *17-8*
- JsonDomCreateBinary in package DOM, *17-9*
- JsonDomCreateBoolean in package DOM, *17-10*
- JsonDomCreateDouble in package DOM, *17-10*
- JsonDomCreateFloat in package DOM, *17-9*
- JsonDomCreateNull in package DOM, *17-12*
- JsonDomCreateObject in package DOM, *17-7*
- JsonDomCreateOCIDate in package DOM, *17-11*
- JsonDomCreateOCIDateTime in package DOM, *17-12*, *17-14*
- JsonDomCreateOCIInterval in package DOM, *17-7*
- JsonDomCreateOCINumber in package DOM, *17-11*
- JsonDomCreateString in package DOM, *17-8*
- JsonDomFreeNode in package DOM, *17-14*
- JsonDomGetArraySize in package DOM, *17-4*
- JsonDomGetElements in package DOM, *17-4*, *17-6*
- JsonDomGetError in package DOM, *17-15*
- JsonDomGetErrorMessage in package DOM, *17-15*
- JsonDomGetNodeType in package DOM, *17-5*
- JsonDomGetNumFields in package DOM, *17-3*
- JsonDomGetRoot in package DOM, *17-2*
- JsonDomSetField in package DOM, *17-13*
- JsonDomSetRoot in package DOM, *17-2*
- XML_ACCESS_CLOSE_F in package Callback, *2-1*
- XML_ACCESS_OPEN_F in package Callback, *2-1*
- XML_ACCESS_READ_F in package Callback, *2-1*

C package methods *(continued)*
- XML_ALLOC_F in package Callback, *2-1*
- XML_ERRMSG_F in package Callback, *2-2*
- XML_FREE_F in package Callback, *2-2*
- XML_STREAM_CLOSE_F in package Callback, *2-2*
- XML_STREAM_OPEN_F in package Callback, *2-2*
- XML_STREAM_READ_F in package Callback, *2-2*
- XML_STREAM_WRITE_F in package Callback, *2-3*
- XmlAccess in package XML, *11-1*
- XmlCreate in package XML, *11-2*
- XmlCreateDocument in package XML, *11-6*
- XmlCreateDTD in package XML, *11-5*
- XmlDestroy in package XML, *11-6*
- XmlDiff in package XML, *11-7*
- XmlDiff in package XmlDiff, *12-1*
- XMLDOM_ACCEPT_NODE_F in package Traversal, *10-3*
- XmlDomAppendChild in package DOM, *3-65*
- XmlDomAppendData in package DOM, *3-12*
- XmlDomCleanNode in package DOM, *3-66*
- XmlDomCloneNode in package DOM, *3-66*
- XmlDomCreateAttr in package DOM, *3-18*
- XmlDomCreateAttrNS in package DOM, *3-19*
- XmlDomCreateCDATA in package DOM, *3-20*
- XmlDomCreateComment in package DOM, *3-21*
- XmlDomCreateElem in package DOM, *3-22*
- XmlDomCreateElemNS in package DOM, *3-22*
- XmlDomCreateEntityRef in package DOM, *3-23*
- XmlDomCreateFragment in package DOM, *3-24*
- XmlDomCreateNodeIter in package Traversal, *10-1*
- XmlDomCreatePI in package DOM, *3-24*
- XmlDomCreateRange in package Range, *6-1*
- XmlDomCreateText in package DOM, *3-25*
- XmlDomCreateTreeWalker in package Traversal, *10-2*

C package methods *(continued)*
XmlDomDeleteData in package DOM, *3-12*
XmlDomFreeNode in package DOM, *3-67*
XmlDomFreeNodeList in package DOM, *3-97*
XmlDomFreeString in package DOM, *3-26*
XmlDomGetAttr in package DOM, *3-43*
XmlDomGetAttrLocal in package DOM, *3-2*
XmlDomGetAttrLocalLen in package DOM, *3-2*
XmlDomGetAttrName in package DOM, *3-3*
XmlDomGetAttrNameLen in package DOM, *3-4*
XmlDomGetAttrNode in package DOM, *3-44*
XmlDomGetAttrNodeNS in package DOM, *3-45*
XmlDomGetAttrNS in package DOM, *3-43*
XmlDomGetAttrPrefix in package DOM, *3-5*
XmlDomGetAttrs in package DOM, *3-68*
XmlDomGetAttrSpecified in package DOM, *3-5*
XmlDomGetAttrURI in package DOM, *3-6*
XmlDomGetAttrURILen in package DOM, *3-7*
XmlDomGetAttrValue in package DOM, *3-8*
XmlDomGetAttrValueLen in package DOM, *3-8*
XmlDomGetAttrValueStream in package DOM, *3-9*
XmlDomGetBaseURI in package DOM, *3-27*
XmlDomGetCharData in package DOM, *3-13*
XmlDomGetCharDataLength in package DOM, *3-14*
XmlDomGetChildNodes in package DOM, *3-68*
XmlDomGetChildrenByTag in package DOM, *3-45*
XmlDomGetChildrenByTagNS in package DOM, *3-46*
XmlDomGetDecl in package DOM, *3-28*
XmlDomGetDefaultNS in package DOM, *3-69*
XmlDomGetDocElem in package DOM, *3-29*
XmlDomGetDocElemByID in package DOM, *3-29*
XmlDomGetDocElemsByTag in package DOM, *3-30*
XmlDomGetDocElemsByTagNS in package DOM, *3-31*
XmlDomGetDTD in package DOM, *3-27*
XmlDomGetDTDEntities in package DOM, *3-38*
XmlDomGetDTDInternalSubset in package DOM, *3-39*
XmlDomGetDTDName in package DOM, *3-40*

C package methods *(continued)*
XmlDomGetDTDNotations in package DOM, *3-40*
XmlDomGetDTDPubID in package DOM, *3-41*
XmlDomGetDTDSysID in package DOM, *3-41*
XmlDomGetElemsByTag in package DOM, *3-47*
XmlDomGetElemsByTagNS in package DOM, *3-47*
XmlDomGetEntityNotation in package DOM, *3-55*
XmlDomGetEntityPubID in package DOM, *3-55*
XmlDomGetEntitySysID in package DOM, *3-56*
XmlDomGetEntityType in package DOM, *3-57*
XmlDomGetFirstChild in package DOM, *3-69*
XmlDomGetFirstPfnsPair in package DOM, *3-70*
XmlDomGetLastChild in package DOM, *3-70*
XmlDomGetLastError in package DOM, *3-32*
XmlDomGetNamedItem in package DOM, *3-58*
XmlDomGetNamedItemNS in package DOM, *3-58*
XmlDomGetNextPfnsPair in package DOM, *3-71*
XmlDomGetNextSibling in package DOM, *3-72*
XmlDomGetNodeListItem in package DOM, *3-97*
XmlDomGetNodeListLength in package DOM, *3-98*
XmlDomGetNodeLocal in package DOM, *3-72*
XmlDomGetNodeLocalLen in package DOM, *3-73*
XmlDomGetNodeMapItem in package DOM, *3-59*
XmlDomGetNodeMapLength in package DOM, *3-60*
XmlDomGetNodeName in package DOM, *3-74*
XmlDomGetNodeNameLen in package DOM, *3-74*
XmlDomGetNodePrefix in package DOM, *3-75*
XmlDomGetNodeType in package DOM, *3-76*
XmlDomGetNodeURI in package DOM, *3-77*
XmlDomGetNodeURILen in package DOM, *3-78*

C package methods *(continued)*
XmlDomGetNodeValue in package DOM, *3-79*
XmlDomGetNodeValueLen in package DOM, *3-79*
XmlDomGetNodeValueStream in package DOM, *3-80*
XmlDomGetNotationPubID in package DOM, *3-99*
XmlDomGetNotationSysID in package DOM, *3-99*
XmlDomGetOwnerDocument in package DOM, *3-81*
XmlDomGetOwnerElem in package DOM, *3-10*
XmlDomGetParentNode in package DOM, *3-81*
XmlDomGetPIData in package DOM, *3-100*
XmlDomGetPITarget in package DOM, *3-101*
XmlDomGetPrevSibling in package DOM, *3-82*
XmlDomGetPullNodeAsBinaryStream in package DOM, *3-82*
XmlDomGetPullNodeAsCharacterStream in package DOM, *3-83*
XmlDomGetPushNodeAsBinaryStream in package DOM, *3-83*
XmlDomGetPushNodeAsCharacterStream in package DOM, *3-84*
XmlDomGetSchema in package DOM, *3-32*
XmlDomGetSourceEntity in package DOM, *3-84*
XmlDomGetSourceLine in package DOM, *3-85*
XmlDomGetSourceLocation in package DOM, *3-85*
XmlDomGetTag in package DOM, *3-48*
XmlDomHasAttr in package DOM, *3-49*
XmlDomHasAttrNS in package DOM, *3-49*
XmlDomHasAttrs in package DOM, *3-85*
XmlDomHasChildNodes in package DOM, *3-86*
XmlDomImportNode in package DOM, *3-33*
XmlDomInsertBefore in package DOM, *3-86*
XmlDomInsertData in package DOM, *3-14*
XmlDomIsSchemaBased in package DOM, *3-34*
XmlDomIterDetach in package Traversal, *10-4*
XmlDomIterNextNode in package Traversal, *10-5*
XmlDomIterPrevNode in package Traversal, *10-5*
XmlDomNormalize in package DOM, *3-87*
XmlDomNumAttrs in package DOM, *3-87*

C package methods *(continued)*
XmlDomNumChildNodes in package DOM, *3-88*
XmlDomPrefixToURI in package DOM, *3-88*
XmlDomRangeClone in package Range, *6-2*
XmlDomRangeCloneContents in package Range, *6-2*
XmlDomRangeCollapse in package Range, *6-3*
XmlDomRangeCompareBoundaryPoints in package Range, *6-3*
XmlDomRangeDeleteContents in package Range, *6-4*
XmlDomRangeDetach in package Range, *6-4*
XmlDomRangeExtractContents in package Range, *6-5*
XmlDomRangeGetCollapsed in package Range, *6-5*
XmlDomRangeGetCommonAncestor in package Range, *6-6*
XmlDomRangeGetDetached in package Range, *6-6*
XmlDomRangeGetEndContainer in package Range, *6-7*
XmlDomRangeGetEndOffset in package Range, *6-7*
XmlDomRangeGetStartContainer in package Range, *6-8*
XmlDomRangeGetStartOffset in package Range, *6-8*
XmlDomRangeIsConsistent in package Range, *6-9*
XmlDomRangeSelectNode in package Range, *6-9*
XmlDomRangeSelectNodeContents in package Range, *6-10*
XmlDomRangeSetEnd in package Range, *6-10*
XmlDomRangeSetEndBefore in package Range, *6-11*
XmlDomRangeSetStart in package Range, *6-11*
XmlDomRangeSetStartAfter in package Range, *6-12*
XmlDomRangeSetStartBefore in package Range, *6-13*
XmlDomRemoveAttr in package DOM, *3-50*
XmlDomRemoveAttrNode in package DOM, *3-51*
XmlDomRemoveAttrNS in package DOM, *3-51*
XmlDomRemoveChild in package DOM, *3-89*
XmlDomRemoveNamedItem in package DOM, *3-60*

