Oracle®

XML API リファレンス

10*g* リリース 1（10.1）

部品番号 : B12491-01

2004 年 2 月

ORACLE®

Oracle XML API リファレンス , 10*g* リリース 1（10.1）

部品番号 : B12491-01

原本名 : Oracle XML API Reference, 10*g* Release 1 (10.1)

原本部品番号 : B10789-01

原本協力者 : Stanley Guan, Dmitry Lenkov, Roza Leyderman, Ian Macky, Anguel Novoselsky, Tomas Saulys, Mark Scardina

目次

4 C 用の DOM API パッケージ

5 C 用の範囲 API パッケージ

6 C 用の SAX API パッケージ

9 C 用の XML API パッケージ

10 C 用の XPath API パッケージ

13 C 用の XSLTVM API パッケージ

16 C++ 用の IO API パッケージ

17 C++ 用の OracleXml API パッケージ

18 C++ 用のパーサー API パッケージ

19 C++ 用のツール API パッケージ

20 C++ 用の XPath API パッケージ

21 C++ 用の XPointer API パッケージ

22 C++ 用の Xsl API パッケージ

表

はじめに

このマニュアルでは、Oracle XML Developer's Kit（XDK）および Oracle XML DB Application Program Interface（API）について説明します。主に、これらの API に対応するファンクション、メソッドおよびプロシージャの構文を示します。

この章では次の内容を扱います。

対象読者

このマニュアルは Oracle の XML アプリケーションを構築する開発者を対象としています。

このマニュアルを使用する場合、オブジェクト指向のプログラミング概念の基本的な理解、Structured Query Language（SQL）の知識、および C または C++ を使用したアプリケーション開発の操作知識が必要です。

このマニュアルの構成

このマニュアルは、C 言語と C++ 言語のそれぞれの API 向けに 2 つのパートに分かれています。

第 I 部「C 用の XML API」

第 I 部では、XML に関連した C API について説明します。このリリースでは、クライアント側とサーバー側の両方のアプリケーションで開発用インタフェースに一貫性を持たせるように、新しい統一 API グループが開発されています。以前の C API は、旧仕様への対応の便宜上、このリリースでも使用できますが、使用しないことをお薦めします。これらの統一前の C API は、次回のリリースから使用できなくなることにご注意ください。

このマニュアルに記載されている統一 C API の内容は、次のとおりです。

- 第 1 章「古い C API から新しい C API へのマッピング」
- 第 2 章「C 用のデータ型」
- 第 3 章「C 用のコールバック API パッケージ」
- 第 4 章「C 用の DOM API パッケージ」
- 第 5 章「C 用の範囲 API パッケージ」
- 第 6 章「C 用の SAX API パッケージ」
- 第 7 章「C 用のスキーマ API パッケージ」
- 第 8 章「C 用の横断 API パッケージ」
- 第 9 章「C 用の XML API パッケージ」
- 第 10 章「C 用の XPath API パッケージ」
- 第 11 章「C 用の XPointer API パッケージ」
- 第 12 章「C 用の XSLT API パッケージ」
- 第 13 章「C 用の XSLTVM API パッケージ」

第 II 部「C++ 用の XML API」

第 II 部では、XML に関連した C++ API について説明します。

- 第 14 章「C++ 用の Ctx API パッケージ」
- 第 15 章「C++ 用の DOM API パッケージ」
- 第 16 章「C++ 用の IO API パッケージ」
- 第 17 章「C++ 用の OracleXml API パッケージ」
- 第 18 章「C++ 用のパーサー API パッケージ」
- 第 19 章「C++ 用のツール API パッケージ」
- 第 20 章「C++ 用の XPath API パッケージ」
- 第 21 章「C++ 用の XPointer API パッケージ」
- 第 22 章「C++ 用の Xsl API パッケージ」

関連ドキュメント

詳細は、次の Oracle マニュアルを参照してください。

- 『Oracle Database 概要』
- 『Oracle Database SQL リファレンス』
- 『Oracle Database アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』
- 『Oracle Database 新機能』
- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

このマニュアルの多くの例で、Oracle のインストール時にデフォルトとしてインストールされるシード・データベースのサンプル・スキーマを使用しています。スキーマの作成および使用方法の詳細は、『Oracle Database サンプル・スキーマ』を参照してください。

リリース・ノート、インストール関連ドキュメント、ホワイト・ペーパーまたはその他の関連ドキュメントは、OTN-J（Oracle Technology Network Japan）から、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。登録は、次の Web サイトから無償で行えます。

```
http://otn.oracle.co.jp/membership/
```

すでに OTN-J のユーザー名およびパスワードを取得している場合は、次の URL で OTN-J Web サイトのドキュメントのセクションに直接接続できます。

```
http://otn.oracle.co.jp/document/
```

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

本文の表記規則

本文では、特定の項目が一目でわかるように、次の表記規則を使用します。次の表に、その規則と使用例を示します。

規則	意味	例
太字	太字は、本文中で定義されている用語および用語集に記載されている用語を示します。	この句を指定すると、**索引構成表**が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システム指定の要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドがあります。また、システム指定の列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみ、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを作成できます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが指定する要素のサンプルを示します。このような要素には、コンピュータ名およびデータベース名、ネット・サービス名および接続識別子があります。また、ユーザーが指定するデータベース・オブジェクトとデータベース構造、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニットおよびパラメータ値も含まれます。 **注意 :** プログラム要素には、大文字と小文字を組み合わせて使用するものもあります。これらの要素は、記載されているとおり入力してください。	sqlplus と入力して、SQL*Plus をオープンします。 パスワードは、orapwd ファイルで指定します。 /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを作成します。 hr.departments 表には、department_id、department_name および location_id 列があります。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーとして接続します。 JRepUtil クラスが次のメソッドを実装します。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。ここで、*old_release* とはアップグレード前にインストールしたリリースを示します。

コード例の表記規則

コード例は、SQL、PL/SQL、SQL*Plus または他のコマンドライン文の例です。次のように固定幅フォントで表示され、通常のテキストと区別されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例で使用される表記規則とその使用例を示します。

規則	意味	例
[ ]	大カッコは、カッコ内の項目を任意に選択することを表します。大カッコは入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコは、カッコ内の項目のうち、1 つが必須であることを表します。中カッコは入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の複数の選択項目の区切りに使用します。項目のうちの 1 つを入力します。縦線は入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平の省略記号は、次のいずれかを示します。	
	■ 例に直接関連しないコードの一部が省略されている。	CREATE TABLE ... AS *subquery*;
	■ コードの一部を繰り返すことができる。	SELECT *col1*, *col2*, ... , *coln* FROM employees;

.
.
	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	//process information in buffer
		.
		.
		blob.close();

その他の記号	大カッコ、中カッコ、縦線および省略記号以外の記号は、記載されているとおりに入力する必要があります。	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
イタリック体	イタリック体は、特定の値を指定する必要のあるプレースホルダまたは変数を示します。	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*
大文字	大文字は、システム指定の要素を示します。これらの要素は、ユーザー定義の要素と区別するために大文字で示されます。大カッコ内にないかぎり、表示されているとおりの順序および綴りで入力します。ただし、大 / 小文字が区別されないため、小文字でも入力できます。	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;

規則	意味	例
小文字	小文字は、ユーザー指定のプログラム要素を示します。たとえば、表名、列名、ファイル名などです。	`SELECT last_name, employee_id FROM employees;` `sqlplus hr/hr`
	注意：プログラム要素には、大文字と小文字を組み合せて使用するものもあります。これらの要素は、記載されているとおり入力してください。	`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Oracle XML API リファレンスに記載されている新機能

この章では『Oracle XML API リファレンス』に記載されている新機能を説明し、追加情報の参照先を示します。

10*g* リリース 1（10.1）

このリリースの新機能は次のとおりです。

- C プログラミング言語用の新しい API。第 I 部「C 用の XML API」を参照してください。
- C++ プログラミング言語用の新しい API。第 II 部「C++ 用の XML API」を参照してください。
- PL/SQL 用のすべての API は、このリリースの『PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』に記載されています。
- SQL 用のすべての API は、このリリースの『Oracle Database SQL リファレンス』に記載されています。
- Java プログラミング言語用のすべての API は、このリリースの新しいマニュアル『Oracle XML Java API Reference』に記載されています。

第 I 部

C 用の XML API

第 I 部に含まれる章は、次のとおりです。

1

古い C API から新しい C API へのマッピング

この章では、Oracle9*i* リリースで使用可能な XML C API を、このリリースの Oracle Database で使用可能な統一 XML C API にマッピングする方法について説明します。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』のフォーマット・モデルに関する項

C パッケージの変更点

既存の C API は oraxml パッケージを介して使用できました。これには、次のような特性がありました。

- 仕様は XML コンテキスト（xmlctx）と XML 文書間の 1 対 1 のマッピングに制限されていました。DOM から一度にアクセスできるのは常に 1 つのドキュメントのみでした。ただし、複数のドキュメントのデータへは、同時にアクセスできました。
- API は常に一貫性があるとはかぎらず、xmlctx の宣言に準拠しない場合もありました。

これに対して、新しい統一 C API ではこれらの問題が解決されています。

- xmlctx は、複数の独立したドキュメント間で共有されます。
- すべての API は xmlctx の宣言に準拠しています。
- XmlDestroy() コールにより明示的に破棄されるまで、各ドキュメントには DOM から同時にアクセスできます。

初期化と解析順序の変更点

統一 C API ではドキュメントの初期化と解析が変更されました。

例 1-1 統一前の C API を使用した初期化と解析の順序 : 一度に 1 ドキュメント

次の擬似コードは、古い C API を使用してドキュメントを一度に 1 つずつ初期化し、解析する方法を示しています。例 1-2 の例と比較してください。

```
#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);
```

例 1-2 統一 C API を使用した初期化と解析の順序 : 一度に 1 ドキュメント

次の擬似コードは、新しい C API を使用してドキュメントを一度に 1 つずつ初期化し、解析する方法を示しています。例 1-1 の例と比較してください。

```
#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);
```

例 1-3 統一前の C API を使用した初期化と解析の順序 : 複数ドキュメントおよび同時 DOM アクセス

次の擬似コードは、古い C API を使用して同時 DOM アクセスにより複数のドキュメントを初期化し、解析する方法を示しています。例 1-4 の例と比較してください。

```
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);
```

例 1-4 統一 C API を使用した初期化と解析の順序 : 複数ドキュメントおよび同時 DOM アクセス

次の擬似コードは、新しい C API を使用して同時 DOM アクセスにより複数のドキュメントを初期化し、解析する方法を示しています。例 1-3 の例と比較してください。

```
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);
```

oraxml パッケージと xml パッケージ間のデータ型のマッピング

表 1-1 に新しい C API におけるデータ型の変更の概略を示します。

表 1-1 oraxml パッケージと xml パッケージでサポートされるデータ型

oraxml でサポートされるデータ型	xml でサポートされるデータ型
uword	xmlerr
xmlacctype	xmlurlacc
xmlattrnode	xmlattrnode
xmlcdatanode	xmlcdatanode
xmlcommentnode	xmlcommentnode
xmlctx	xmlctx
xmldocnode	xmldocnode
xmldomimp	廃止。xmlctx を使用
xmldtdnode	xmldtdnode
xmlelemnode	xmlelemnode
xmlentnode	xmlentnode
xmlentrefnode	xmlentrefnode
xmlflags	ub4
xmlfragnode	xmlfragnode
xmlihdl	xmlurlhdl
xmlmemcb	個々のファンクション・ポインタを使用
xmlnode	xmlnode
xmlnodes	xmlnodelist, xmlnamedmap
xmlnotenode	xmlnotenode
xmlntype	xmlnodetype
xmlpflags	ub4
xmlpinode	xmlpinode
xmlsaxcb	xmlsaxcb
xmlstream	xmlistream, xmliostream