C package methods *(continued)*
XmlDomRemoveNamedItemNS in package DOM, *3-61*
XmlDomRenameNode in package DOM, *3-89*
XmlDomRenameNodeNS in package DOM, *3-90*
XmlDomReplaceChild in package DOM, *3-91*
XmlDomReplaceData in package DOM, *3-15*
XmlDomSaveString in package DOM, *3-34*
XmlDomSaveString2 in package DOM, *3-35*
XmlDomSetAttr in package DOM, *3-52*
XmlDomSetAttrNode in package DOM, *3-53*
XmlDomSetAttrNodeNS in package DOM, *3-54*
XmlDomSetAttrNS in package DOM, *3-53*
XmlDomSetAttrValue in package DOM, *3-10*
XmlDomSetAttrValueStream in package DOM, *3-11*
XmlDomSetBaseURI in package DOM, *3-35*
XmlDomSetCharData in package DOM, *3-16*
XmlDomSetDefaultNS in package DOM, *3-91*
XmlDomSetDocOrder in package DOM, *3-37*
XmlDomSetDTD in package DOM, *3-36*
XmlDomSetLastError in package DOM, *3-37*
XmlDomSetNamedItem in package DOM, *3-62*
XmlDomSetNamedItemNS in package DOM, *3-62*
XmlDomSetNodePrefix in package DOM, *3-92*
XmlDomSetNodeValue in package DOM, *3-92*
XmlDomSetNodeValueLen in package DOM, *3-93*
XmlDomSetNodeValueStream in package DOM, *3-94*
XmlDomSetPIData in package DOM, *3-101*
XmlDomSetPullNodeAsBinaryStream in package DOM, *3-94*
XmlDomSetPullNodeAsCharacterStream in package DOM, *3-95*
XmlDomSetPushNodeAsBinaryStream in package DOM, *3-95*
XmlDomSetPushNodeAsCharacterStream in package DOM, *3-96*
XmlDomSplitText in package DOM, *3-102*
XmlDomSubstringData in package DOM, *3-16*
XmlDomSync in package DOM, *3-38*
XmlDomValidate in package DOM, *3-96*
XmlDomWalkerFirstChild in package Traversal, *10-7*

C package methods *(continued)*
XmlDomWalkerGetCurrentNode in package Traversal, *10-7*
XmlDomWalkerGetRoot in package Traversal, *10-8*
XmlDomWalkerLastChild in package Traversal, *10-8*
XmlDomWalkerNextNode in package Traversal, *10-9*
XmlDomWalkerNextSibling in package Traversal, *10-9*
XmlDomWalkerParentNode in package Traversal, *10-10*
XmlDomWalkerPrevNode in package Traversal, *10-11*
XmlDomWalkerPrevSibling in package Traversal, *10-11*
XmlDomWalkerSetCurrentNode in package Traversal, *10-12*
XmlDomWalkerSetRoot in package Traversal, *10-12*
XmlEvCleanPPCtx in package Event, *4-5*
XmlEvCreatePPCtx in package Event, *4-5*
XmlEvCreateSVCtx in package Event, *4-6*
XmlEvDestroyPPCtx in package Event, *4-7*
XmlEvDestroySVCtx in package Event, *4-7*
XmlEvGetAttrCount in package Event, *4-8*
XmlEvGetAttrDeclBody in package Event, *4-8*
XmlEvGetAttrDeclBody0 in package Event, *4-9*
XmlEvGetAttrDeclCount in package Event, *4-9*
XmlEvGetAttrDeclElName in package Event, *4-9*
XmlEvGetAttrDeclElName0 in package Event, *4-10*
XmlEvGetAttrDeclLocalName in package Event, *4-10*
XmlEvGetAttrDeclLocalName0 in package Event, *4-11*
XmlEvGetAttrDeclName in package Event, *4-11*
XmlEvGetAttrDeclName0 in package Event, *4-12*
XmlEvGetAttrDeclPrefix in package Event, *4-12*
XmlEvGetAttrDeclPrefix0 in package Event, *4-13*
XmlEvGetAttrID in package Event, *4-13*
XmlEvGetAttrLocalName in package Event, *4-14*
XmlEvGetAttrLocalName0 in package Event, *4-14*
XmlEvGetAttrName in package Event, *4-14*

C package methods *(continued)*
XmlEvGetAttrName0 in package Event, *4-15*
XmlEvGetAttrPrefix in package Event, *4-15*
XmlEvGetAttrPrefix0 in package Event, *4-16*
XmlEvGetAttrURI in package Event, *4-16*
XmlEvGetAttrURI0 in package Event, *4-17*
XmlEvGetAttrUriID in package Event, *4-17*
XmlEvGetAttrValue in package Event, *4-18*
XmlEvGetAttrValue0 in package Event, *4-18*
XmlEvGetElDeclContent in package Event, *4-18*
XmlEvGetElDeclContent0 in package Event, *4-19*
XmlEvGetEncoding in package Event, *4-19*
XmlEvGetError in package Event, *4-20*
XmlEvGetLocalName in package Event, *4-21*
XmlEvGetLocalName0 in package Event, *4-22*
XmlEvGetLocation in package Event, *4-22*
XmlEvGetName in package Event, *4-20*
XmlEvGetName0 in package Event, *4-21*
XmlEvGetPEisGen in package Event, *4-24*
XmlEvGetPERepl in package Event, *4-25*
XmlEvGetPERepl0 in package Event, *4-25*
XmlEvGetPIData in package Event, *4-23*
XmlEvGetPIData0 in package Event, *4-23*
XmlEvGetPITarget in package Event, *4-23*
XmlEvGetPITarget0 in package Event, *4-24*
XmlEvGetPrefix in package Event, *4-25*
XmlEvGetPrefix0 in package Event, *4-26*
XmlEvGetPubId in package Event, *4-26*
XmlEvGetPubId0 in package Event, *4-27*
XmlEvGetSysId in package Event, *4-27*
XmlEvGetSysId0 in package Event, *4-28*
XmlEvGetTagID in package Event, *4-28*
XmlEvGetTagUriID in package Event, *4-29*
XmlEvGetText in package Event, *4-29*
XmlEvGetText0 in package Event, *4-30*
XmlEvGetUENdata in package Event, *4-30*
XmlEvGetUENdata0 in package Event, *4-31*
XmlEvGetURI in package Event, *4-31*
XmlEvGetURI0 in package Event, *4-31*
XmlEvGetVersion in package Event, *4-32*
XmlEvIsEncodingSpecified in package Event, *4-32*
XmlEvIsStandalone in package Event, *4-33*
XmlEvLoadPPDoc in package Event, *4-34*
XmlEvNamespaceAttr in package Event, *4-33*
XmlEvNext in package Event, *4-33*
XmlEvNextTag in package Event, *4-34*
XmlEvSchemaValidate in package Event, *4-35*
XmlFreeDocument in package XML, *11-8*
XmlGetEncoding in package XML, *11-8*

C package methods *(continued)*
XmlHasFeature in package XML, *11-9*
XmlHash in package XmlDiff, *12-3*
XmlIsSimple in package XML, *11-10*
XmlIsUnicode in package XML, *11-10*
XmlLoadDom in package XML, *11-11*
XmlLoadSax in package XML, *11-12*
XmlLoadSaxVA in package XML, *11-13*
XmlPatch in package XmlDiff, *12-4*
XmlSaveDom in package XML, *11-13*
XmlSaxAttributeDecl in package SAX, *7-1*
XmlSaxBeginGen in package SAX, *7-2*
XmlSaxCDATA in package SAX, *7-2*
XmlSaxCharacters in package SAX, *7-3*
XmlSaxComment in package SAX, *7-4*
XmlSaxElementDecl in package SAX, *7-4*
XmlSaxEndDocument in package SAX, *7-5*
XmlSaxEndElement in package SAX, *7-5*
XmlSaxEndGen in package SAX, *7-5*
XmlSaxNotationDecl in package SAX, *7-6*
XmlSaxParsedEntityDecl in package SAX, *7-6*
XmlSaxPI in package SAX, *7-6*
XmlSaxStartDocument in package SAX, *7-7*
XmlSaxStartElement in package SAX, *7-8*
XmlSaxStartElementNS in package SAX, *7-8*
XmlSaxUnparsedEntityDecl in package SAX, *7-9*
XmlSaxWhitespace in package SAX, *7-10*
XmlSaxXmlDecl in package SAX, *7-11*
XmlSchemaClean in package Schema, *8-8*
XmlSchemaCreate in package Schema, *8-1*
XmlSchemaDestroy in package Schema, *8-2*
XmlSchemaErrorWhere in package Schema, *8-2*
XmlSchemaLoad in package Schema, *8-3*
XmlSchemaLoadedList in package Schema, *8-4*
XmlSchemaSetErrorHandler in package Schema, *8-4*
XmlSchemaSetValidateOptions in package Schema, *8-5*
XmlSchemaTargetNamespace in package Schema, *8-6*
XmlSchemaUnload in package Schema, *8-6*
XmlSchemaValidate in package Schema, *8-7*
XmlSchemaVersion in package Schema, *8-8*
XmlSoapAddBodyElement in package SOAP, *9-2*
XmlSoapAddFaultReason in package SOAP, *9-3*
XmlSoapAddFaultSubDetail in package SOAP, *9-3*
XmlSoapAddHeaderElement in package SOAP, *9-4*

C package methods *(continued)*
XmlSoapCall in package SOAP, *9-5*
XmlSoapCreateConnection in package SOAP, *9-5*
XmlSoapCreateCtx in package SOAP, *9-7*
XmlSoapCreateMsg in package SOAP, *9-8*
XmlSoapDestroyConnection in package SOAP, *9-8*
XmlSoapDestroyCtx in package SOAP, *9-9*
XmlSoapDestroyMsg in package SOAP, *9-9*
XmlSoapError in package SOAP, *9-10*
XmlSoapGetBody in package SOAP, *9-10*
XmlSoapGetBodyElement in package SOAP, *9-11*
XmlSoapGetEnvelope in package SOAP, *9-12*
XmlSoapGetFault in package SOAP, *9-12*
XmlSoapGetHeader in package SOAP, *9-13*
XmlSoapGetHeaderElement in package SOAP, *9-14*
XmlSoapGetMustUnderstand in package SOAP, *9-14*
XmlSoapGetReasonLang in package SOAP, *9-15*
XmlSoapGetReasonNum in package SOAP, *9-16*
XmlSoapGetRelay in package SOAP, *9-16*
XmlSoapGetRole in package SOAP, *9-17*
XmlSoapHasFault in package SOAP, *9-17*
XmlSoapSetFault in package SOAP, *9-18*
XmlSoapSetMustUnderstand in package SOAP, *9-19*
XmlSoapSetRelay in package SOAP, *9-20*
XmlSoapSetRole in package SOAP, *9-20*
XmlVersion in package XML, *11-14*
XmlXPathCreateCtx in package XPath, *13-1*
XmlXPathDestroyCtx in package XPath, *13-2*
XmlXPathEval in package XPath, *13-2*
XmlXPathGetObjectBoolean in package XPath, *13-3*
XmlXPathGetObjectFragment in package XPath, *13-3*
XmlXPathGetObjectNSetNode in package XPath, *13-4*
XmlXPathGetObjectNSetNum in package XPath, *13-4*
XmlXPathGetObjectNumber in package XPath, *13-5*
XmlXPathGetObjectString in package XPath, *13-5*
XmlXPathGetObjectType in package XPath, *13-6*
XmlXPathParse in package XPath, *13-6*
XmlXPointerEval in package XPointer, *14-1*
C package methods *(continued)*
XmlXPtrLocGetNode in package XPointer, *14-2*
XmlXPtrLocGetPoint in package XPointer, *14-2*
XmlXPtrLocGetRange in package XPointer, *14-2*
XmlXPtrLocGetType in package XPointer, *14-3*
XmlXPtrLocSetFree in package XPointer, *14-4*
XmlXPtrLocSetGetItem in package XPointer, *14-4*
XmlXPtrLocSetGetLength in package XPointer, *14-5*
XmlXPtrLocToString in package XPointer, *14-3*
XmlXslCreate in package XSLT, *15-1*
XmlXslDestroy in package XSLT, *15-2*
XmlXslGetBaseURI in package XSLT, *15-2*
XmlXslGetOutput in package XSLT, *15-3*
XmlXslGetStylesheetDom in package XSLT, *15-3*
XmlXslGetTextParam in package XSLT, *15-3*
XmlXslProcess in package XSLT, *15-4*
XmlXslResetAllParams in package XSLT, *15-4*
XmlXslSetOutputDom in package XSLT, *15-5*
XmlXslSetOutputEncoding in package XSLT, *15-5*
XmlXslSetOutputMethod in package XSLT, *15-5*
XmlXslSetOutputSax in package XSLT, *15-6*
XmlXslSetOutputStream in package XSLT, *15-6*
XmlXslSetTextParam in package XSLT, *15-7*
XMLXVM_DEBUG_F in package XSLTVM, *16-7*
XmlXvmCompileBuffer in package XSLTVM, *16-1*
XmlXvmCompileDom in package XSLTVM, *16-1*
XmlXvmCompileFile in package XSLTVM, *16-2*
XmlXvmCompileURI in package XSLTVM, *16-3*
XmlXvmCompileXPath in package XSLTVM, *16-4*
XmlXvmCreate in package XSLTVM, *16-7*
XmlXvmCreateComp in package XSLTVM, *16-4*
XmlXvmDestroy in package XSLTVM, *16-8*
XmlXvmDestroyComp in package XSLTVM, *16-5*

C package methods *(continued)*
XmlXvmEvaluateXPath in package XSLTVM, *16-8*
XmlXvmGetBytecodeLength in package XSLTVM, *16-5*
XmlXvmGetObjectBoolean in package XSLTVM, *16-9*
XmlXvmGetObjectNSetNode in package XSLTVM, *16-10*
XmlXvmGetObjectNSetNum in package XSLTVM, *16-10*
XmlXvmGetObjectNumber in package XSLTVM, *16-11*
XmlXvmGetObjectString in package XSLTVM, *16-11*
XmlXvmGetObjectType in package XSLTVM, *16-12*
XmlXvmGetOutputDom in package XSLTVM, *16-12*
XmlXvmResetParams in package XSLTVM, *16-13*
XmlXvmSetBaseURI in package XSLTVM, *16-13*
XmlXvmSetBytecodeBuffer in package XSLTVM, *16-13*
XmlXvmSetBytecodeFile in package XSLTVM, *16-14*
XmlXvmSetBytecodeURI in package XSLTVM, *16-14*
XmlXvmSetDebugFunc in package XSLTVM, *16-15*
XmlXvmSetOutputDom in package XSLTVM, *16-15*
XmlXvmSetOutputEncoding in package XSLTVM, *16-16*
XmlXvmSetOutputSax in package XSLTVM, *16-16*
XmlXvmSetOutputStream in package XSLTVM, *16-17*
XmlXvmSetTextParam in package XSLTVM, *16-17*
XmlXvmTransformBuffer in package XSLTVM, *16-18*
XmlXvmTransformDom in package XSLTVM, *16-18*
XmlXvmTransformFile in package XSLTVM, *16-19*
XmlXvmTransformURI in package XSLTVM, *16-20*
C packages
Callback, *2-1*
DOM, *3-1*
Event, *4-1*
Range, *6-1*
SAX, *7-1*
Schema, *8-1*
SOAP, *9-1*
Traversal, *10-1*
XML, *11-1*
XmlDiff, *12-1*
XPath, *13-1*
XPointer, *14-1*
XSLT, *15-1*
XSLTVM, *16-1*
Callback package for C, *2-1*
constants, *17-16*

D

DOM Node type, *17-16*
DOM package for C, *3-1*

E

Event package for C, *4-1*

J

JSON validation constants, *17-16*
JsonDateTime, *1-17*
JsonDayInterval, *1-17*
JsonDomArray, *1-15*
JsonDomClearError in package DOM package for C, *17-16*
JsonDomCreateArray in package DOM package for C, *17-8*
JsonDomCreateBinary in package DOM package for C, *17-9*
JsonDomCreateBoolean in package DOM package for C, *17-10*
JsonDomCreateDouble in package DOM package for C, *17-10*
JsonDomCreateFloat in package DOM package for C, *17-9*
JsonDomCreateNull in package DOM package for C, *17-12*
JsonDomCreateObject in package DOM package for C, *17-7*
JsonDomCreateOCIDate in package DOM package for C, *17-11*
JsonDomCreateOCIDateTime in package DOM package for C, *17-12*, *17-14*
JsonDomCreateOCIInterval in package DOM package for C, *17-7*
JsonDomCreateOCINumber in package DOM package for C, *17-11*
JsonDomCreateString in package DOM package for C, *17-8*
JsonDomDoc, *1-14*

JsonDomFreeNode in package DOM package for C, *17-14*
JsonDomGetArraySize in package DOM package for C, *17-4*
JsonDomGetElements in package DOM package for C, *17-4*, *17-6*
JsonDomGetError in package DOM package for C, *17-15*
JsonDomGetErrorMessage in package DOM package for C, *17-15*
JsonDomGetNodeType in package DOM package for C, *17-5*
JsonDomGetNumFields in package DOM package for C, *17-3*
JsonDomGetRoot in package DOM package for C, *17-2*
JsonDomNode, *1-15*, *1-16*
JsonDomObject, *1-15*
JsonDomScalar, *1-15*
JsonDomSetField in package DOM package for C, *17-13*
JsonDomSetRoot in package DOM package for C, *17-2*
JsonOCIVal, *1-16*
JsonYearInterval, *1-17*
jznnodetype, *1-15*
jznvaltype, *1-15*

M

methods
- JsonDomClearError in package DOM for C, *17-16*
- JsonDomCreateArray in package DOM for C, *17-8*
- JsonDomCreateBinary in package DOM for C, *17-9*
- JsonDomCreateBoolean in package DOM for C, *17-10*
- JsonDomCreateDouble in package DOM for C, *17-10*
- JsonDomCreateFloat in package DOM for C, *17-9*
- JsonDomCreateNull in package DOM for C, *17-12*
- JsonDomCreateObject in package DOM for C, *17-7*
- JsonDomCreateOCIDate in package DOM for C, *17-11*
- JsonDomCreateOCIDateTime in package DOM for C, *17-12*, *17-14*
- JsonDomCreateOCIInterval in package DOM for C, *17-7*
- JsonDomCreateOCINumber in package DOM for C, *17-11*

methods *(continued)*
- JsonDomCreateString in package DOM for C, *17-8*
- JsonDomFreeNode in package DOM for C, *17-14*
- JsonDomGetArraySize in package DOM for C, *17-4*
- JsonDomGetElements in package DOM for C, *17-4*, *17-6*
- JsonDomGetError in package DOM for C, *17-15*
- JsonDomGetErrorMessage in package DOM for C, *17-15*
- JsonDomGetNodeType in package DOM for C, *17-5*
- JsonDomGetNumFields in package DOM for C, *17-3*
- JsonDomGetRoot in package DOM for C, *17-2*
- JsonDomSetField in package DOM for C, *17-13*
- JsonDomSetRoot in package DOM for C, *17-2*
- XML_ACCESS_CLOSE_F in package Callback for C, *2-1*
- XML_ACCESS_OPEN_F in package Callback for C, *2-1*
- XML_ACCESS_READ_F in package Callback for C, *2-1*
- XML_ALLOC_F in package Callback for C, *2-1*
- XML_ERRMSG_F in package Callback for C, *2-2*
- XML_FREE_F in package Callback for C, *2-2*
- XML_STREAM_CLOSE_F in package Callback for C, *2-2*
- XML_STREAM_OPEN_F in package Callback for C, *2-2*
- XML_STREAM_READ_F in package Callback for C, *2-2*
- XML_STREAM_WRITE_F in package Callback for C, *2-3*
- XmlAccess in package XML for C, *11-1*
- XmlCreate in package XML for C, *11-2*
- XmlCreateDocument in package XML for C, *11-6*
- XmlCreateDTD in package XML for C, *11-5*
- XmlDestroy in package XML for C, *11-6*
- XmlDiff in package XML for C, *11-7*
- XmlDiff in package XmlDiff for C, *12-1*
- XMLDOM_ACCEPT_NODE_F in package Traversal for C, *10-3*
- XmlDomAppendChild in package DOM for C, *3-65*