表 1-1 oraxml パッケージと xml パッケージでサポートされるデータ型（続き）

oraxml でサポートされるデータ型	xml でサポートされるデータ型
xmltextnode	xmltextnode
xpctx	xpctx
xpexpr	xpexpr
xpnset	廃止。XmlXPathGetObjectNSetNum() と XmlXPathGetObjectNSetNode() を使用
xpnsetele	廃止。XmlXPathGetObjectNSetNum() と XmlXPathGetObjectNSetNode() を使用
xpobj	xpobj
xpobjtyp	xmlxslobjtype
xslctx	xslctx
xsloutputmethod	xmlxsloutputmethod

oraxml パッケージと xml パッケージのメソッドのマッピング

表 1-2 に新しい C API におけるメソッドの変更の概略を示します。

表 1-2 oraxml パッケージと xml パッケージのメソッド

パッケージ oraxml のメソッド	パッケージ xml のメソッド
appendChild()	XmlDomAppendChild()
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()
getAttributes()	XmlDomGetAttrs()
getAttrLocal()	XmlDomGetAttrLocal(), XmlDomGetAttrLocalLen()
getAttrName()	XmlDomGetAttrName()

表 1-2 oraxml パッケージと xml パッケージのメソッド（続き）

パッケージ oraxml のメソッド	パッケージ xml のメソッド
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()	廃止。ドキュメントは XmlLoadDom***xxx***() コールにより戻されます。
getDocumentElement()	XmlDomGetDoctElem()
getElementByID()	XmlDomGetElemByID()
getElementsByTagName()	XmlDomGetElemsByTag()
getElementsByTagNameNS()	XmlDomGetElemsByTag()
getEncoding()	XmlDomGetEncoding()
getEntityNotation()	XmlDomGetEntityNotation()
getEntityPubID()	XmlDomGetEntityPubID()
getEntitySysID()	XmlDomGetEntitySysID()
getFirstChild()	XmlDomGetFirstChild()
getImplementation()	廃止。DOMImplementation のかわりに xmlctx を使用
getLastChild()	XmlDomGetLastChild()
getNamedItem()	XmlDomGetNamedItem()
getNextSibling()	XmlDomGetNextSibling()

表 1-2 oraxml パッケージと xml パッケージのメソッド（続き）

パッケージ oraxml のメソッド	パッケージ xml のメソッド
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()
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()

表 1-2 oraxml パッケージと xml パッケージのメソッド（続き）

パッケージ oraxml のメソッド	パッケージ xml のメソッド
numChildNodes()	XmlDomNumChildNodes()
prefixToURI()	XmlDomPrefixToURI()
printBuffer()	XmlSaveDomBuffer()
printBufferEnc()	XmlSaveDomBuffer()
printCallback()	XmlSaveDomStream()
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()
substringData()	XmlDomSubstringData()
xmlaccess()	XmlAccess()

表 1-2 oraxml パッケージと xml パッケージのメソッド（続き）

パッケージ oraxml のメソッド	パッケージ xml のメソッド
xmlinit()	XmlCreate()
xmlinitenc()	XmlCreate()
xmlLocation()	TBD.
xmlparse()	XmlLoadDomURI()
xmlparsebuf()	XmlLoadDomBuffer()
xmlparsedtd()	廃止。XmlLoad*XXX*() で XML_LOAD_FLAG_DTD_ONLY フラグを使用
xmlparsefile()	XmlLoadDomFile()
xmlparsestream()	XmlLoadDomStream()
xmlterm()	XmlDestroy()
xmlwhere()	TBD
xpevalxpathexpr()	XmlXPathEval()
xpfreexpathctx()	XmlXPathDeleteCtx()
xpgetbooleanval()	XmlXPathGetObjectBoolean()
xpgetfirstnsetelem()	XmlXPathGetObjectNSetNum()
xpgetnextnsetelem()	XmlXPathGetObjectNSetNum()
xpgetnsetelemnode()	XmlXPathGetObjectNSetNum()
xpgetnsetval()	XmlXPathGetObjectNSetNum()
xpgetnumval()	XmlXPathGetObjectNumber()
xpgetrtfragval()	XmlXPathGetObjectFragment()
xpgetstrval()	XmlXPathGetObjectString()
xpgetxpobjtyp()	XmlXPathGetObjectType()
xpmakexpathctx()	XmlXPathCreateCtx()
xpparsexpathexpr()	XmlXPathParse()
xslgetbaseuri()	XmlXslGetBaseURI()
xslgetoutputdomctx()	XmlXslGetOutputDom()
xslgetoutputsax()	不要
xslgetoutputstream()	不要

表 1-2 oraxml パッケージと xml パッケージのメソッド（続き）

パッケージ oraxml のメソッド	パッケージ xml のメソッド
xslgetresultdocfrag()	XmlXslGetOutputFragment()
xslgettextparam()	XmlXslGetTextParam()
xslgetxslctx()	不要
xslinit()	XmlXslCreateCtx()
xslprocess()	XmlXslProcess()
xslprocessex()	XmlXslProcess()
xslprocessxml()	XmlXslProcess()
xslprocessxmldocfrag()	XmlXslProcess()
xslresetallparams()	XmlXslResetAllParams()
xslsetoutputdomctx()	XmlXslSetOutputDom()
xslsetoutputencoding()	XmlXslSetOutputEncoding()
xslsetoutputmethod()	XmlXslSetOutputMethod()
xslsetoutputsax()	XmlXslSetOutputSax()
xslsetoutputsaxctx()	XmlXslSetOutputSax()
xslsetoutputstream()	XmlXslSetOutputStream()
xslsettextparam()	XmlXslSetTextParam()
xslterm()	XmlXslDeleteCtx()

2

C 用のデータ型

このパッケージでは、XML コールバックのファンクション（またはファンクション・ポインタ）を宣言するマクロを定義します。コールバックはエラーメッセージ処理、メモリーの割当てと解放およびストリーム操作に使用されます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

C のデータ型

表 2-1 に、C のすべてのデータ型と、その説明を示しています。

表 2-1 C のデータ型の概要

データ型	目的
2-3 ページ「xmlcmphow」	DOM 範囲の比較に使用される定数。
2-3 ページ「xmlctx」	XML セッションのすべてのドキュメントで共有されるコンテキスト。
2-4 ページ「xmlerr」	多くのファンクションにより戻される数値のエラー・コード。
2-4 ページ「xmlistream」	ユーザー定義の汎用入力ストリーム。
2-5 ページ「xmliter」	DOM2 `NodeIterator` と `TreeWalker` の構造の制御。
2-5 ページ「xmlnodetype」	ノードの数値型コード。
2-6 ページ「xmlostream」	ユーザー定義の汎用出力ストリーム。
2-7 ページ「xmlpoint」	XPointer のポイント位置。
2-7 ページ「xmlrange」	DOM2 範囲の構造の制御。
2-7 ページ「xmlshowbits」	表示するノードのタイプの選択に使用されるビット・フラグ。
2-8 ページ「xmlurlacc」	URL からデータを取り出すための、既知のアクセス・メソッドの列挙。
2-8 ページ「xmlurlhdl」	この共有体には、URL データへのアクセスに必要なハンドル、すなわちストリームまたは `stdio` ポインタ、ファイル・ディスクリプタなどが含まれます。
2-9 ページ「xmlurlpart」	この構造には URL のサブパートが含まれます。
2-9 ページ「xmlxptrloc」	XPointer 位置のデータ型。
2-9 ページ「xmlxptrlocset」	XPointer 位置の設定データ型。
2-10 ページ「xmlxslobjtype」	戻される XSLT オブジェクトのタイプ。
2-10 ページ「xmlxslomethod」	XSLT プロセッサで生成される出力のタイプ。
2-10 ページ「xmlxvm」	`xmlxvm` 型のオブジェクトは XML ドキュメントの変換に使用されます。
2-11 ページ「xmlxvmcomp」	`xmlxvmcomp` 型のオブジェクトは XSL スタイルシートのコンパイルに使用されます。

表 2-1 C のデータ型の概要（続き）

データ型	目的
2-11 ページ「xmlxvmflags」	XSLT コンパイラの制御フラグ。
2-11 ページ「xmlxvmobjtype」	XSLTVM オブジェクトのタイプ。
2-12 ページ「xpctx」	XPath のトップレベルのコンテキスト。
2-12 ページ「xpexpr」	XPath 式。
2-12 ページ「xpobj」	XPath オブジェクト。
2-12 ページ「xsdctx」	XMLSchema バリデータのコンテキスト。
2-13 ページ「xslctx」	XSL の最上位コンテキスト。
2-13 ページ「xvmobj」	XSLVM プロセッサのランタイム・オブジェクト。コンテンツは内部的に使用されるため、ユーザーがアクセスすることはできません。

xmlcmphow

DOM 範囲の比較に使用される定数。

定義

```
typedef enum {
    XMLDOM_START_TO_START = 0,
    XMLDOM_START_TO_END   = 1,
    XMLDOM_END_TO_END     = 2,
    XMLDOM_END_TO_START   = 3
} xmlcmphow;
```

xmlctx

XML セッションのすべてのドキュメントで共有されるコンテキスト。エンコーディング情報、低レベルのメモリー割当てファンクション・ポインタ、エラー・メッセージの言語 / エンコーディングおよびオプションのハンドラ・ファンクションなどが含まれます。ドキュメントのロード（解析）、および DOM の作成、SAX の生成などに必要です。

定義

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

xmlerr

多くのファンクションにより戻される数値のエラー・コード。値が 0（ゼロ）の場合は成功を、0（ゼロ）以外の場合にはエラーを表します。

定義

```
typedef enum {
   XMLERR_OK              = 0, /* success return */
   XMLERR_NULL_PTR        = 1, /* NULL pointer */
   XMLERR_NO_MEMORY       = 2, /* out of memory */
   XMLERR_HASH_DUP        = 3, /* duplicate entry in hash table */
   XMLERR_INTERNAL        = 4,  /* internal error */
   XMLERR_BUFFER_OVERFLOW = 5,  /* name/quoted string too long */
   XMLERR_BAD_CHILD       = 6,  /* invalid child for parent */
   XMLERR_EOI             = 7,  /* unexpected EndOfInformation */
   XMLERR_BAD_MEMCB       = 8,  /* invalid memory callbacks */
   XMLERR_UNICODE_ALIGN   = 12, /* Unicode data misalignment */
   XMLERR_NODE_TYPE       = 13, /* wrong node type */
   XMLERR_UNCLEAN         = 14, /* context is not clean */
   XMLERR_NESTED_STRINGS  = 18, /* internal: nested open str */
   XMLERR_PROP_NOT_FOUND  = 19, /* property not found */
   XMLERR_SAVE_OVERFLOW   = 20, /* save output overflowed */
   XMLERR_NOT_IMP         = 21, /* feature not implemented */
   XMLERR_NLS_MISMATCH    = 50, /* specify lxglo/lxd or neither*/
   XMLERR_NLS_INIT        = 51, /* error at NLS initialization */
   XMLERR_LEH_INIT        = 52, /* error at LEH initialization */
   XMLERR_LML_INIT        = 53, /* error at LML initialization */
   XMLERR_LPU_INIT        = 54  /* error at LPU initialization */
} xmlerr;
```

xmlistream

ユーザー定義の汎用入力ストリーム。3 つのファンクション・ポインタが必要です（スタブでもかまいません）。コンテキスト・ポインタは完全なユーザー定義ポインタです。ストリームの管理に必要な状態の情報をすべて示します。このポインタはユーザー・ファンクションに最初の引数として渡されます。