methods *(continued)*
XmlDomAppendData in package DOM for C, *3-12*
XmlDomCleanNode in package DOM for C, *3-66*
XmlDomCloneNode in package DOM for C, *3-66*
XmlDomCreateAttr in package DOM for C, *3-18*
XmlDomCreateAttrNS in package DOM for C, *3-19*
XmlDomCreateCDATA in package DOM for C, *3-20*
XmlDomCreateComment in package DOM for C, *3-21*
XmlDomCreateElem in package DOM for C, *3-22*
XmlDomCreateElemNS in package DOM for C, *3-22*
XmlDomCreateEntityRef in package DOM for C, *3-23*
XmlDomCreateFragment in package DOM for C, *3-24*
XmlDomCreateNodeIter in package Traversal for C, *10-1*
XmlDomCreatePI in package DOM for C, *3-24*
XmlDomCreateRange in package Range for C, *6-1*
XmlDomCreateText in package DOM for C, *3-25*
XmlDomCreateTreeWalker in package Traversal for C, *10-2*
XmlDomDeleteData in package DOM for C, *3-12*
XmlDomFreeNode in package DOM for C, *3-67*
XmlDomFreeNodeList in package DOM for C, *3-97*
XmlDomFreeString in package DOM for C, *3-26*
XmlDomGetAttr in package DOM for C, *3-43*
XmlDomGetAttrLocal in package DOM for C, *3-2*
XmlDomGetAttrLocalLen in package DOM for C, *3-2*
XmlDomGetAttrName in package DOM for C, *3-3*
XmlDomGetAttrNameLen in package DOM for C, *3-4*
XmlDomGetAttrNode in package DOM for C, *3-44*
XmlDomGetAttrNodeNS in package DOM for C, *3-45*

methods *(continued)*
XmlDomGetAttrNS in package DOM for C, *3-43*
XmlDomGetAttrPrefix in package DOM for C, *3-5*
XmlDomGetAttrs in package DOM for C, *3-68*
XmlDomGetAttrSpecified in package DOM for C, *3-5*
XmlDomGetAttrURI in package DOM for C, *3-6*
XmlDomGetAttrURILen in package DOM for C, *3-7*
XmlDomGetAttrValue in package DOM for C, *3-8*
XmlDomGetAttrValueLen in package DOM for C, *3-8*
XmlDomGetAttrValueStream in package DOM for C, *3-9*
XmlDomGetBaseURI in package DOM for C, *3-27*
XmlDomGetCharData in package DOM for C, *3-13*
XmlDomGetCharDataLength in package DOM for C, *3-14*
XmlDomGetChildNodes in package DOM for C, *3-68*
XmlDomGetChildrenByTag in package DOM for C, *3-45*
XmlDomGetChildrenByTagNS in package DOM for C, *3-46*
XmlDomGetDecl in package DOM for C, *3-28*
XmlDomGetDefaultNS in package DOM for C, *3-69*
XmlDomGetDocElem in package DOM for C, *3-29*
XmlDomGetDocElemByID in package DOM for C, *3-29*
XmlDomGetDocElemsByTag in package DOM for C, *3-30*
XmlDomGetDocElemsByTagNS in package DOM for C, *3-31*
XmlDomGetDTD in package DOM for C, *3-27*
XmlDomGetDTDEntities in package DOM for C, *3-38*
XmlDomGetDTDInternalSubset in package DOM for C, *3-39*
XmlDomGetDTDName in package DOM for C, *3-40*
XmlDomGetDTDNotations in package DOM for C, *3-40*
XmlDomGetDTDPubID in package DOM for C, *3-41*

methods *(continued)*
XmlDomGetDTDSysID in package DOM for C, *3-41*
XmlDomGetElemsByTag in package DOM for C, *3-47*
XmlDomGetElemsByTagNS in package DOM for C, *3-47*
XmlDomGetEntityNotation in package DOM for C, *3-55*
XmlDomGetEntityPubID in package DOM for C, *3-55*
XmlDomGetEntitySysID in package DOM for C, *3-56*
XmlDomGetEntityType in package DOM for C, *3-57*
XmlDomGetFirstChild in package DOM for C, *3-69*
XmlDomGetFirstPfnsPair in package DOM for C, *3-70*
XmlDomGetLastChild in package DOM for C, *3-70*
XmlDomGetLastError in package DOM for C, *3-32*
XmlDomGetNamedItem in package DOM for C, *3-58*
XmlDomGetNamedItemNS in package DOM for C, *3-58*
XmlDomGetNextPfnsPair in package DOM for C, *3-71*
XmlDomGetNextSibling in package DOM for C, *3-72*
XmlDomGetNodeListItem in package DOM for C, *3-97*
XmlDomGetNodeListLength in package DOM for C, *3-98*
XmlDomGetNodeLocal in package DOM for C, *3-72*
XmlDomGetNodeLocalLen in package DOM for C, *3-73*
XmlDomGetNodeMapItem in package DOM for C, *3-59*
XmlDomGetNodeMapLength in package DOM for C, *3-60*
XmlDomGetNodeName in package DOM for C, *3-74*
XmlDomGetNodeNameLen in package DOM for C, *3-74*
XmlDomGetNodePrefix in package DOM for C, *3-75*
XmlDomGetNodeType in package DOM for C, *3-76*
XmlDomGetNodeURI in package DOM for C, *3-77*
XmlDomGetNodeURILen in package DOM for C, *3-78*
methods *(continued)*
XmlDomGetNodeValue in package DOM for C, *3-79*
XmlDomGetNodeValueLen in package DOM for C, *3-79*
XmlDomGetNodeValueStream in package DOM for C, *3-80*
XmlDomGetNotationPubID in package DOM for C, *3-99*
XmlDomGetNotationSysID in package DOM for C, *3-99*
XmlDomGetOwnerDocument in package DOM for C, *3-81*
XmlDomGetOwnerElem in package DOM for C, *3-10*
XmlDomGetParentNode in package DOM for C, *3-81*
XmlDomGetPIData in package DOM for C, *3-100*
XmlDomGetPITarget in package DOM for C, *3-101*
XmlDomGetPrevSibling in package DOM for C, *3-82*
XmlDomGetPullNodeAsBinaryStream in package DOM for C, *3-82*
XmlDomGetPullNodeAsCharacterStream in package DOM for C, *3-83*
XmlDomGetPushNodeAsBinaryStream in package DOM for C, *3-83*
XmlDomGetPushNodeAsCharacterStream in package DOM for C, *3-84*
XmlDomGetSchema in package DOM for C, *3-32*
XmlDomGetSourceEntity in package DOM for C, *3-84*
XmlDomGetSourceLine in package DOM for C, *3-85*
XmlDomGetSourceLocation in package DOM for C, *3-85*
XmlDomGetTag in package DOM for C, *3-48*
XmlDomHasAttr in package DOM for C, *3-49*
XmlDomHasAttrNS in package DOM for C, *3-49*
XmlDomHasAttrs in package DOM for C, *3-85*
XmlDomHasChildNodes in package DOM for C, *3-86*
XmlDomImportNode in package DOM for C, *3-33*
XmlDomInsertBefore in package DOM for C, *3-86*
XmlDomInsertData in package DOM for C, *3-14*
XmlDomIsSchemaBased in package DOM for C, *3-34*

methods *(continued)*
XmlDomIterDetach in package Traversal for C, *10-4*
XmlDomIterNextNode in package Traversal for C, *10-5*
XmlDomIterPrevNode in package Traversal for C, *10-5*
XmlDomNormalize in package DOM for C, *3-87*
XmlDomNumAttrs in package DOM for C, *3-87*
XmlDomNumChildNodes in package DOM for C, *3-88*
XmlDomPrefixToURI in package DOM for C, *3-88*
XmlDomRangeClone in package Range for C, *6-2*
XmlDomRangeCloneContents in package Range for C, *6-2*
XmlDomRangeCollapse in package Range for C, *6-3*
XmlDomRangeCompareBoundaryPoints in package Range for C, *6-3*
XmlDomRangeDeleteContents in package Range for C, *6-4*
XmlDomRangeDetach in package Range for C, *6-4*
XmlDomRangeExtractContents in package Range for C, *6-5*
XmlDomRangeGetCollapsed in package Range for C, *6-5*
XmlDomRangeGetCommonAncestor in package Range for C, *6-6*
XmlDomRangeGetDetached in package Range for C, *6-6*
XmlDomRangeGetEndContainer in package Range for C, *6-7*
XmlDomRangeGetEndOffset in package Range for C, *6-7*
XmlDomRangeGetStartContainer in package Range for C, *6-8*
XmlDomRangeGetStartOffset in package Range for C, *6-8*
XmlDomRangeIsConsistent in package Range for C, *6-9*
XmlDomRangeSelectNode in package Range for C, *6-9*
XmlDomRangeSelectNodeContents in package Range for C, *6-10*
XmlDomRangeSetEnd in package Range for C, *6-10*
XmlDomRangeSetEndBefore in package Range for C, *6-11*
XmlDomRangeSetStart in package Range for C, *6-11*

methods *(continued)*
XmlDomRangeSetStartAfter in package Range for C, *6-12*
XmlDomRangeSetStartBefore in package Range for C, *6-13*
XmlDomRemoveAttr in package DOM for C, *3-50*
XmlDomRemoveAttrNode in package DOM for C, *3-51*
XmlDomRemoveAttrNS in package DOM for C, *3-51*
XmlDomRemoveChild in package DOM for C, *3-89*
XmlDomRemoveNamedItem in package DOM for C, *3-60*
XmlDomRemoveNamedItemNS in package DOM for C, *3-61*
XmlDomRenameNode in package DOM for C, *3-89*
XmlDomRenameNodeNS in package DOM for C, *3-90*
XmlDomReplaceChild in package DOM for C, *3-91*
XmlDomReplaceData in package DOM for C, *3-15*
XmlDomSaveString in package DOM for C, *3-34*
XmlDomSaveString2 in package DOM for C, *3-35*
XmlDomSetAttr in package DOM for C, *3-52*
XmlDomSetAttrNode in package DOM for C, *3-53*
XmlDomSetAttrNodeNS in package DOM for C, *3-54*
XmlDomSetAttrNS in package DOM for C, *3-53*
XmlDomSetAttrValue in package DOM for C, *3-10*
XmlDomSetAttrValueStream in package DOM for C, *3-11*
XmlDomSetBaseURI in package DOM for C, *3-35*
XmlDomSetCharData in package DOM for C, *3-16*
XmlDomSetDefaultNS in package DOM for C, *3-91*
XmlDomSetDocOrder in package DOM for C, *3-37*
XmlDomSetDTD in package DOM for C, *3-36*
XmlDomSetLastError in package DOM for C, *3-37*
XmlDomSetNamedItem in package DOM for C, *3-62*
XmlDomSetNamedItemNS in package DOM for C, *3-62*

methods *(continued)*
XmlDomSetNodePrefix in package DOM for C, *3-92*
XmlDomSetNodeValue in package DOM for C, *3-92*
XmlDomSetNodeValueLen in package DOM for C, *3-93*
XmlDomSetNodeValueStream in package DOM for C, *3-94*
XmlDomSetPIData in package DOM for C, *3-101*
XmlDomSetPullNodeAsBinaryStream in package DOM for C, *3-94*
XmlDomSetPullNodeAsCharacterStream in package DOM for C, *3-95*
XmlDomSetPushNodeAsBinaryStream in package DOM for C, *3-95*
XmlDomSetPushNodeAsCharacterStream in package DOM for C, *3-96*
XmlDomSplitText in package DOM for C, *3-102*
XmlDomSubstringData in package DOM for C, *3-16*
XmlDomSync in package DOM for C, *3-38*
XmlDomValidate in package DOM for C, *3-96*
XmlDomWalkerFirstChild in package Traversal for C, *10-7*
XmlDomWalkerGetCurrentNode in package Traversal for C, *10-7*
XmlDomWalkerGetRoot in package Traversal for C, *10-8*
XmlDomWalkerLastChild in package Traversal for C, *10-8*
XmlDomWalkerNextNode in package Traversal for C, *10-9*
XmlDomWalkerNextSibling in package Traversal for C, *10-9*
XmlDomWalkerParentNode in package Traversal for C, *10-10*
XmlDomWalkerPrevNode in package Traversal for C, *10-11*
XmlDomWalkerPrevSibling in package Traversal for C, *10-11*
XmlDomWalkerSetCurrentNode in package Traversal for C, *10-12*
XmlDomWalkerSetRoot in package Traversal for C, *10-12*
XmlEvCleanPPCtx package Event for C, *4-5*
XmlEvCreatePPCtx package Event for C, *4-5*
XmlEvCreateSVCtx package Event for C, *4-6*
XmlEvDestroyPPCtx package Event for C, *4-7*
XmlEvDestroySVCtx package Event for C, *4-7*

methods *(continued)*
XmlEvGetAttrCount in package Event for C, *4-8*
XmlEvGetAttrDeclBody in package Event for C, *4-8*
XmlEvGetAttrDeclBody0 in package Event for C, *4-9*
XmlEvGetAttrDeclCount in package Event for C, *4-9*
XmlEvGetAttrDeclElName in package Event for C, *4-9*
XmlEvGetAttrDeclElName0 in package Event for C, *4-10*
XmlEvGetAttrDeclLocalName in package Event for C, *4-10*
XmlEvGetAttrDeclLocalName0 in package Event for C, *4-11*
XmlEvGetAttrDeclName in package Event for C, *4-11*
XmlEvGetAttrDeclName0 in package Event for C, *4-12*
XmlEvGetAttrDeclPrefix in package Event for C, *4-12*
XmlEvGetAttrDeclPrefix0 in package Event for C, *4-13*
XmlEvGetAttrID in package Event for C, *4-13*
XmlEvGetAttrLocalName in package Event for C, *4-14*
XmlEvGetAttrLocalName0 in package Event for C, *4-14*
XmlEvGetAttrName in package Event for C, *4-14*
XmlEvGetAttrName0 in package Event for C, *4-15*
XmlEvGetAttrPrefix in package Event for C, *4-15*
XmlEvGetAttrPrefix0 in package Event for C, *4-16*
XmlEvGetAttrURI in package Event for C, *4-16*
XmlEvGetAttrURI0 in package Event for C, *4-17*
XmlEvGetAttrUriID in package Event for C, *4-17*
XmlEvGetAttrValue in package Event for C, *4-18*
XmlEvGetAttrValue0 in package Event for C, *4-18*
XmlEvGetElDeclContent in package Event for C, *4-18*
XmlEvGetElDeclContent0 in package Event for C, *4-19*
XmlEvGetEncoding in package Event for C, *4-19*
XmlEvGetError in package Event for C, *4-20*

methods *(continued)*
XmlEvGetLocalName in package Event for C, *4-21*
XmlEvGetLocalName0 in package Event for C, *4-22*
XmlEvGetLocation in package Event for C, *4-22*
XmlEvGetName in package Event for C, *4-20*
XmlEvGetName0 in package Event for C, *4-21*
XmlEvGetPERepl in package Event for C, *4-25*
XmlEvGetPERepl0 in package Event for C, *4-24*, *4-25*
XmlEvGetPIData in package Event for C, *4-23*
XmlEvGetPIData0 in package Event for C, *4-23*
XmlEvGetPITarget in package Event for C, *4-23*
XmlEvGetPITarget0 in package Event for C, *4-24*
XmlEvGetPrefix in package Event for C, *4-25*
XmlEvGetPrefix0 in package Event for C, *4-26*
XmlEvGetPubId in package Event for C, *4-26*
XmlEvGetPubId0 in package Event for C, *4-27*
XmlEvGetSysId in package Event for C, *4-27*
XmlEvGetSysId0 in package Event for C, *4-28*
XmlEvGetTagID in package Event for C, *4-28*
XmlEvGetTagUriID in package Event for C, *4-29*
XmlEvGetText in package Event for C, *4-29*
XmlEvGetText0 in package Event for C, *4-30*
XmlEvGetUENdata in package Event for C, *4-30*
XmlEvGetUENdata0 in package Event for C, *4-31*
XmlEvGetURI in package Event for C, *4-31*
XmlEvGetURI0 in package Event for C, *4-31*
XmlEvGetVersion in package Event for C, *4-32*
XmlEvIsEncodingSpecified in package Event for C, *4-32*
XmlEvIsStandalone in package Event for C, *4-33*
XmlEvLoadPPDoc in package Event for C, *4-34*
XmlEvNamespaceAttr in package Event for C, *4-33*
XmlEvNext in package Event for C, *4-33*
XmlEvSchemaValidate in package Event for C, *4-35*

methods *(continued)*
XmlFreeDocument in package XML for C, *11-8*
XmlGetEncoding in package XML for C, *11-8*
XmlHasFeature in package XML for C, *11-9*
XmlHash in package XmlDiff for C, *12-3*
XmlIsSimple in package XML for C, *11-10*
XmlIsUnicode in package XML for C, *11-10*
XmlLoadDom in package XML for C, *11-11*
XmlLoadSax in package XML for C, *11-12*
XmlLoadSaxVA in package XML for C, *11-13*
XmlPatch in package XmlDiff for C, *12-4*
XmlSaveDom in package XML for C, *11-13*
XmlSaxAttributeDecl in package SAX for C, *7-1*
XmlSaxBeginGen in package SAX for C, *7-2*
XmlSaxCDATA in package SAX for C, *7-2*
XmlSaxCharacters in package SAX for C, *7-3*
XmlSaxComment in package SAX for C, *7-4*
XmlSaxElementDecl in package SAX for C, *7-4*
XmlSaxEndDocument in package SAX for C, *7-5*
XmlSaxEndElement in package SAX for C, *7-5*
XmlSaxEndGen in package SAX for C, *7-5*
XmlSaxNotationDecl in package SAX for C, *7-6*
XmlSaxParsedEntityDecl in package SAX for C, *7-6*
XmlSaxPI in package SAX for C, *7-6*
XmlSaxStartDocument in package SAX for C, *7-7*
XmlSaxStartElement in package SAX for C, *7-8*
XmlSaxStartElementNS in package SAX for C, *7-8*
XmlSaxUnparsedEntityDecl in package SAX for C, *7-9*
XmlSaxWhitespace in package SAX for C, *7-10*
XmlSaxXmlDecl in package SAX for C, *7-11*
XmlSchemaClean in package Schema for C, *8-8*
XmlSchemaCreate in package Schema for C, *8-1*
XmlSchemaDestroy in package Schema for C, *8-2*
XmlSchemaErrorWhere in package Schema for C, *8-2*
XmlSchemaLoad in package Schema for C, *8-3*
XmlSchemaLoadedList in package Schema for C, *8-4*

methods *(continued)*
XmlSchemaSetErrorHandler in package Schema for C, *8-4*
XmlSchemaSetValidateOptions in package Schema for C, *8-5*
XmlSchemaTargetNamespace in package Schema for C, *8-6*
XmlSchemaUnload in package Schema for C, *8-6*
XmlSchemaValidate in package Schema for C, *8-7*
XmlSchemaVersion in package Schema for C, *8-8*
XmlSoapAddBodyElement in package SOAP for C, *9-2*
XmlSoapAddFaultReason in package SOAP for C, *9-3*
XmlSoapAddFaultSubDetail in package SOAP for C, *9-3*
XmlSoapAddHeaderElement in package SOAP for C, *9-4*
XmlSoapCall in package SOAP for C, *9-5*
XmlSoapCreateConnection in package SOAP for C, *9-5*
XmlSoapCreateCtx in package SOAP for C, *9-7*
XmlSoapCreateMsg in package SOAP for C, *9-8*
XmlSoapDestroyConnection in package SOAP for C, *9-8*
XmlSoapDestroyCtx in package SOAP for C, *9-9*
XmlSoapDestroyMsg in package SOAP for C, *9-9*
XmlSoapError in package SOAP for C, *9-10*
XmlSoapGetBody in package SOAP for C, *9-10*
XmlSoapGetBodyElement in package SOAP for C, *9-11*
XmlSoapGetEnvelope in package SOAP for C, *9-12*
XmlSoapGetFault in package SOAP for C, *9-12*
XmlSoapGetHeader in package SOAP for C, *9-13*
XmlSoapGetHeaderElement in package SOAP for C, *9-14*
XmlSoapGetMustUnderstand in package SOAP for C, *9-14*
XmlSoapGetReasonLang in package SOAP for C, *9-15*
XmlSoapGetReasonNum in package SOAP for C, *9-16*
XmlSoapGetRelay in package SOAP for C, *9-16*
methods *(continued)*
XmlSoapGetRole in package SOAP for C, *9-17*
XmlSoapHasFault in package SOAP for C, *9-17*
XmlSoapSetFault in package SOAP for C, *9-18*
XmlSoapSetMustUnderstand in package SOAP for C, *9-19*
XmlSoapSetRelay in package SOAP for C, *9-20*
XmlSoapSetRole in package SOAP for C, *9-20*
XmlVersion in package XML for C, *11-14*
XmlXPathCreateCtx in package XPath for C, *13-1*
XmlXPathDestroyCtx in package XPath for C, *13-2*
XmlXPathEval in package XPath for C, *13-2*
XmlXPathGetObjectBoolean in package XPath for C, *13-3*
XmlXPathGetObjectFragment in package XPath for C, *13-3*
XmlXPathGetObjectNSetNode in package XPath for C, *13-4*
XmlXPathGetObjectNSetNum in package XPath for C, *13-4*
XmlXPathGetObjectNumber in package XPath for C, *13-5*
XmlXPathGetObjectString in package XPath for C, *13-5*
XmlXPathGetObjectType in package XPath for C, *13-6*
XmlXPathParse in package XPath for C, *13-6*
XmlXPointerEval in package XPointer for C, *14-1*
XmlXPtrLocGetNode in package XPointer for C, *14-2*
XmlXPtrLocGetPoint in package XPointer for C, *14-2*
XmlXPtrLocGetRange in package XPointer for C, *14-2*
XmlXPtrLocGetType in package XPointer for C, *14-3*
XmlXPtrLocSetFree in package XPointer for C, *14-4*
XmlXPtrLocSetGetItem in package XPointer for C, *14-4*
XmlXPtrLocSetGetLength in package XPointer for C, *14-5*
XmlXPtrLocToString in package XPointer for C, *14-3*
XmlXslCreate in package XSLT for C, *15-1*
XmlXslDestroy in package XSLT for C, *15-2*