定義

```
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

DOM 2 NodeIterator と TreeWalker の構造を制御します。

定義

```
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

ノードの数値型コード。0 は無効を意味し、1 ～ 13 は DOM1.0 の標準番号です。14 以上の数値は内部的に使用されます。

定義

```
typedef enum {
    XMLDOM_NONE     = 0,  /* bogus node */
```

```
    XMLDOM_ELEM     = 1,  /* element */
    XMLDOM_ATTR     = 2,  /* attribute */
    XMLDOM_TEXT     = 3,  /* char data not escaped by CDATA */
    XMLDOM_CDATA    = 4,  /* char data escaped by CDATA */
    XMLDOM_ENTREF   = 5,  /* entity reference */
    XMLDOM_ENTITY   = 6,  /* entity */
    XMLDOM_PI       = 7,  /* <?processing instructions?> */
    XMLDOM_COMMENT  = 8,  /* <!-- Comments --> */
    XMLDOM_DOC      = 9,  /* Document */
    XMLDOM_DTD      = 10, /* DTD */
    XMLDOM_FRAG     = 11, /* Document fragment */
    XMLDOM_NOTATION = 12, /* notation */

    /* Oracle extensions from here on */
    XMLDOM_ELEMDECL = 13, /* DTD element declaration */
    XMLDOM_ATTRDECL = 14, /* DTD attribute declaration */

    /* Content Particles (nodes in element's Content Model) */
    XMLDOM_CPELEM   = 15, /* element */
    XMLDOM_CPCHOICE = 16, /* choice (a|b) */
    XMLDOM_CPSEQ    = 17, /* sequence (a,b) */
    XMLDOM_CPPCDATA = 18, /* #PCDATA */
    XMLDOM_CPSTAR   = 19, /* '*' (zero or more) */
    XMLDOM_CPPLUS   = 20, /* '+' (one or more) */
    XMLDOM_CPOPT    = 21, /* '?' (optional) */
    XMLDOM_CPEND    = 22  /* end marker */
} xmlnodetype;
```

xmlostream

ユーザー定義の汎用出力ストリーム。3 つのファンクション・ポインタが必要です（スタブでもかまいません）。コンテキスト・ポインタは完全なユーザー定義ポインタです。ストリームの管理に必要な状態の情報をすべて示します。このポインタはユーザー・ファンクションに最初の引数として渡されます。

定義

```
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 のポイント位置。

定義

```
typedef struct xmlpoint xmlpoint;
```

xmlrange

DOM 2 範囲の構造を制御します。

定義

```
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;
```

xmlshowbits

表示するノードのタイプの選択に使用されるビット・フラグ。

定義

```
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

URL からデータを取り出すための、既知のアクセス・メソッドの列挙。オープン、読込み、およびクローズの各ファンクションを指定して、デフォルトの動作をオーバーライドすることができます。

定義

```
typedef enum {
    XML_ACCESS_NONE    = 0, /* not specified */
    XML_ACCESS_UNKNOWN = 1, /* specified but unknown */
    XML_ACCESS_FILE    = 2, /* filesystem access */
    XML_ACCESS_HTTP    = 3, /* HTTP */
    XML_ACCESS_FTP     = 4, /* FTP */
    XML_ACCESS_GOPHER  = 5, /* Gopher */
    XML_ACCESS_ORADB   = 6, /* Oracle DB */
    XML_ACCESS_STREAM  = 7  /* user-defined stream */
} xmlurlacc;
```

xmlurlhdl

この共有体には、URL データへのアクセスに必要なハンドル、すなわちストリームまたは stdio ポインタ、ファイル・ディスクリプタなどが含まれます。

定義

```
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

この構造には URL のサブパートが含まれます。この元の URL は解析され、NULL で終了する各パートのコピーは作業バッファに格納され、その後この構造はパートを示すように埋められます。URL を http://user:pwd@baz.com:8080/pub/baz.html;quux=1?huh#fraggy とした場合、URL のコンポーネント・パートの例が表示されます。

定義

```
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 位置のデータ型。

定義

```
typedef struct xmlxptrloc xmlxptrloc;
```

xmlxptrlocset

XPointer 位置の設定データ型。

定義

```
typedef struct xmlxptrlocset xmlxptrlocset;
```

xmlxslobjtype

戻される XSLT オブジェクトのタイプ。

定義

```
typedef enum xmlxslobjtype {
    XMLXSL_TYPE_UNKNOWN = 0, /* Not a defined type */
    XMLXSL_TYPE_NDSET   = 1, /* Node-set */
    XMLXSL_TYPE_BOOL    = 2, /* Boolean value */
    XMLXSL_TYPE_NUM     = 3, /* Numeric value (double) */
    XMLXSL_TYPE_STR     = 4, /* String */
    XMLXSL_TYPE_FRAG    = 5  /* Document Fragment */
} xmlxslobjtype;
```

xmlxslomethod

XSLT プロセッサで生成される出力のタイプ。

定義

```
typedef enum xmlxslomethod {
 XMLXSL_OUTPUT_UNKNOWN = 0, /* Not defined */
 XMLXSL_OUTPUT_XML     = 1, /* Produce a Document Fragment */
 XMLXSL_OUTPUT_STREAM  = 2, /* Stream out formatted result */
 XMLXSL_OUTPUT_HTML    = 3  /* Stream out HTML formatted result */
} xmlxslomethod;
```

xmlxvm

xmlxvm 型のオブジェクトは XML ドキュメントの変換に使用されます。xmlxvm のコンテンツは内部的に使用されるため、ユーザーからはアクセスできません。

定義

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

xmlxvmcomp

xmlxvmcomp 型のオブジェクトは XSL スタイルシートのコンパイルに使用されます。xmlxvmcomp のコンテンツは内部的に使用されるため、ユーザーからはアクセスできません。

定義

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

xmlxvmflags

XSLT コンパイラの制御フラグ。

- XMLXVM_DEBUG はコンパイラに対して、バイトコードへのデバッグ情報の挿入を強制します。
- XMLXVM_STRIPSPACE は xsl:strip-space elements="*" と同じ動作を強制します。

定義

```
typedef ub4 xmlxvmflag;
#define XMLXVM_NOFLAG     0x00
#define XMLXVM_DEBUG      0x01 /* insert debug info into bytecode */
#define XMLXVM_STRIPSPACE 0x02 /* same as xsl:strip-space elements="*" */
```

xmlxvmobjtype

XSLTVM オブジェクトのタイプ。

定義

```
typedef enum xmlxvmobjtype {
    XMLXVM_TYPE_UNKNOWN = 0,
    XMLXVM_TYPE_NDSET   = 1,
    XMLXVM_TYPE_BOOL    = 2,
    XMLXVM_TYPE_NUM     = 3,
    XMLXVM_TYPE_STR     = 4,
    XMLXVM_TYPE_FRAG    = 5
} xmlxvmobjtype;
```

xpctx

XPath の最上位コンテキスト。

定義

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

xpexpr

XPath 式。

定義

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

xpobj

XPath オブジェクト。

定義

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

xsdctx

XML スキーマ・バリデータ・コンテキスト。`XmlSchemaCreate` により作成され、ほとんどのスキーマ・ファンクションに渡されます。

定義

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

xslctx

XSL の最上位コンテキスト。

定義

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

xvmobj

XSLVM プロセッサのランタイム・オブジェクト。コンテンツは内部的に使用されるため、ユーザーがアクセスすることはできません。

定義

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

3

C 用のコールバック API パッケージ

このパッケージでは、XML コールバックのファンクション（またはファンクション・ポインタ）を宣言するマクロを定義します。コールバックはエラー・メッセージ処理、メモリーの割当てと解放およびストリーム操作に使用されます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

コールバック・メソッド

表 3-1 にコールバック・インタフェースで使用できるメソッドの概要を示します。

表 3-1 コールバック・メソッドの概要

ファンクション	概要
3-3 ページ「XML_ACCESS_CLOSE_F」	ユーザー定義のアクセス・メソッド・クローズ・コールバック。
3-3 ページ「XML_ACCESS_OPEN_F」	ユーザー定義のアクセス・メソッド・オープン・コールバック。
3-4 ページ「XML_ACCESS_READ_F」	ユーザー定義のアクセス・メソッド読込みコールバック。
3-5 ページ「XML_ALLOC_F」	低レベルのメモリー割当て。
3-6 ページ「XML_ERRMSG_F」	エラー・メッセージの処理。
3-6 ページ「XML_FREE_F」	低レベルのメモリー解放。
3-7 ページ「XML_STREAM_CLOSE_F」	ユーザー定義のストリーム・クローズ・コールバック。
3-8 ページ「XML_STREAM_OPEN_F」	ユーザー定義のストリーム・オープン・コールバック。
3-9 ページ「XML_STREAM_READ_F」	ユーザー定義のストリーム読込みコールバック。
3-10 ページ「XML_STREAM_WRITE_F」	ユーザー定義のストリーム書込みコールバック。

XML_ACCESS_CLOSE_F

このマクロは URL へのアクセスに使用する、クローズ・ファンクション・コールバックのプロトタイプを定義します。

構文

```
#define XML_ACCESS_CLOSE_F(func, ctx, uh)
xmlerr func(
   void *ctx,
    xmlurlhdl *uh)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザー定義のコンテキスト
uh	IN	URL ハンドル

戻り値

(xmlerr) 数値のエラー・コード、成功時には 0 を戻します。

関連項目： XML_ACCESS_OPEN_F、XML_ACCESS_READ_F

XML_ACCESS_OPEN_F

このマクロは URL へのアクセスに使用する、オープン・ファンクション・コールバックのプロトタイプを定義します。

構文

```
#define XML_ACCESS_OPEN_F(func, ctx, uri, parts, length, uh)
xmlerr func(
   void *ctx,
   oratext *uri,
   xmlurlpart *parts,
   ubig_ora *length,
   xmlurlhdl *uh)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザー定義のコンテキスト
uri	IN	オープンされる URI
parts	IN	コンポーネントに分割された URI
length	OUT	入力データの長さがわかっている場合は合計の長さ、わからない場合は 0
uh	IN	URL ハンドル

戻り値

(xmlerr) 数値のエラー・コード、成功時には 0 を戻します。

関連項目： XML_ACCESS_CLOSE_F、XML_ACCESS_READ_F

XML_ACCESS_READ_F

このマクロは URL へのアクセスに使用する、読込みファンクション・コールバックのプロトタイプを定義します。

構文

```
#define XML_ACCESS_READ_F(func, ctx, uh, data, nraw, eoi)
xmlerr func(
   void *ctx,
   xmlurlhdl *uh,
   oratext **data,
   ubig_ora *nraw,
   ub1 *eoi)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザー定義のコンテキスト
uh	IN	URL ハンドル
data	IN/OUT	受信者のデータ・バッファ。データの先頭にリセット
nraw	OUT	読み込まれる実際のデータ・バイト数
eoi	OUT	End of Information（EOI）へのシグナル、最後のチャンク

戻り値

(xmlerr) 数値のエラー・コード、成功時には 0 を戻します。

関連項目： XML_ACCESS_OPEN_F、XML_ACCESS_CLOSE_F

XML_ALLOC_F

このマクロはユーザーが指定する低レベル・メモリー alloc ファンクションのプロトタイプを定義します。アロケータが指定されない場合、malloc が使用されます。このファンクションにより、メモリーがゼロにならないようにしてください。XML_FREE_F に対応します。

構文

```
#define XML_ALLOC_F(func, mctx, size)
void *func(
   void *mctx,
   size_t size)
```

パラメータ	IN/OUT	説明
mctx	IN	低レベルのメモリー・コンテキスト
size	IN	割り当てられるバイト数

戻り値

(void *) 割り当てられたメモリー数。

関連項目： XML_FREE_F

XML_ERRMSG_F

このマクロはエラー・メッセージ処理ファンクションのプロトタイプを定義します。XML 初期化の際にエラー・メッセージ・コールバックが割り当てられない場合、エラーは stderr に出力されます。ハンドラが指定されている場合、stderr への出力のかわりにハンドラが呼び出されます。

構文

```
#define XML_ERRMSG_F(func, ectx, msg, err)
void func(
   void *ectx,
   oratext *msg,
   xmlerr err)
```

パラメータ	IN/OUT	説明
ectx	IN	エラー・メッセージ・コンテキスト
msg	IN	エラー・メッセージのテキスト
err	IN	数値のエラー・コード

関連項目： 第 9 章「C 用の XML API パッケージ」の XmlCreate

XML_FREE_F

このマクロはユーザーが指定する低レベル・メモリー解放ファンクションのプロトタイプを定義します。アロケータが指定されない場合、free() が使用されます。XML_ALLOC_F に対応します。

構文

```
#define XML_FREE_F(func, mctx, ptr)
void func(
   void *mctx,
   void *ptr)
```

パラメータ	IN/OUT	説明
mctx	IN	低レベルのメモリー・コンテキスト
ptr	IN	解放されるメモリー

XML_STREAM_CLOSE_F

このマクロはオープン・ソースをクローズし、そのリソースを解放するためにコールされるクローズ・ファンクション・コールバックのプロトタイプを定義します。

構文

```
#define XML_STREAM_CLOSE_F(func, xctx, sctx)
void func(
   xmlctx *xctx,
   void *sctx)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
sctx	IN	ユーザー定義のストリーム・コンテキスト

関連項目： XML_STREAM_OPEN_F、XML_STREAM_READ_F、XML_STREAM_WRITE_F

XML_STREAM_OPEN_F

このマクロはオープン・ファンクション・コールバックのプロトタイプを定義します。このファンクション・コールバックは入力ソースをオープンするために 1 回コールされます。このファンクションは、正常に実行されると XMLERR_OK を戻します。

構文

```
#define XML_STREAM_OPEN_F(func, xctx, sctx, path, parts, length)
xmlerr func(
   xmlctx *xctx,
   void *sctx,
   oratext *path,
   void *parts,
   ubig_ora *length)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
sctx	IN	ユーザー定義のストリーム・コンテキスト
path	IN	オープンする URI へのフルパス
parts	IN	コンポーネントに分割された URI（不透明ポインタ）
length	(OUT)	入力データの長さがわかっている場合は合計の長さ、わからない場合は 0

戻り値

(xmlerr) 数値のエラー・コード、成功時には 0 を戻します。

関連項目： XML_STREAM_CLOSE_F、XML_STREAM_READ_F、XML_STREAM_WRITE_F

XML_STREAM_READ_F

このマクロは読込みファンクション・コールバックのプロトタイプを定義します。このファンクション・コールバックはオープン・ソースのデータをバッファに読み込むためにコールされ、読み込まれるバイト数を戻します（エラー時は < 0 を戻します）。これがデータの最終ブロックかどうかは、eoi フラグにより判断されます。

EOI では、クローズ・ファンクションが自動的にコールされます。

構文

```
#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)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
sctx	IN	ユーザー定義のストリーム・コンテキスト
path	IN	（エラー・メッセージの）オープン・ソースのフル URI
dest	(OUT)	データの読込み先の宛先バッファ
size	IN	宛先バッファのサイズ。
nraw	(OUT)	読み込まれるバイト数。
eoi	(OUT)	End of Information（EOI）へのシグナル、最後のチャンク

戻り値

(xmlerr) 数値のエラー・コード、成功時には 0 を戻します。

関連項目： XML_STREAM_OPEN_F、XML_STREAM_CLOSE_F、XML_STREAM_WRITE_F

XML_STREAM_WRITE_F

このマクロはユーザー定義のストリームにデータを書き込むためにコールされる、書込みファンクション・コールバックのプロトタイプを定義します。

構文

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

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
sctx	IN	ユーザー定義のストリーム・コンテキスト
path	IN	（エラー・メッセージの）オープン・ソースのフル URI
src	IN	データを読み込むソース・バッファ
size	IN	ソースのサイズ（バイト）

戻り値

(xmlerr) 数値のエラー・コード、成功時には 0 を戻します。

関連項目： XML_STREAM_OPEN_F、XML_STREAM_CLOSE_F、XML_STREAM_READ_F

4

C 用の DOM API パッケージ

この実装は、REC-DOM-Level-1-19981001 に準拠しています。DOM 標準はオブジェクト指向であるため、C 言語用にいくつかの変更が行われました。

- 再利用されるファンクション名を拡張する必要があります。`Attr` インタフェースの `getValue` には、DOM 2 の `getNodeValue` によって確立されるパターンと一致する一意の名前 `XmlDomGetAttrValue` があります。
- 標準を超えて DOM を拡張するために、ファンクションが追加されました。その一例が、子ノードの数を戻す `XmlDomNumChildNodes` です。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

Attr インタフェース

表 4-1 に、Attr インタフェースを介して使用できるメソッドを示します。

表 4-1 Attr メソッドの概要 : DOM パッケージ

ファンクション	概要
4-4 ページ「XmlDomGetAttrLocal」	属性の名前空間のローカル名を、NULL で終了する文字列として戻します。
4-4 ページ「XmlDomGetAttrLocalLen」	属性の名前空間のローカル名を、長さがエンコードされた文字列として戻します。
4-5 ページ「XmlDomGetAttrName」	属性の名前を、NULL で終了する文字列として戻します。
4-6 ページ「XmlDomGetAttrNameLen」	属性の名前を、長さがエンコードされた文字列として戻します。
4-7 ページ「XmlDomGetAttrPrefix」	属性の名前空間の接頭辞を戻します。
4-8 ページ「XmlDomGetAttrSpecified」	属性が明示的に作成されたかどうかを示すフラグを戻します。
4-8 ページ「XmlDomGetAttrURI」	属性の名前空間 URI を、NULL で終了する文字列として戻します。
4-9 ページ「XmlDomGetAttrURILen」	属性の名前空間 URI を、長さがエンコードされた文字列として戻します。
4-10 ページ「XmlDomGetAttrValue」	属性の値を、NULL で終了する文字列として戻します。
4-11 ページ「XmlDomGetAttrValueLen」	属性の値を、長さがエンコードされた文字列として戻します。
4-12 ページ「XmlDomGetAttrValueStream」	属性値のストリーム形式、つまりチャンクを取得します。
4-12 ページ「XmlDomGetOwnerElem」	属性の元の要素を戻します。
4-13 ページ「XmlDomSetAttrValue」	属性の値を設定します。
4-13 ページ「XmlDomSetAttrValueStream」	属性値のストリーム形式（チャンク）を設定します。

XmlDomGetAttrLocal

（データ・エンコーディングの）属性の名前空間のローカル名を戻します。属性名が完全に修飾されていない（接頭辞がない）場合、ローカル名は属性名と同じになります。

長さがエンコードされたバージョンは、XmlDomGetAttrURILen として使用できます。これは、ローカル名をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(oratext *) 属性のローカル名（データ・エンコーディング)。

関連項目： XmlDomGetAttrLocalLen、XmlDomGetAttrName、XmlDomGetAttrURI、XmlDomGetAttrPrefix

XmlDomGetAttrLocalLen

（データ・エンコーディングの）属性の名前空間のローカル名を戻します。属性名が完全に修飾されていない（接頭辞がない）場合、ローカル名は属性名と同じになります。

NULL で終了するバージョンは、XmlDomGetAttrLocal として使用できます。これは、ローカル名を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、属性のデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく GetXXX ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0（ゼロ）以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。
buf	IN	入力バッファ（オプション）。
buflen	IN	入力バッファ長（オプション）。
len	OUT	ローカル名の長さ（文字数）。

戻り値

(oratext *) Attr のローカル名（データ・エンコーディング）。

関連項目： XmlDomGetAttrLocal、XmlDomGetAttrName、XmlDomGetAttrURI、XmlDomGetAttrPrefix

XmlDomGetAttrName

(データ・コーディングの）属性の完全修飾名を、NULL で終了する文字列として戻します。たとえば、bar¥0、foo:bar¥0 などです。

長さがエンコードされたバージョンは、XmlDomGetAttrNameLen として使用できます。これは、属性名をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(oratext *) 属性の名前（データ・エンコーディング）。

関連項目： XmlDomGetAttrNameLen、XmlDomGetAttrURI、XmlDomGetAttrPrefix、XmlDomGetAttrLocal

XmlDomGetAttrNameLen

（データ・コーディングの）属性の完全修飾名を、長さがエンコードされた文字列として戻します。たとえば、("bar", 3)、("foo:bar", 7) などです。

NULL で終了するバージョンは、XmlDomGetAttrName として使用できます。これは、属性名を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、属性のデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Get*XXX* ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0（ゼロ）以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

パラメータ	In/Out	説明
buf	IN	入力バッファ（オプション）。
buflen	IN	入力バッファ長（オプション）。
len	OUT	ローカル名の長さ（文字数）。

戻り値

(oratext *) 属性の名前（データ・エンコーディング）。

関連項目： XmlDomGetAttrName、XmlDomGetAttrURI、XmlDomGetAttrPrefix、XmlDomGetAttrLocal

XmlDomGetAttrPrefix

（データ・エンコーディングの）属性の名前空間の接頭辞を戻します。属性名が完全に修飾されていない（接頭辞がない）場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(oratext *) 属性の名前空間の接頭辞（データ・エンコーディング）または NULL。