methods *(continued)*
XmlXslGetBaseURI in package XSLT for C, *15-2*
XmlXslGetOutput in package XSLT for C, *15-3*
XmlXslGetStylesheetDom in package XSLT for C, *15-3*
XmlXslGetTextParam in package XSLT for C, *15-3*
XmlXslProcess in package XSLT for C, *15-4*
XmlXslResetAllParams in package XSLT for C, *15-4*
XmlXslSetOutputDom in package XSLT for C, *15-5*
XmlXslSetOutputEncoding in package XSLT for C, *15-5*
XmlXslSetOutputMethod in package XSLT for C, *15-5*
XmlXslSetOutputSax in package XSLT for C, *15-6*
XmlXslSetOutputStream in package XSLT for C, *15-6*
XmlXslSetTextParam in package XSLT for C, *15-7*
XMLXVM_DEBUG_F in package XSLTVM for C, *16-7*
XmlXvmCompileBuffer in package XSLTVM for C, *16-1*
XmlXvmCompileDom in package XSLTVM for C, *16-1*
XmlXvmCompileFile in package XSLTVM for C, *16-2*
XmlXvmCompileURI in package XSLTVM for C, *16-3*
XmlXvmCompileXPath in package XSLTVM for C, *16-4*
XmlXvmCreate in package XSLTVM for C, *16-7*
XmlXvmCreateComp in package XSLTVM for C, *16-4*
XmlXvmDestroy in package XSLTVM for C, *16-8*
XmlXvmDestroyComp in package XSLTVM for C, *16-5*
XmlXvmEvaluateXPath in package XSLTVM for C, *16-8*
XmlXvmGetBytecodeLength in package XSLTVM for C, *16-5*
XmlXvmGetObjectBoolean in package XSLTVM for C, *16-9*
XmlXvmGetObjectNSetNode in package XSLTVM for C, *16-10*
XmlXvmGetObjectNSetNum in package XSLTVM for C, *16-10*
methods *(continued)*
XmlXvmGetObjectNumber in package XSLTVM for C, *16-11*
XmlXvmGetObjectString in package XSLTVM for C, *16-11*
XmlXvmGetObjectType in package XSLTVM for C, *16-12*
XmlXvmGetOutputDom in package XSLTVM for C, *16-12*
XmlXvmResetParams in package XSLTVM for C, *16-13*
XmlXvmSetBaseURI in package XSLTVM for C, *16-13*
XmlXvmSetBytecodeBuffer in package XSLTVM for C, *16-13*
XmlXvmSetBytecodeFile in package XSLTVM for C, *16-14*
XmlXvmSetBytecodeURI in package XSLTVM for C, *16-14*
XmlXvmSetDebugFunc in package XSLTVM for C, *16-15*
XmlXvmSetOutputDom in package XSLTVM for C, *16-15*
XmlXvmSetOutputEncoding in package XSLTVM for C, *16-16*
XmlXvmSetOutputSax in package XSLTVM for C, *16-16*
XmlXvmSetOutputStream in package XSLTVM for C, *16-17*
XmlXvmSetTextParam in package XSLTVM for C, *16-17*
XmlXvmTransformBuffer in package XSLTVM for C, *16-18*
XmlXvmTransformDom in package XSLTVM for C, *16-18*
XmlXvmTransformFile in package XSLTVM for C, *16-19*
XmlXvmTransformURI in package XSLTVM for C, *16-20*

P

packages
Callback for C, *2-1*
DOM for C, *3-1*
Event for C, *4-1*
Range for C, *6-1*
SAX for C, *7-1*
Schema for C, *8-1*
SOAP for C, *9-1*
Traversal for C, *10-1*
XML for C, *11-1*
XmlDiff for C, *12-1*
XPath for C, *13-1*
XPointer for C, *14-1*

packages *(continued)*
XSLT for C, *15-1*
XSLTVM for C, *16-1*

R

Range package for C, *6-1*

S

SAX package for C, *7-1*
Scalar type, *17-16*
Schema package for C, *8-1*
SOAP package for C, *9-1*

T

Traversal package for C, *10-1*

X

XML package for C, *11-1*
XML_ACCESS_CLOSE_F in package Callback package for C, *2-1*
XML_ACCESS_OPEN_F in package Callback package for C, *2-1*
XML_ACCESS_READ_F in package Callback package for C, *2-1*
XML_ALLOC_F in package Callback package for C, *2-1*
XML_ERRMSG_F in package Callback package for C, *2-2*
XML_FREE_F in package Callback package for C, *2-2*
XML_STREAM_CLOSE_F in package Callback package for C, *2-2*
XML_STREAM_OPEN_F in package Callback package for C, *2-2*
XML_STREAM_READ_F in package Callback package for C, *2-2*
XML_STREAM_WRITE_F in package Callback package for C, *2-3*
XmlAccess in package XML package for C, *11-1*
XmlCreate in package XML package for C, *11-2*
XmlCreateDocument in package XML package for C, *11-6*
XmlCreateDTD in package XML package for C, *11-5*
XmlDestroy in package XML package for C, *11-6*
XmlDiff in package XML package for C, *11-7*
XmlDiff in package XmlDiff package for C, *12-1*
XmlDiff package for C, *12-1*
XMLDOM_ACCEPT_NODE_F in package Traversal package for C, *10-3*
XmlDomAppendChild in package DOM package for C, *3-65*
XmlDomAppendData in package DOM package for C, *3-12*
XmlDomCleanNode in package DOM package for C, *3-66*
XmlDomCloneNode in package DOM package for C, *3-66*
XmlDomCreateAttr in package DOM package for C, *3-18*
XmlDomCreateAttrNS in package DOM package for C, *3-19*
XmlDomCreateCDATA in package DOM package for C, *3-20*
XmlDomCreateComment in package DOM package for C, *3-21*
XmlDomCreateElem in package DOM package for C, *3-22*
XmlDomCreateElemNS in package DOM package for C, *3-22*
XmlDomCreateEntityRef in package DOM package for C, *3-23*
XmlDomCreateFragment in package DOM package for C, *3-24*
XmlDomCreateNodeIter in package Traversal package for C, *10-1*
XmlDomCreatePI in package DOM package for C, *3-24*
XmlDomCreateRange in package Range package for C, *6-1*
XmlDomCreateText in package DOM package for C, *3-25*
XmlDomCreateTreeWalker in package Traversal package for C, *10-2*
XmlDomDeleteData in package DOM package for C, *3-12*
XmlDomFreeNode in package DOM package for C, *3-67*
XmlDomFreeNodeList in package DOM package for C, *3-97*
XmlDomFreeString in package DOM package for C, *3-26*
XmlDomGetAttr in package DOM package for C, *3-43*
XmlDomGetAttrLocal in package DOM package for C, *3-2*
XmlDomGetAttrLocalLen in package DOM package for C, *3-2*
XmlDomGetAttrName in package DOM package for C, *3-3*
XmlDomGetAttrNameLen in package DOM package for C, *3-4*
XmlDomGetAttrNode in package DOM package for C, *3-44*

XmlDomGetAttrNodeNS in package DOM package for C, 3-45
XmlDomGetAttrNS in package DOM package for C, 3-43
XmlDomGetAttrPrefix in package DOM package for C, 3-5
XmlDomGetAttrs in package DOM package for C, 3-68
XmlDomGetAttrSpecified in package DOM package for C, 3-5
XmlDomGetAttrURI in package DOM package for C, 3-6
XmlDomGetAttrURILen in package DOM package for C, 3-7
XmlDomGetAttrValue in package DOM package for C, 3-8
XmlDomGetAttrValueLen in package DOM package for C, 3-8
XmlDomGetAttrValueStream in package DOM package for C, 3-9
XmlDomGetBaseURI in package DOM package for C, 3-27
XmlDomGetCharData in package DOM package for C, 3-13
XmlDomGetCharDataLength in package DOM package for C, 3-14
XmlDomGetChildNodes in package DOM package for C, 3-68
XmlDomGetChildrenByTag in package DOM package for C, 3-45
XmlDomGetChildrenByTagNS in package DOM package for C, 3-46
XmlDomGetDecl in package DOM package for C, 3-28
XmlDomGetDefaultNS in package DOM package for C, 3-69
XmlDomGetDocElem in package DOM package for C, 3-29
XmlDomGetDocElemByID in package DOM package for C, 3-29
XmlDomGetDocElemsByTag in package DOM package for C, 3-30
XmlDomGetDocElemsByTagNS in package DOM package for C, 3-31
XmlDomGetDTD in package DOM package for C, 3-27
XmlDomGetDTDEntities in package DOM package for C, 3-38
XmlDomGetDTDInternalSubset in package DOM package for C, 3-39
XmlDomGetDTDName in package DOM package for C, 3-40
XmlDomGetDTDNotations in package DOM package for C, 3-40
XmlDomGetDTDPubID in package DOM package for C, 3-41
XmlDomGetDTDSysID in package DOM package for C, 3-41
XmlDomGetElemsByTag in package DOM package for C, 3-47
XmlDomGetElemsByTagNS in package DOM package for C, 3-47
XmlDomGetEntityNotation in package DOM package for C, 3-55
XmlDomGetEntityPubID in package DOM package for C, 3-55
XmlDomGetEntitySysID in package DOM package for C, 3-56
XmlDomGetEntityType in package DOM package for C, 3-57
XmlDomGetFirstChild in package DOM package for C, 3-69
XmlDomGetFirstPfnsPair in package DOM package for C, 3-70
XmlDomGetLastChild in package DOM package for C, 3-70
XmlDomGetLastError in package DOM package for C, 3-32
XmlDomGetNamedItem in package DOM package for C, 3-58
XmlDomGetNamedItemNS in package DOM package for C, 3-58
XmlDomGetNextPfnsPair in package DOM package for C, 3-71
XmlDomGetNextSibling in package DOM package for C, 3-72
XmlDomGetNodeListItem in package DOM package for C, 3-97
XmlDomGetNodeListLength in package DOM package for C, 3-98
XmlDomGetNodeLocal in package DOM package for C, 3-72
XmlDomGetNodeLocalLen in package DOM package for C, 3-73
XmlDomGetNodeMapItem in package DOM package for C, 3-59
XmlDomGetNodeMapLength in package DOM package for C, 3-60
XmlDomGetNodeName in package DOM package for C, 3-74
XmlDomGetNodeNameLen in package DOM package for C, 3-74
XmlDomGetNodePrefix in package DOM package for C, 3-75
XmlDomGetNodeType in package DOM package for C, 3-76
XmlDomGetNodeURI in package DOM package for C, 3-77