関連項目： XmlDomGetAttrName、XmlDomGetAttrURI、XmlDomGetAttrLocal

XmlDomGetAttrSpecified

属性の指定されたフラグを戻します。元のドキュメントでこの属性に値が明示的に指定されている場合は TRUE、それ以外の場合は FALSE を戻します。ノードが属性ではない場合は、FALSE を戻します。ユーザーが DOM を介して属性の値を設定した場合、specified フラグは TRUE になります。属性をデフォルト値（ある場合）に戻すには、属性を削除する必要があります。これにより、デフォルト値を使用して属性が自動的に再作成されます（specified は FALSE になります）。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(boolean) 属性の specified フラグ。

関連項目： XmlDomSetAttrValue

XmlDomGetAttrURI

（データ・エンコーディングの）属性の名前空間 URI を戻します。属性名が修飾されていない（名前空間の接頭辞が含まれない）場合は、ノードが作成されたときに、デフォルトの名前空間が使用されます（NULL になる可能性があります）。

長さがエンコードされたバージョンは、XmlDomGetAttrURILen として使用できます。これは、URI をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(oratext *) 属性の名前空間 URI（データ・エンコーディング）または NULL。

関連項目： XmlDomGetAttrURILen、XmlDomGetAttrPrefix、XmlDomGetAttrLocal

XmlDomGetAttrURILen

(データ・エンコーディングの）属性の名前空間 URI を、長さがエンコードされた文字列として戻します。属性名が修飾されていない（名前空間の接頭辞が含まれない）場合は、ノードが作成されたときに、デフォルトの名前空間が使用されます（NULL になる可能性があります)。

NULL で終了するバージョンは、XmlDomGetAttrURI として使用できます。これは、URI を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、属性のデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Get ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0（ゼロ）以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。
buf	IN	入力バッファ（オプション）。
buflen	IN	入力バッファ長（オプション）。
len	OUT	URI の長さ（文字数）。

戻り値

(oratext *) 属性の名前空間 URI（データ・エンコーディング）または NULL。

関連項目： XmlDomGetAttrURI、XmlDomGetAttrPrefix、XmlDomGetAttrLocal

XmlDomGetAttrValue

（データ・エンコーディングの）属性の値（文字データ）を、NULL で終了する文字列として戻します。文字および汎用エンティティが置換されています。

長さがエンコードされたバージョンは、XmlDomGetAttrValueLen として使用できます。これは、属性値をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(oratext *) 属性の値。

関連項目： XmlDomGetAttrValueLen、XmlDomSetAttrValue

XmlDomGetAttrValueLen

(データ・エンコーディングの) 属性の値 (文字データ) を、長さがエンコードされた文字列として戻します。文字および汎用エンティティが置換されています。

NULL で終了するバージョンは、XmlDomGetAttrValue として使用できます。これは、属性値を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、属性のデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Get*XXX* ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0 (ゼロ) 以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。
buf	IN	入力バッファ (オプション)。
buflen	IN	入力バッファ長 (オプション)。
len	OUT	属性値の長さ (文字数)。

戻り値

(oratext *) 属性の値。

関連項目： XmlDomGetAttrValue、XmlDomSetAttrValue

XmlDomGetAttrValueStream

属性の大きい値（対応する文字データ）を戻し、分割してユーザーの出力ストリームに送信します。非常に大きいデータの場合、単一の連続するチャンクとして（効率的に）格納できるとはかぎりません。このファンクションは、そのタイプのチャンク・データにアクセスするために使用されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。
ostream	IN	出力ストリーム・オブジェクト。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

XmlDomGetOwnerElem

属性に対応する Element ノードを戻します。各 attr は要素（唯一の要素）に属するか、または分離されており DOM ツリーの一部ではありません。前者の場合は要素ノードが戻されます。attr が未割当の場合は NULL が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。

戻り値

(xmlelemnode *) 属性の要素ノード（または NULL）。

関連項目： XmlDomGetOwnerDocument

XmlDomSetAttrValue

指定された属性の値をデータに設定します。ノードが属性ではない場合、処理は不要です。新しい値はデータ・エンコーディング内にある必要があります。新しい値は、検証、変換またはチェックされません。新しい値の設定後、属性の specified フラグは TRUE になります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。
value	IN	新しい属性の値（データ・エンコーディング）。

関連項目： XmlDomGetAttrValue

XmlDomSetAttrValueStream

属性の大きい値（対応する文字データ）を、入力ストリームとは別に設定します。非常に大きいデータの場合、単一の連続するチャンクとして効率的に格納できるとはかぎりません。このファンクションは、そのタイプのチャンク・データへのアクセスに使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
attr	IN	属性ノード。
isream	IN	入力ストリーム。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

CharacterData インタフェース

表 4-2 に、CharacterData インタフェースを介して使用できるメソッドを示します。

表 4-2 CharacterData メソッドの概要 : DOM パッケージ

ファンクション	概要
4-15 ページ「XmlDomAppendData」	ノードの現行データの最後にデータを追加します。
4-16 ページ「XmlDomDeleteData」	ノードのデータの一部を削除します。
4-17 ページ「XmlDomGetCharData」	ノードのデータを戻します。
4-18 ページ「XmlDomGetCharDataLength」	ノードのデータの長さを戻します。
4-18 ページ「XmlDomInsertData」	ノードの現行データに文字列を挿入します。
4-19 ページ「XmlDomReplaceData」	ノードのデータの一部を置換します。
4-20 ページ「XmlDomSetCharData」	ノードのデータを設定します。
4-21 ページ「XmlDomSubstringData」	ノードのデータの部分文字列を戻します。

XmlDomAppendData

CharacterData ノードのデータの最後に文字列を追加します。ノードが Text、Comment または CDATA 以外の場合、または追加する文字列が NULL の場合、処理を行いません。追加されるデータは、データ・エンコーディング内にある必要があります。追加されるデータは、検証、変換またはチェックされません。

新しいノード・データは DOM によって割当ておよび管理されますが、前のノード値がユーザーによって割当ておよび管理されていた場合、新しいノード・データが前のノード値を解放します。これにより、前のノード値が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。
data	IN	追加するデータ（データ・エンコーディング）。
old	OUT	ノードの以前のデータ（データ・エンコーディング）。

関連項目： XmlDomGetCharData、XmlDomInsertData、XmlDomDeleteData、XmlDomReplaceData、XmlDomSplitText

XmlDomDeleteData

CharacterData ノードのデータから文字の範囲を削除します。ノードがテキスト、コメントまたは CDATA ではない場合、または offset が元のデータの範囲外にある場合は、処理を行いません。offset は 0（ゼロ）から始まるため、offset が 0（ゼロ）ということは、データの開始を意味します。offset および count は、バイト数ではなく文字数です。offset および count の合計がデータ長を超える場合、offset からデータの最後の文字までがすべて削除されます。

新しいノード・データは DOM によって割当ておよび管理されますが、前のノード値がユーザーによって割当ておよび管理されていた場合、新しいノード・データが前のノード値を解放します。これにより、前のノード値が戻されます。

構文

```
void XmlDomDeleteData(
   xmlctx *xctx,
   xmlnode *node,
   ub4 offset,
   ub4 count,
   oratext **old)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。
offset	IN	削除を開始する文字のオフセット。

パラメータ	In/Out	説明
count	IN	削除する文字数。
old	OUT	ノードの前のデータ（データ・エンコーディング）。

関連項目： XmlDomGetCharData、XmlDomAppendData、XmlDomInsertData、XmlDomReplaceData、XmlDomSplitText

XmlDomGetCharData

データ・エンコーディングの CharacterData ノードのデータ（テキスト、コメントまたは CDATA の型）を戻します。ノードのタイプがそれ以外の場合、またはデータがない場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。Text、Comment または CDATA。

戻り値

(oratext *) ノードの文字データ（データ・エンコーディング）。

関連項目： XmlDomSetCharData、XmlDomCreateText、XmlDomCreateComment、XmlDomCreateCDATA

XmlDomGetCharDataLength

CharacterData ノードのデータ長およびタイプ（Text、Comment または CDATA）を、バイト数ではなく文字数で戻します。ノードのタイプがそれ以外の場合は、0（ゼロ）を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。Text、Comment または CDATA。

戻り値

(ub4) ノードのデータの長さ（バイト数ではなく文字数)。

関連項目： XmlDomGetCharData

XmlDomInsertData

指定された位置で CharacterData ノードのデータに文字列を挿入します。ノードが Text、Comment または CDATA ではない場合や、挿入される文字列が NULL の場合、または offset が元のデータの範囲外にある場合は、処理を行いません。挿入されるデータは、データ・エンコーディング内にある必要があります。挿入されるデータは、検証、変換またはチェックされません。offset は、バイトではなく文字として指定されます。offset は 0（ゼロ）から始まるため、offset に 0（ゼロ）を挿入すると、データが付加されます。

新しいノード・データは DOM によって割当ておよび管理されますが、前のノード値がユーザーによって割当ておよび管理されていた場合、新しいノード・データが前のノード値を解放します。これにより、前のノード値が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。Text、Comment または CDATA。
offset	IN	挿入を開始する文字のオフセット。
arg	IN	挿入するデータ。
old	OUT	ノードの前のデータ（データ・エンコーディング）。

関連項目： XmlDomGetCharData、XmlDomAppendData、XmlDomDeleteData、XmlDomReplaceData、XmlDomSplitText

XmlDomReplaceData

CharacterData ノードのデータの文字の範囲を、新しい文字列に置換します。ノードがテキスト、コメントまたは CDATA ではない場合や、offset が元のデータの範囲外にある場合、または置換文字列が NULL の場合は、処理を行いません。count が 0（ゼロ）の場合は、XmlDomInsertData と同じように機能します。offset は 0（ゼロ）から始まるため、offset が 0（ゼロ）ということは、データの開始を意味します。置換データは、データ・エンコーディング内にある必要があります。置換されるデータは、検証、変換またはチェックされません。offset および count は、バイト数ではなく文字数です。offset および count の合計がデータ長を超える場合、データの最後までの文字がすべて置換されます。

新しいノード・データは DOM によって割当ておよび管理されますが、前のノード値がユーザーによって割当ておよび管理されていた場合、新しいノード・データが前のノード値を解放します。これにより、前のノード値が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。Text、Comment または CDATA。
offset	IN	置換を開始する文字のオフセット。
count	IN	置換する文字数。
arg	IN	置換する部分文字列（データ・エンコーディング）。
old	OUT	ノードの前のデータ（データ・エンコーディング）。

関連項目： XmlDomGetCharData、XmlDomAppendData、XmlDomInsertData、XmlDomDeleteData、XmlDomSplitText

XmlDomSetCharData

データ・エンコーディングの CharacterData ノードのデータ（テキスト、コメントまたは CDATA の型）を設定します。ノードのタイプがそれ以外の場合は、処理を行いません。新しいデータは、検証、変換またはチェックされず、データ・エンコーディング内にある必要があります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。Text、Comment または CDATA。
data	IN	ノードの新しいデータ。

関連項目： XmlDomGetCharData

XmlDomSubstringData

CharacterData ノードの文字データの範囲および型（Text、Comment または CDATA）を戻します。ノードのタイプがそれ以外の場合、または count が 0（ゼロ）の場合は、NULL を戻します。データはデータ・エンコーディング内にあるため、offset および count は、バイト数ではなく文字数です。文字列の先頭は、offset 0 です。offset および count の合計がデータ長を超える場合、データの最後までの文字がすべて戻されます。

部分文字列は、ノードのドキュメントのメモリー・プール内に永続的に割り当てられます。部分文字列を解放するには、XmlDomFreeString を使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	CharacterData ノード。Text、Comment または CDATA。
offset	IN	部分文字列の抽出を開始する文字のオフセット。
count	IN	抽出する文字数。

戻り値

(oratext *) 指定された部分文字列。

関連項目： XmlDomAppendData、XmlDomInsertData、XmlDomDeleteData、XmlDomReplaceData、XmlDomSplitText、XmlDomFreeString

Document インタフェース

表 4-3 に、Document インタフェースを介して使用できるメソッドを示します。

表 4-3 Document メソッドの概要 : DOM パッケージ

ファンクション	概要
4-24 ページ「XmlDomCreateAttr」	属性ノードを作成します。
4-25 ページ「XmlDomCreateAttrNS」	名前空間の情報を含む属性ノードを作成します。
4-26 ページ「XmlDomCreateCDATA」	CDATA ノードを作成します。
4-27 ページ「XmlDomCreateComment」	コメント・ノードを作成します。
4-28 ページ「XmlDomCreateElem」	要素ノードを作成します。
4-29 ページ「XmlDomCreateElemNS」	名前空間の情報を含む要素ノードを作成します。
4-30 ページ「XmlDomCreateEntityRef」	実体参照ノードを作成します。
4-31 ページ「XmlDomCreateFragment」	文書フラグメントを作成します。
4-31 ページ「XmlDomCreatePI」	PI ノードを作成します。
4-32 ページ「XmlDomCreateText」	テキスト・ノードを作成します。
4-33 ページ「XmlDomFreeString」	XmlDomSubstringData などによって割り当てられた文字列を解放します。
4-34 ページ「XmlDomGetBaseURI」	ドキュメントのベース URI を戻します。
4-35 ページ「XmlDomGetDTD」	ドキュメントの DTD を取得します。
4-35 ページ「XmlDomGetDecl」	ドキュメントの XMLDecl 情報を戻します。
4-36 ページ「XmlDomGetDocElem」	ドキュメントの最上位要素を取得します。
4-37 ページ「XmlDomGetDocElemByID」	ドキュメント要素に指定された ID を取得します。
4-38 ページ「XmlDomGetDocElemsByTag」	ドキュメント要素を取得します。
4-39 ページ「XmlDomGetDocElemsByTagNS」	ドキュメント要素（名前空間認識バージョン）を取得します。
4-40 ページ「XmlDomGetLastError」	ドキュメントの最後のエラー・コードを戻します。
4-40 ページ「XmlDomGetSchema」	ドキュメントに対応するスキーマの URI を戻します。

表 4-3 Document メソッドの概要 : DOM パッケージ（続き）

ファンクション	概要
4-41 ページ「XmlDomImportNode」	別の DOM からノードをインポートします。
4-42 ページ「XmlDomIsSchemaBased」	スキーマがドキュメントに対応するかどうかを示します。
4-43 ページ「XmlDomSaveString」	ドキュメントのメモリー・プールに文字列を永続的に保存します。
4-44 ページ「XmlDomSaveString2」	ドキュメントのメモリー・プールに Unicode 文字列を永続的に保存します。
4-45 ページ「XmlDomSetBaseURI」	ドキュメントのベース URL を設定します。
4-45 ページ「XmlDomSetDTD」	ドキュメントの DTD を設定します。
4-46 ページ「XmlDomSetDocOrder」	すべてのノードのドキュメント順序を設定します。
4-47 ページ「XmlDomSetLastError」	ドキュメントの最後のエラー・コードを設定します。
4-47 ページ「XmlDomSync」	ドキュメントの永続バージョンと DOM を同期化します。

XmlDomCreateAttr

（データ・エンコーディングで）指定された名前と値を使用して、属性ノードを作成します。このファンクションは、属性の初期値を設定できない DOM 仕様とは異なることに注意してください（`XmlDomSetAttrValue` を参照）。名前は必須ですが、値は `NULL` の可能性があり、どちらも検証、変換またはチェックされません。

これは、名前空間を認識しないファンクションです（`XmlDomCreateAttrNS` を参照）。新しい属性には、`NULL` の名前空間 URI および接頭辞が設定され、指定された属性名が修飾名の場合でも、属性のローカル名は属性名と同じになります。

初期値が指定された場合、属性の specified フラグは `TRUE` になります。

新しいノードは、親がなく孤立しているため、`XmlDomAppendChild` などを使用して、DOM ツリーに追加する必要があります。

1 回の操作で属性を作成および追加する「`XmlDomSetAttr`」を参照してください。

名前および値はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
name	IN	新しいノードの名前（データ・エンコーディング）。ユーザーが制御します。
value	IN	新しいノードの値（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlattrnode *) 新しい Attr ノード。

関連項目： XmlDomSetAttrValue、XmlDomCreateAttrNS、XmlDomSetAttr、XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateAttrNS

指定された名前空間 URI および修飾名を使用して、属性ノードを作成します。これは、XmlDomCreateAttr の名前空間を認識したバージョンです。このファンクションは、属性の初期値を設定できない DOM 仕様とは異なることに注意してください（XmlDomSetAttrValue を参照）。名前は必須ですが、値は NULL の可能性があり、どちらも検証、変換またはチェックされません。

初期値が指定された場合、属性の specified フラグは TRUE になります。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。1 回の操作で属性を作成および追加する「XmlDomSetAttr」を参照してください。

URI、修飾名および値はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

```
xmlattrnode* XmlDomCreateAttrNS(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *uri,
   oratext *qname,
   oratext *value)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
uri	IN	ノードの名前空間 URI（データ・エンコーディング）。ユーザーが制御します。
qname	IN	ノードの修飾名（データ・エンコーディング）。ユーザーが制御します。
value	IN	新しいノードの値（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlattrnode *) 新しい Attr ノード。

関連項目： XmlDomSetAttrValue、XmlDomCreateAttr、XmlDomSetAttr、XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateCDATA

指定された初期データ（データ・エンコーディング内にある必要があります）を使用して、CDATASection ノードを作成します。CDATASection は逐語的とみなされ、解析されることはありません。正規化処理によって、隣接する Text ノードと結合されることもありません。初期データは NULL の可能性があります。初期データを指定しなかった場合、検証、変換またはチェックは行われません。CDATA ノードの名前は常に #cdata-section です。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

CDATA はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
data	IN	新しいノードの CDATA（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlcdatanode *) 新しい CDATA ノード。

関連項目： XmlDomCreateText、XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateComment

指定された初期データ（データ・エンコーディング内にある必要があります）を使用して、Comment ノードを作成します。データは NULL の可能性があります。初期データを指定しなかった場合、検証、変換またはチェックは行われません。Comment ノードの名前は常に #comment です。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

コメント・データはコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
data	IN	新しいノードのコメント（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlcommentnode *) 新しい Comment ノード。

関連項目： XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateElem

指定されたタグ名（データ・エンコーディング内にある必要があります）を使用して、要素ノードを作成します。要素のタグ名は、大 / 小文字を区別します。これは、名前空間を認識しないファンクションです。新しいノードには、NULL の名前空間 URI および接頭辞が設定され、指定されたタグ名が修飾名の場合でも、ノードのローカル名はタグ名と同じになります。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

tagname はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
tagname	IN	新しいノードの名前（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlelemnode *) 新しい Element ノード。

関連項目： XmlDomCreateElemNS、XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateElemNS

指定された名前空間 URI および修飾名を使用して、要素を作成します。要素名は、大 / 小文字を区別します。URI は NULL の可能性がありますが、修飾名は必須です。修飾名は接頭辞とローカル部分に分割され、XmlDomGetNodePrefix、XmlDomGetNodeLocal などを使用して取得できます。tagName は完全修飾名になります。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

URI および修飾名はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
uri	IN	新しいノードの名前空間 URI（データ・エンコーディング）。ユーザーが制御します。
qname	IN	新しいノードの修飾名（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlelemnode *) 新しい Element ノード。

関連項目： XmlDomCreateElem、XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateEntityRef

EntityReference ノードを作成します。名前（データ・エンコーディング内にある必要があります）は、参照されるエンティティの名前です。名前付きエンティティは必要ありません。名前は、検証、変換またはチェックされません。

EntityReference ノードはパーサーによって生成されることはなく、実体参照が検出されたときに拡張されます。出力時に、実体参照ノードは "&name;" というスタイル参照に変わります。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