XmlDomGetNodeURILen in package DOM package for C, *3-78*
XmlDomGetNodeValue in package DOM package for C, *3-79*
XmlDomGetNodeValueLen in package DOM package for C, *3-79*
XmlDomGetNodeValueStream in package DOM package for C, *3-80*
XmlDomGetNotationPubID in package DOM package for C, *3-99*
XmlDomGetNotationSysID in package DOM package for C, *3-99*
XmlDomGetOwnerDocument in package DOM package for C, *3-81*
XmlDomGetOwnerElem in package DOM package for C, *3-10*
XmlDomGetParentNode in package DOM package for C, *3-81*
XmlDomGetPIData in package DOM package for C, *3-100*
XmlDomGetPITarget in package DOM package for C, *3-101*
XmlDomGetPrevSibling in package DOM package for C, *3-82*
XmlDomGetPullNodeAsBinaryStream in package DOM package for C, *3-82*
XmlDomGetPullNodeAsCharacterStream in package DOM package for C, *3-83*
XmlDomGetPushNodeAsBinaryStream in package DOM package for C, *3-83*
XmlDomGetPushNodeAsCharacterStream in package DOM package for C, *3-84*
XmlDomGetSchema in package DOM package for C, *3-32*
XmlDomGetSourceEntity in package DOM package for C, *3-84*
XmlDomGetSourceLine in package DOM package for C, *3-85*
XmlDomGetSourceLocation in package DOM package for C, *3-85*
XmlDomGetTag in package DOM package for C, *3-48*
XmlDomHasAttr in package DOM package for C, *3-49*
XmlDomHasAttrNS in package DOM package for C, *3-49*
XmlDomHasAttrs in package DOM package for C, *3-85*
XmlDomHasChildNodes in package DOM package for C, *3-86*
XmlDomImportNode in package DOM package for C, *3-33*
XmlDomInsertBefore in package DOM package for C, *3-86*
XmlDomInsertData in package DOM package for C, *3-14*
XmlDomIsSchemaBased in package DOM package for C, *3-34*
XmlDomIterDetach in package Traversal package for C, *10-4*
XmlDomIterNextNode in package Traversal package for C, *10-5*
XmlDomIterPrevNode in package Traversal package for C, *10-5*
XmlDomNormalize in package DOM package for C, *3-87*
XmlDomNumAttrs in package DOM package for C, *3-87*
XmlDomNumChildNodes in package DOM package for C, *3-88*
XmlDomPrefixToURI in package DOM package for C, *3-88*
XmlDomRangeClone in package Range package for C, *6-2*
XmlDomRangeCloneContents in package Range package for C, *6-2*
XmlDomRangeCollapse in package Range package for C, *6-3*
XmlDomRangeCompareBoundaryPoints in package Range package for C, *6-3*
XmlDomRangeDeleteContents in package Range package for C, *6-4*
XmlDomRangeDetach in package Range package for C, *6-4*
XmlDomRangeExtractContents in package Range package for C, *6-5*
XmlDomRangeGetCollapsed in package Range package for C, *6-5*
XmlDomRangeGetCommonAncestor in package Range package for C, *6-6*
XmlDomRangeGetDetached in package Range package for C, *6-6*
XmlDomRangeGetEndContainer in package Range package for C, *6-7*
XmlDomRangeGetEndOffset in package Range package for C, *6-7*
XmlDomRangeGetStartContainer in package Range package for C, *6-8*
XmlDomRangeGetStartOffset in package Range package for C, *6-8*
XmlDomRangeIsConsistent in package Range package for C, *6-9*
XmlDomRangeSelectNode in package Range package for C, *6-9*
XmlDomRangeSelectNodeContents in package Range package for C, *6-10*
XmlDomRangeSetEnd in package Range package for C, *6-10*

XmlDomRangeSetEndBefore in package Range package for C, *6-11*
XmlDomRangeSetStart in package Range package for C, *6-11*
XmlDomRangeSetStartAfter in package Range package for C, *6-12*
XmlDomRangeSetStartBefore in package Range package for C, *6-13*
XmlDomRemoveAttr in package DOM package for C, *3-50*
XmlDomRemoveAttrNode in package DOM package for C, *3-51*
XmlDomRemoveAttrNS in package DOM package for C, *3-51*
XmlDomRemoveChild in package DOM package for C, *3-89*
XmlDomRemoveNamedItem in package DOM package for C, *3-60*
XmlDomRemoveNamedItemNS in package DOM package for C, *3-61*
XmlDomRenameNode in package DOM package for C, *3-89*
XmlDomRenameNodeNS in package DOM package for C, *3-90*
XmlDomReplaceChild in package DOM package for C, *3-91*
XmlDomReplaceData in package DOM package for C, *3-15*
XmlDomSaveString in package DOM package for C, *3-34*
XmlDomSaveString2 in package DOM package for C, *3-35*
XmlDomSetAttr in package DOM package for C, *3-52*
XmlDomSetAttrNode in package DOM package for C, *3-53*
XmlDomSetAttrNodeNS in package DOM package for C, *3-54*
XmlDomSetAttrNS in package DOM package for C, *3-53*
XmlDomSetAttrValue in package DOM package for C, *3-10*
XmlDomSetAttrValueStream in package DOM package for C, *3-11*
XmlDomSetBaseURI in package DOM package for C, *3-35*
XmlDomSetCharData in package DOM package for C, *3-16*
XmlDomSetDefaultNS in package DOM package for C, *3-91*
XmlDomSetDocOrder in package DOM package for C, *3-37*
XmlDomSetDTD in package DOM package for C, *3-36*
XmlDomSetLastError in package DOM package for C, *3-37*
XmlDomSetNamedItem in package DOM package for C, *3-62*
XmlDomSetNamedItemNS in package DOM package for C, *3-62*
XmlDomSetNodePrefix in package DOM package for C, *3-92*
XmlDomSetNodeValue in package DOM package for C, *3-92*
XmlDomSetNodeValueLen in package DOM package for C, *3-93*
XmlDomSetNodeValueStream in package DOM package for C, *3-94*
XmlDomSetPIData in package DOM package for C, *3-101*
XmlDomSetPullNodeAsBinaryStream in package DOM package for C, *3-94*
XmlDomSetPullNodeAsCharacterStream in package DOM package for C, *3-95*
XmlDomSetPushNodeAsBinaryStream in package DOM package for C, *3-95*
XmlDomSetPushNodeAsCharacterStream in package DOM package for C, *3-96*
XmlDomSplitText in package DOM package for C, *3-102*
XmlDomSubstringData in package DOM package for C, *3-16*
XmlDomSync in package DOM package for C, *3-38*
XmlDomValidate in package DOM package for C, *3-96*
XmlDomWalkerFirstChild in package Traversal package for C, *10-7*
XmlDomWalkerGetCurrentNode in package Traversal package for C, *10-7*
XmlDomWalkerGetRoot in package Traversal package for C, *10-8*
XmlDomWalkerLastChild in package Traversal package for C, *10-8*
XmlDomWalkerNextNode in package Traversal package for C, *10-9*
XmlDomWalkerNextSibling in package Traversal package for C, *10-9*
XmlDomWalkerParentNode in package Traversal package for C, *10-10*
XmlDomWalkerPrevNode in package Traversal package for C, *10-11*
XmlDomWalkerPrevSibling in package Traversal package for C, *10-11*
XmlDomWalkerSetCurrentNode in package Traversal package for C, *10-12*
XmlDomWalkerSetRoot in package Traversal package for C, *10-12*

XmlEvCleanPPCtx in package Event package for C, *4-5*
XmlEvCreatePPCtx in package Event package for C, *4-5*
XmlEvCreateSVCtx in package Event package for C, *4-6*
XmlEvDestroyPPCtx in package Event package for C, *4-7*
XmlEvDestroySVCtx in package Event package for C, *4-7*
XmlEvGetAttrCount in package Event package for C, *4-8*
XmlEvGetAttrDeclBody in package Event package for C, *4-8*
XmlEvGetAttrDeclBody0 in package Event package for C, *4-9*
XmlEvGetAttrDeclCount in package Event package for C, *4-9*
XmlEvGetAttrDeclElName in package Event package for C, *4-9*
XmlEvGetAttrDeclElName0 in package Event package for C, *4-10*
XmlEvGetAttrDeclLocalName in package Event package for C, *4-10*
XmlEvGetAttrDeclLocalName0 in package Event package for C, *4-11*
XmlEvGetAttrDeclName in package Event package for C, *4-11*
XmlEvGetAttrDeclName0 in package Event package for C, *4-12*
XmlEvGetAttrDeclPrefix in package Event package for C, *4-12*
XmlEvGetAttrDeclPrefix0 in package Event package for C, *4-13*
XmlEvGetAttrID in package Event package for C, *4-13*
XmlEvGetAttrLocalName in package Event package for C, *4-14*
XmlEvGetAttrLocalName0 in package Event package for C, *4-14*
XmlEvGetAttrName in package Event package for C, *4-14*
XmlEvGetAttrName0 in package Event package for C, *4-15*
XmlEvGetAttrPrefix in package Event package for C, *4-15*
XmlEvGetAttrPrefix0 in package Event package for C, *4-16*
XmlEvGetAttrURI in package Event package for C, *4-16*
XmlEvGetAttrURI0 in package Event package for C, *4-17*
XmlEvGetAttrUriID in package Event package for C, *4-17*
XmlEvGetAttrValue in package Event package for C, *4-18*
XmlEvGetAttrValue0 in package Event package for C, *4-18*
XmlEvGetElDeclContent in package Event package for C, *4-18*
XmlEvGetElDeclContent0 in package Event package for C, *4-19*
XmlEvGetEncoding in package Event package for C, *4-19*
XmlEvGetError in package Event package for C, *4-20*
XmlEvGetLocalName in package Event package for C, *4-21*
XmlEvGetLocalName0 in package Event package for C, *4-22*
XmlEvGetLocation in package Event package for C, *4-22*
XmlEvGetName in package Event package for C, *4-20*
XmlEvGetName0 in package Event package for C, *4-21*
XmlEvGetPEisGen in package Event package for C, *4-24*
XmlEvGetPERepl in package Event package for C, *4-25*
XmlEvGetPERepl0 in package Event package for C, *4-25*
XmlEvGetPIData in package Event package for C, *4-23*
XmlEvGetPIData0 in package Event package for C, *4-23*
XmlEvGetPITarget in package Event package for C, *4-23*
XmlEvGetPITarget0 in package Event package for C, *4-24*
XmlEvGetPrefix in package Event package for C, *4-25*
XmlEvGetPrefix0 in package Event package for C, *4-26*
XmlEvGetPubId in package Event package for C, *4-26*
XmlEvGetPubId0 in package Event package for C, *4-27*
XmlEvGetSysId in package Event package for C, *4-27*
XmlEvGetSysId0 in package Event package for C, *4-28*
XmlEvGetTagID in package Event package for C, *4-28*
XmlEvGetTagUriID in package Event package for C, *4-29*
XmlEvGetText in package Event package for C, *4-29*