実体参照名はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

```
xmlentrefnode* XmlDomCreateEntityRef(
   xmlctx *xctx,
   xmldocnode *doc,
   oratext *name)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
name	IN	実体参照名（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlentrefnode *) 新しい EntityReference ノード。

XmlDomCreateFragment

空の DocumentFragment ノードを作成します。ドキュメント・フラグメントは、DOM ツリーに挿入されたときに特別とみなされます。フラグメント・ノード自体ではなく、フラグメントの子が順に挿入されます。挿入後、フラグメント・ノードは存続しますが、子はなくなります。「XmlDomInsertBefore」、「XmlDomReplaceChild」、「XmlDomAppendChild」などを参照してください。フラグメント・ノードの名前は常に "#document-fragment" です。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(xmlfragnode *) 新しい空の DocumentFragment ノード。

関連項目： XmlDomInsertBefore、XmlDomReplaceChild、XmlDomAppendChild

XmlDomCreatePI

指定されたターゲットおよびデータ（データ・エンコーディング内にある必要があります）を使用して、ProcessingInstruction ノードを作成します。データは、最初は NULL で、（XmlDomSetPIData を使用して）後で変更される可能性がありますが、ターゲットは必須で、変更できません。ターゲットは、検証、変換またはチェックされません。PI ノードの名前は、ターゲットと同じです。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

PI のターゲットおよびデータはコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
target	IN	新しいノードのターゲット（データ・エンコーディング）。ユーザーが制御します。
data	IN	新しいノードのデータ（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlpinode *) 新しい PI ノード。

関連項目： XmlDomGetPITarget、XmlDomGetPIData、XmlDomSetPIData、XmlDomCleanNode、XmlDomFreeNode

XmlDomCreateText

指定された初期データ（NULL 以外で、データ・エンコーディング内にある必要があります）を使用して、Text ノードを作成します。データは NULL の可能性があります。データを指定しなかった場合、検証、変換、チェックまたは解析は行われません（エンティティは拡張されません）。フラグメント・ノードの名前は常に "#text" です。Text ノードの新しいデータは、XmlDomSetNodeValue を使用して設定できます。メソッドの編集については、「CharacterData インタフェース」を参照してください。

新しいノードは、親がなく孤立しているため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。

テキスト・データはコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
data	IN	新しいノードのテキスト（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmltextnode *) 新しい Text ノード。

関連項目： XmlDomCreateCDATA、XmlDomSetNodeValue、XmlDomGetNodeValue、XmlDomSetCharData、XmlDomGetCharData、XmlDomGetCharDataLength、XmlDomSubstringData、XmlDomAppendData、XmlDomInsertData、XmlDomDeleteData、XmlDomReplaceData、XmlDomCleanNode、XmlDomFreeNode

XmlDomFreeString

XmlDomSubstringData または同様のファンクションによって割り当てられた文字列を解放します。XmlDomSaveString を使用して明示的に保存された文字列は、個別に解放できません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	文字列が属するドキュメント。
str	IN	解放する文字列。

関連項目： XmlDomSaveString、XmlDomSaveString2

XmlDomGetBaseURI

ドキュメントのベース URI を戻します。通常は、URI からロードされたドキュメントにのみ、ベース URI が自動的に設定されます。他のソース（stdin、バッファなど）からロードされたドキュメントには通常、ベース URI は設定されていませんが、相対 URI を解決するために、XmlDomSetBaseURI を使用してベース URI が設定されている場合もあります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(oratext *) ドキュメントのベース URI（または NULL）。

関連項目： XmlDomSetBaseURI

XmlDomGetDTD

現在のドキュメントに対応する DTD ノードを戻します。DTD がない場合は、NULL を戻します。DTD は編集できませんが、他のノードのタイプについては、XmlDomGetChildNodes を使用して子を取得できます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(xmldtdnode *) ドキュメントの DTD ノード（または NULL）。

関連項目： XmlDomSetDTD、第 9 章「C 用の XML API パッケージ」の XmlCreateDTD、第 9 章「C 用の XML API パッケージ」の XmlCreateDocument、XmlDomGetDTDName、XmlDomGetDTDEntities および XmlDomGetDTDNotations

XmlDomGetDecl

ドキュメントの XMLDecl から情報を戻します。XMLDecl が存在しない場合は、XMLERR_NO_DECL を戻します。XML バージョン（"1.0" または "2.0"）、指定されたエンコーディングおよびスタンドアロン値が戻されます。エンコーディングが指定されていない場合は、NULL が設定されます。スタンドアロン・フラグには 3 つの状態があります。スタンドアロンが指定されていない場合は < 0、スタンドアロンが指定されていて FALSE の場合は 0、スタンドアロンが指定されていて TRUE の場合は > 0 になります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
ver	OUT	XML バージョン。
enc	OUT	エンコーディング仕様。
std	OUT	スタンドアロン仕様。

戻り値

(xmlerr) XML エラー・コード。バージョン / エンコーディング / スタンドアロンが設定されます。

XmlDomGetDocElem

DOM ツリーのルート要素（ノード）を戻します。ルート要素がない場合は NULL を戻します。各ドキュメントには、ルート要素と呼ばれる最上位の Element ノードが 1 つのみあります。この最上位ノードは、ドキュメントが正常に解析された後に作成されるか、または XmlDomCreateElem、XmlDomAppendChild などによって手動で作成されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(xmlelemnode *) ルート要素（または NULL）。

関連項目： XmlDomCreateElem

XmlDomGetDocElemByID

指定された ID を持つ要素ノードを戻します。このような ID が定義されていない場合は、NULL を戻します。名前が "ID" である属性が、自動的に ID 型になるわけではありません。ID 属性（任意の名前を付けられます）は、DTD で ID 型として宣言する必要があります。

指定された ID は、データ・エンコーディング内にあるか、一致しない可能性があります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
id	IN	要素の一意の ID（データ・エンコーディング）。

戻り値

(xmlelemnode *) 一致する要素。

関連項目： XmlDomGetDocElemsByTag、XmlDomGetDocElemsByTagNS

XmlDomGetDocElemsByTag

指定されたタグ名を持つルート・ノードをルートとするドキュメント・ツリー内のすべての要素のリストを、ドキュメント順（ツリーの先行順走査で検出した順）に戻します。ルートが NULL の場合は、ドキュメント全体を検索します。

特殊名「*」を指定すると、すべてのタグ名に一致します。NULL 名を指定すると、何も一致しません。タグ名は大 / 小文字を区別するため、データ・エンコーディング内にある必要があります。データ・エンコーディング内にない場合は、不一致になる可能性があります。

このファンクションは、名前空間を認識しません。完全なタグ名が比較されます。同じ URI にマップされる異なる接頭辞を持つ 2 つの修飾名を比較すると、比較は失敗します。名前空間を認識するバージョンについては、「XmlDomGetElemsByTagNS」を参照してください。

リストが不要になった場合は、XmlDomFreeNodeList を使用して解放する必要があります。

リストは LIVE ではなく、スナップショットです。つまり、リストが戻された後に、タグ名に一致する新しいノードが DOM に追加された場合、そのノードを含むように、リストが自動的に更新されることはありません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
name	IN	一致させるタグ名（データ・エンコーディング）。すべてのタグ名を一致させるには「*」を指定します。

戻り値

(xmlnodelist *) 一致したすべての Element を含む、新しい NodeList。

関連項目： XmlDomGetDocElemByID、XmlDomGetDocElemsByTagNS、XmlDomFreeNodeList

XmlDomGetDocElemsByTagNS

指定された名前空間 URI およびローカル名を持つ（指定されたノードをルートとするドキュメント・ツリー内の）すべての要素のリストを、ツリーの先行順走査で検出した順に戻します。ルートが NULL の場合は、ドキュメント全体を検索します。

URI およびローカル名は、データ・エンコーディング内にある必要があります。特殊名「*」を指定すると、すべてのローカル名に一致します。NULL ローカル名を指定すると、何も一致しません。ただし、名前空間 URI は常に一致する必要があります。ワイルドカードは使用できません。比較では大 / 小文字が区別されます。名前空間を認識しないバージョンについては、「XmlDomGetDocElemsByTag」を参照してください。

リストが必要なくなったときは、XmlDomFreeNodeList を使用して解放する必要があります。

リストは最新ではなく、スナップショットです。つまり、リストが戻された後に、タグ名に一致する新しいノードが DOM に追加された場合、そのノードを含むように、リストが自動的に更新されることはありません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
uri	IN	一致させる名前空間 URI（データ・エンコーディング）。すべての名前空間 URI を一致させるには「*」を指定します。
local	IN	一致させるローカル名（データ・エンコーディング）。すべてのローカル名を一致させるには「*」を指定します。

戻り値

(xmlnodelist *) 一致したすべての Element を含む、新しい NodeList。

関連項目： XmlDomGetDocElemByID、XmlDomGetDocElemsByTag、XmlDomFreeNodeList

XmlDomGetLastError

指定されたドキュメントで最後に発生したエラーのエラー・コードを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(xmlerr) 数値のエラー・コード。エラーがない場合は、0（ゼロ）を戻します。

XmlDomGetSchema

ドキュメントに対応するスキーマの URI を戻します。URI がない場合は、NULL を戻します。XmlLoadDom ファンクションは、スキーマの場所のヒント（URI）を取得します。スキーマは、XMLType データを効率的にレイアウトするために使用されます。スキーマがロード時に指定された場合、このファンクションは TRUE を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(oratext *) スキーマ URI または NULL。

関連項目： 第 9 章「C 用の XML API パッケージ」の XmlDomIsSchemaBased、XmlLoadDom

XmlDomImportNode

Document から別のドキュメントにノードをインポートします。新しいノードは、孤立していて親がないため、XmlDomAppendChild などを使用して、DOM ツリーに追加する必要があります。元のノードを変更したり、ドキュメントから削除することはできませんが、元のノードの修飾名、接頭辞、名前空間 URI およびローカル名のすべてのコピーを使用して、新しいノードが作成されます。

XmlDomCloneNode と同様に、ノードの子を再帰的にインポートするかどうかを deep が制御します。FALSE の場合、ノードのみがインポートされ、ノードが子を持つことはありません。TRUE の場合、ノードのすべての子孫も同様にインポートされ、新しいサブツリー全体が作成されます。

Document ノードおよび DocumentType ノードは、インポートできません。インポートされた属性の specified フラグは、TRUE に設定されます。要素には、指定された属性のみがインポートされます。指定されていない（デフォルト）属性は省略されます。その後、新しいデフォルト属性（宛先ドキュメントの場合）が追加されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
nctx	IN	インポートされたノードの XML コンテキスト。
node	IN	インポートするノード。
deep	IN	サブツリーを再帰的にインポートする場合は TRUE。

戻り値

(xmlnode *) 新しくインポートされたノード（この Document 内）。

関連項目： XmlDomCloneNode

XmlDomIsSchemaBased

このドキュメントに対応するスキーマがあるかどうかを指定するフラグを戻します。XmlLoadDom ファンクションは、スキーマの場所のヒント（URI）を取得します。スキーマは、XMLType データを効率的にレイアウトするために使用されます。スキーマがロード時に指定された場合、このファンクションは TRUE を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(boolean) ドキュメントに対応するスキーマがある場合は TRUE。

関連項目： 第 9 章「C 用の XML API パッケージ」の XmlDomGetSchema、XmlLoadDom

XmlDomSaveString

指定された文字列をドキュメントのメモリー・プールにコピーすることで、ドキュメントが存続する間、その文字列も存続するようにします。個別の文字列は解放できません。ドキュメント全体が解放されたときにのみ、記憶域が戻されます。シングルバイトまたはマルチバイト・エンコーディングで機能します。Unicode 文字列の場合は、XmlDomSaveString2 を使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
str	IN	保存する文字列（データ・エンコーディング）。シングルバイトまたはマルチバイトのみです。

戻り値

(oratext *) 文字列の保存されたコピー。

関連項目： 第 9 章「C 用の XML API パッケージ」の XmlDomSaveString2、XmlFreeDocument

XmlDomSaveString2

指定された文字列をドキュメントのメモリー・プールにコピーすることで、ドキュメントが存続する間、その文字列も存続するようにします。個別の文字列は解放できません。ドキュメント全体が解放されたときにのみ、記憶域が戻されます。Unicode 文字列でのみ機能します。シングルバイトまたはマルチバイト文字列の場合は、XmlDomSaveString を使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