XmlEvGetText0 in package Event package for C, *4-30*
XmlEvGetUENdata in package Event package for C, *4-30*
XmlEvGetUENdata0 in package Event package for C, *4-31*
XmlEvGetURI in package Event package for C, *4-31*
XmlEvGetURI0 in package Event package for C, *4-31*
XmlEvGetVersion in package Event package for C, *4-32*
XmlEvIsEncodingSpecified in package Event package for C, *4-32*
XmlEvIsStandalone in package Event package for C, *4-33*
XmlEvLoadPPDoc in package Event package for C, *4-34*
XmlEvNamespaceAttr in package Event package for C, *4-33*
XmlEvNext in package Event package for C, *4-33*
XmlEvNextTag in package Event package for C, *4-34*
XmlEvSchemaValidate in package Event package for C, *4-35*
XmlFreeDocument in package XML package for C, *11-8*
XmlGetEncoding in package XML package for C, *11-8*
XmlHasFeature in package XML package for C, *11-9*
XmlHash in package XPath package for C, *12-3*
XmlIsSimple in package XML package for C, *11-10*
XmlIsUnicode in package XML package for C, *11-10*
XmlLoadDom in package XML package for C, *11-11*
XmlLoadSax in package XML package for C, *11-12*
XmlLoadSaxVA in package XML package for C, *11-13*
XmlPatch in package XmlDiff package for C, *12-4*
XmlSaveDom in package XML package for C, *11-13*
XmlSaxAttributeDecl in package SAX package for C, *7-1*
XmlSaxBeginGen in package SAX package for C, *7-2*
XmlSaxCDATA in package SAX package for C, *7-2*
XmlSaxCharacters in package SAX package for C, *7-3*
XmlSaxComment in package SAX package for C, *7-4*
XmlSaxElementDecl in package SAX package for C, *7-4*
XmlSaxEndDocument in package SAX package for C, *7-5*
XmlSaxEndElement in package SAX package for C, *7-5*
XmlSaxEndGen in package SAX package for C, *7-5*
XmlSaxNotationDecl in package SAX package for C, *7-6*
XmlSaxParsedEntityDecl in package SAX package for C, *7-6*
XmlSaxPI in package SAX package for C, *7-6*
XmlSaxStartDocument in package SAX package for C, *7-7*
XmlSaxStartElement in package SAX package for C, *7-8*
XmlSaxStartElementNS in package SAX package for C, *7-8*
XmlSaxUnparsedEntityDecl in package SAX package for C, *7-9*
XmlSaxWhitespace in package SAX package for C, *7-10*
XmlSaxXmlDecl in package SAX package for C, *7-11*
XmlSchemaClean in package Schema package for C, *8-8*
XmlSchemaCreate in package Schema package for C, *8-1*
XmlSchemaDestroy in package Schema package for C, *8-2*
XmlSchemaErrorWhere in package Schema package for C, *8-2*
XmlSchemaLoad in package Schema package for C, *8-3*
XmlSchemaLoadedList in package Schema package for C, *8-4*
XmlSchemaSetErrorHandler in package Schema package for C, *8-4*
XmlSchemaSetValidateOptions in package Schema package for C, *8-5*
XmlSchemaTargetNamespace in package Schema package for C, *8-6*
XmlSchemaUnload in package Schema package for C, *8-6*
XmlSchemaValidate in package Schema package for C, *8-7*
XmlSchemaVersion in package Schema package for C, *8-8*
XmlSoapAddBodyElement in package SOAP package for C, *9-2*
XmlSoapAddFaultReason in package SOAP package for C, *9-3*

XmlSoapAddFaultSubDetail in package SOAP package for C, *9-3*
XmlSoapAddHeaderElement in package SOAP package for C, *9-4*
XmlSoapCall in package SOAP package for C, *9-5*
XmlSoapCreateConnection in package SOAP package for C, *9-5*
XmlSoapCreateCtx in package SOAP package for C, *9-7*
XmlSoapCreateMsg in package SOAP package for C, *9-8*
XmlSoapDestroyConnection in package SOAP package for C, *9-8*
XmlSoapDestroyCtx in package SOAP package for C, *9-9*
XmlSoapDestroyMsg in package SOAP package for C, *9-9*
XmlSoapError in package SOAP package for C, *9-10*
XmlSoapGetBody in package SOAP package for C, *9-10*
XmlSoapGetBodyElement in package SOAP package for C, *9-11*
XmlSoapGetEnvelope in package SOAP package for C, *9-12*
XmlSoapGetFault in package SOAP package for C, *9-12*
XmlSoapGetHeader in package SOAP package for C, *9-13*
XmlSoapGetHeaderElement in package SOAP package for C, *9-14*
XmlSoapGetMustUnderstand in package SOAP package for C, *9-14*
XmlSoapGetReasonLang in package SOAP package for C, *9-15*
XmlSoapGetReasonNum in package SOAP package for C, *9-16*
XmlSoapGetRelay in package SOAP package for C, *9-16*
XmlSoapGetRole in package SOAP package for C, *9-17*
XmlSoapHasFault in package SOAP package for C, *9-17*
XmlSoapSetFault in package SOAP package for C, *9-18*
XmlSoapSetMustUnderstand in package SOAP package for C, *9-19*
XmlSoapSetRelay in package SOAP package for C, *9-20*
XmlSoapSetRole in package SOAP package for C, *9-20*
XmlVersion in package XML package for C, *11-14*
XmlXPathCreateCtx in package XPath package for C, *13-1*
XmlXPathDestroyCtx in package XPath package for C, *13-2*
XmlXPathEval in package XPath package for C, *13-2*
XmlXPathGetObjectBoolean in package XPath package for C, *13-3*
XmlXPathGetObjectFragment in package XPath package for C, *13-3*
XmlXPathGetObjectNSetNode in package XPath package for C, *13-4*
XmlXPathGetObjectNSetNum in package XPath package for C, *13-4*
XmlXPathGetObjectNumber in package XPath package for C, *13-5*
XmlXPathGetObjectString in package XPath package for C, *13-5*
XmlXPathGetObjectType in package XPath package for C, *13-6*
XmlXPathParse in package XPath package for C, *13-6*
XmlXPointerEval in package XPointer package for C, *14-1*
XmlXPtrLocGetNode in package XPointer package for C, *14-2*
XmlXPtrLocGetPoint in package XPointer package for C, *14-2*
XmlXPtrLocGetRange in package XPointer package for C, *14-2*
XmlXPtrLocGetType in package XPointer package for C, *14-3*
XmlXPtrLocSetFree in package XPointer package for C, *14-4*
XmlXPtrLocSetGetItem in package XPointer package for C, *14-4*
XmlXPtrLocSetGetLength in package XPointer package for C, *14-5*
XmlXPtrLocToString in package XPointer package for C, *14-3*
XmlXslCreate in package XSLT package for C, *15-1*
XmlXslDestroy in package XSLT package for C, *15-2*
XmlXslGetBaseURI in package XSLT package for C, *15-2*
XmlXslGetOutput in package XSLT package for C, *15-3*
XmlXslGetStylesheetDom in package XSLT package for C, *15-3*
XmlXslGetTextParam in package XSLT package for C, *15-3*
XmlXslProcess in package XSLT package for C, *15-4*

XmlXslResetAllParams in package XSLT package for C, *15-4*
XmlXslSetOutputDom in package XSLT package for C, *15-5*
XmlXslSetOutputEncoding in package XSLT package for C, *15-5*
XmlXslSetOutputMethod in package XSLT package for C, *15-5*
XmlXslSetOutputSax in package XSLT package for C, *15-6*
XmlXslSetOutputStream in package XSLT package for C, *15-6*
XmlXslSetTextParam in package XSLT package for C, *15-7*
XMLXVM_DEBUG_F in package XSLTVM package for C, *16-7*
XmlXvmCompileBuffer in package XSLTVM package for C, *16-1*
XmlXvmCompileDom in package XSLTVM package for C, *16-1*
XmlXvmCompileFile in package XSLTVM package for C, *16-2*
XmlXvmCompileURI in package XSLTVM package for C, *16-3*
XmlXvmCompileXPath in package XSLTVM package for C, *16-4*
XmlXvmCreate in package XSLTVM package for C, *16-7*
XmlXvmCreateComp in package XSLTVM package for C, *16-4*
XmlXvmDestroy in package XSLTVM package for C, *16-8*
XmlXvmDestroyComp in package XSLTVM package for C, *16-5*
XmlXvmEvaluateXPath in package XSLTVM package for C, *16-8*
XmlXvmGetBytecodeLength in package XSLTVM package for C, *16-5*
XmlXvmGetObjectBoolean in package XSLTVM package for C, *16-9*
XmlXvmGetObjectNSetNode in package XSLTVM package for C, *16-10*
XmlXvmGetObjectNSetNum in package XSLTVM package for C, *16-10*
XmlXvmGetObjectNumber in package XSLTVM package for C, *16-11*
XmlXvmGetObjectString in package XSLTVM package for C, *16-11*
XmlXvmGetObjectType in package XSLTVM package for C, *16-12*
XmlXvmGetOutputDom in package XSLTVM package for C, *16-12*
XmlXvmResetParams in package XSLTVM package for C, *16-13*
XmlXvmSetBaseURI in package XSLTVM package for C, *16-13*
XmlXvmSetBytecodeBuffer in package XSLTVM package for C, *16-13*
XmlXvmSetBytecodeFile in package XSLTVM package for C, *16-14*
XmlXvmSetBytecodeURI in package XSLTVM package for C, *16-14*
XmlXvmSetDebugFunc in package XSLTVM package for C, *16-15*
XmlXvmSetOutputDom in package XSLTVM package for C, *16-15*
XmlXvmSetOutputEncoding in package XSLTVM package for C, *16-16*
XmlXvmSetOutputSax in package XSLTVM package for C, *16-16*
XmlXvmSetOutputStream in package XSLTVM package for C, *16-17*
XmlXvmSetTextParam in package XSLTVM package for C, *16-17*
XmlXvmTransformBuffer in package XSLTVM package for C, *16-18*
XmlXvmTransformDom in package XSLTVM package for C, *16-18*
XmlXvmTransformFile in package XSLTVM package for C, *16-19*
XmlXvmTransformURI in package XSLTVM package for C, *16-20*
XPath package for C, *13-1*
XPointer package for C, *14-1*
XSLT package for C, *15-1*
XSLTVM package for C, *16-1*