ustr	IN	保存する文字列（データ・エンコーディング）。Unicode のみです。

戻り値

(ub2 *) 文字列の保存されたコピー。

関連項目： 第 9 章「C 用の XML API パッケージ」の XmlDomSaveString、XmlFreeDocument

XmlDomSetBaseURI

URI からロードされたドキュメントのみに、ベース URI が自動的に設定されます。他のソース（stdin、バッファなど）からロードされたドキュメントには、ベース URI は自動的には設定されません。したがって、相対 URI を解決するために、この API を使用してベース URI を設定します。ベース URI は、データ・エンコーディング内にある必要があります。コピーが作成されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
uri	IN	設定されるベース URI（データ・エンコーディング）。

戻り値

(xmlerr) XML エラー・コード。

関連項目： XmlDomGetBaseURI

XmlDomSetDTD

ドキュメントの DTD を設定します。このコールは、解析が行われる前に、空のドキュメントに対してのみ使用されます。1 つの DTD を複数のドキュメントに対して設定できるため、DTD が設定されたドキュメントを解放しても、設定された DTD は解放されません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
dtdnode	IN	設定される DocumentType ノード

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

関連項目： XmlDomGetDTD、XmlDomGetDTDName、XmlDomGetDTDEntities、XmlDomGetDTDNotations

XmlDomSetDocOrder

現行のドキュメントの各ノードに対してドキュメント順序を設定します。XSLT 処理を行うには、最終ドキュメントに対してこのファンクションをコールする必要があります。このファンクションは、XSLT プロセッサによって自動的にコールされるため、ユーザーがこのファンクションをコールする必要はありません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
start_id	IN	文字列の ID 番号。

戻り値

(ub4) 割り当てられた最大の序数。

XmlDomSetLastError

指定されたドキュメントの最後のエラー・コードを設定します。doc が NULL の場合は、XML コンテキストのエラー・コードを設定します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。
errcode	IN	設定されるエラー・コード。エラーを消去するには 0 を設定します。

戻り値

(xmlerr) 元のエラー・コード。

XmlDomSync

変更された DOM を元のソースにすべて書き込んで、永続的なストアおよびメモリー内バージョンを同期化します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
doc	IN	XML 文書ノード。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

DocumentType インタフェース

表 4-4 に、DocumentType インタフェースを介して使用できるメソッドを示します。

表 4-4 DocumentType メソッドの概要 : DOM パッケージ

ファンクション	概要
4-48 ページ「XmlDomGetDTDEntities」	DTD のエンティティを取得します。
4-49 ページ「XmlDomGetDTDInternalSubset」	DTD の内部サブセットを取得します。
4-49 ページ「XmlDomGetDTDName」	DTD 名を取得します。
4-50 ページ「XmlDomGetDTDNotations」	DTD の表記法を取得します。
4-51 ページ「XmlDomGetDTDPubID」	DTD の公開識別子を取得します。
4-51 ページ「XmlDomGetDTDSysID」	DTD のシステム識別子を取得します。

XmlDomGetDTDEntities

DTD に対して定義されている汎用エンティティの名前付きノード・マップを戻します。ノードが DTD ではないか、または汎用エンティティを持っていない場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
dtd	IN	DTD ノード。

戻り値

(xmlnamedmap *) DTD で宣言されたエンティティを含む、名前付きノード・マップ。

関連項目： XmlDomGetDTD、XmlDomGetDTDName、XmlDomGetDTDNotations、XmlDomGetDTDSysID、XmlDomGetDTDInternalSubset

XmlDomGetDTDInternalSubset

要素のコンテンツ・モデルを戻します。DTD が存在しない場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
dtd	IN	DTD ノード。
name	IN	Element の名前（データ・エンコーディング）。

戻り値

(xmlnode *) コンテンツ・モデル・サブツリー。

関連項目： XmlDomGetDTD、XmlDomGetDTDName、XmlDomGetDTDEntities、XmlDomGetDTDNotations、XmlDomGetDTDPubID

XmlDomGetDTDName

DTD 名（DOCTYPE キーワードの直後に指定されます）、またはノードのタイプが DTD 以外の場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
dtd	IN	DTD ノード。

戻り値

(oratext *) DTD 名。

関連項目： XmlDomGetDTD、XmlDomGetDTDEntities、XmlDomGetDTDNotations、XmlDomGetDTDSysID、XmlDomGetDTDInternalSubset

XmlDomGetDTDNotations

DTD で宣言されている表記法の名前付きノード・マップを戻します。ノードが DTD ではないか、または Notation を持っていない場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
dtd	IN	DTD ノード。

戻り値

(xmlnamedmap *) DTD で宣言された表記法を含む、名前付きノード・マップ。

関連項目： XmlDomGetDTD、XmlDomGetDTDName、XmlDomGetDTDEntities、XmlDomGetDTDSysID、XmlDomGetDTDInternalSubset

XmlDomGetDTDPubID

DTD の公開識別子を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
dtd	IN	DTD ノード。

戻り値

(oratext *) DTD の公開識別子（データ・エンコーディング）。

関連項目： XmlDomGetDTD、XmlDomGetDTDName、XmlDomGetDTDEntities、XmlDomGetDTDSysID、XmlDomGetDTDInternalSubset

XmlDomGetDTDSysID

DTD のシステム識別子を取得します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
dtd	IN	DTD ノード。

戻り値

(oratext *) DTD のシステム識別子（データ・エンコーディング）。

関連項目： XmlDomGetDTD、XmlDomGetDTDName、XmlDomGetDTDEntities、XmlDomGetDTDPubID、XmlDomGetDTDInternalSubset

Element インタフェース

表 4-5 に、Element インタフェースを介して使用できるメソッドを示します。

表 4-5 Element メソッドの概要 : DOM パッケージ

ファンクション	概要
4-54 ページ「XmlDomGetAttr」	名前が指定された属性の値を戻します。
4-54 ページ「XmlDomGetAttrNS」	URI およびローカル名が指定された属性の値を戻します。
4-55 ページ「XmlDomGetAttrNode」	属性名を持つ属性を取得します。
4-56 ページ「XmlDomGetAttrNodeNS」	属性名を持つ属性（名前空間認識バージョン）を取得します。
4-57 ページ「XmlDomGetChildrenByTag」	指定されたタグ名（名前空間を認識しないバージョン）を持つ要素の子を戻します。
4-58 ページ「XmlDomGetChildrenByTagNS」	タグ名（名前空間認識バージョン）を持つ要素の子を戻します。
4-38 ページ「XmlDomGetDocElemsByTag」	ドキュメント要素を取得します。
4-39 ページ「XmlDomGetDocElemsByTagNS」	ドキュメント要素（名前空間認識バージョン）を取得します。
4-61 ページ「XmlDomGetTag」	要素ノードのタグ名を戻します。
4-61 ページ「XmlDomHasAttr」	名前付き属性は存在しますか。
4-62 ページ「XmlDomHasAttrNS」	名前付き属性（名前空間認識バージョン）は存在しますか。
4-63 ページ「XmlDomRemoveAttr」	指定された名前を持つ属性を削除します。
4-63 ページ「XmlDomRemoveAttrNS」	指定された URI およびローカル名を持つ属性を削除します。
4-64 ページ「XmlDomRemoveAttrNode」	属性ノードを削除します。
4-65 ページ「XmlDomSetAttr」	要素の新しい属性を設定します。
4-65 ページ「XmlDomSetAttrNS」	要素の新しい属性（名前空間認識バージョン）を設定します。
4-66 ページ「XmlDomSetAttrNode」	属性ノードを設定します。
4-67 ページ「XmlDomSetAttrNodeNS」	属性ノード（名前空間認識バージョン）を設定します。

XmlDomGetAttr

（名前で指定された）要素の属性値を戻します。属性には空の文字列を値として設定できますが、NULL にはできません。要素に指定された名前を持つ属性がない場合は、NULL が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
name	IN	属性名。

戻り値

(oratext *) 名前付き属性の値（データ・エンコーディング。NULL の可能性があります)。

関連項目： XmlDomGetAttrNS、XmlDomGetAttrs、XmlDomGetAttrNode

XmlDomGetAttrNS

（URI およびローカル名で指定された）要素の属性値を戻します。属性には空の文字列を値として設定できますが、NULL にはできません。要素に指定された名前を持つ属性がない場合は、NULL が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
uri	IN	属性の名前空間 URI（データ・エンコーディング）。
local	IN	属性のローカル名（データ・エンコーディング）。

戻り値

(oratext *) 名前付き属性の値（データ・エンコーディング。NULL の可能性があります）。

関連項目： XmlDomGetAttr、XmlDomGetAttrs、XmlDomGetAttrNode

XmlDomGetAttrNode

名前で指定された要素の属性を戻します。ノードが要素ではないか、または名前付き属性が存在しない場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
name	IN	属性名（データ・エンコーディング）。

戻り値

(xmlattrnode *) 指定した名前を持つ属性（または NULL）。

関連項目： XmlDomGetAttrNodeNS、XmlDomGetAttr

XmlDomGetAttrNodeNS

URI およびローカル名で指定された要素の属性を戻します。ノードが要素ではないか、または名前付き属性が存在しない場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
uri	IN	属性の名前空間 URI（データ・エンコーディング）。
local	IN	属性のローカル名（データ・エンコーディング）。

戻り値

(xmlattrnode *) 指定した URI/ ローカル名を持つ属性ノード（または NULL）。

関連項目： XmlDomGetAttrNode、XmlDomGetAttr

XmlDomGetChildrenByTag

指定されたタグ名を持つ要素の子のリストを、ツリーの先行順走査で検出された順に戻します。タグ名は、データ・エンコーディング内にある必要があります。特殊名「*」を指定すると、すべてのタグ名に一致します。NULL 名を指定すると、何も一致しません。タグ名では大 / 小文字が区別されます。このファンクションは、名前空間を認識しません。完全なタグ名が比較されます。同じ URI にマップされる 2 つの接頭辞を比較すると、比較は失敗します。名前空間を認識するバージョンについては、「XmlDomGetChildrenByTagNS」を参照してください。戻されるリストは、XmlDomFreeNodeList を使用して解放できます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
name	IN	一致させるタグ名（データ・エンコーディング）。すべてのタグ名を一致させるには「*」を指定します。

戻り値

(xmlnodelist *) 一致する子のノード・リスト。

関連項目： XmlDomGetChildrenByTagNS、XmlDomFreeNodeList

XmlDomGetChildrenByTagNS

指定された URI およびローカル名を持つ要素の子のリストを、ツリーの先行順走査で検出された順に戻します。URI およびローカル名は、データ・エンコーディング内にある必要があります。特殊名「*」を指定すると、すべての URI またはタグ名に一致します。NULL 名を指定すると、何も一致しません。名前では大 / 小文字が区別されます。名前空間を認識しないバージョンについては、「XmlDomGetChildrenByTag」を参照してください。戻されるリストは、XmlDomFreeNodeList を使用して解放できます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
uri	IN	一致させる名前空間 URI（データ・エンコーディング）。すべての名前空間 URI を一致させるには「*」を指定します。
local	IN	一致させるローカル名（データ・エンコーディング）。すべてのローカル名を一致させるには「*」を指定します。

戻り値

(xmlnodelist *) 一致する子のノード・リスト。

関連項目： XmlDomGetChildrenByTag、XmlDomFreeNodeList

XmlDomGetElemsByTag

指定されたタグ名を持つ（ルート・ノードをルートとするドキュメント・ツリー内の）すべての要素のリストを、ツリーの先行順走査で検出した順に戻します。ルートが NULL の場合は、ドキュメント全体を検索します。タグ名は、データ・エンコーディング内にある必要があります。特殊名「*」を指定すると、すべてのタグ名に一致します。NULL 名を指定すると、何も一致しません。タグ名では大 / 小文字が区別されます。このファンクションは、名前空間を認識しません。完全なタグ名が比較されます。同じ URI にマップされる 2 つの接頭辞を比較すると、比較は失敗します。名前空間を認識するバージョンについては、「XmlDomGetElemsByTagNS」を参照してください。戻されるリストは、XmlDomFreeNodeList を使用して解放できます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
name	IN	一致させるタグ名（データ・エンコーディング）。すべてのタグ名を一致させるには「*」を指定します。

戻り値

(xmlnodelist *) 一致する要素のノード・リスト。

関連項目： XmlDomGetElemsByTagNS、XmlDomFreeNodeList

XmlDomGetElemsByTagNS

指定された URI およびローカル名を持つ（ルート・ノードをルートとするドキュメント・ツリー内の）すべての要素のリストを、ツリーの先行順走査で検出した順に戻します。ルートが NULL の場合は、ドキュメント全体を検索します。タグ名は、データ・エンコーディング内にある必要があります。特殊名「*」を指定すると、すべてのタグ名に一致します。NULL 名を指定すると、何も一致しません。タグ名では大 / 小文字が区別されます。このファンクションは、名前空間を認識しません。完全なタグ名が比較されます。同じ URI にマップされる 2 つの接頭辞を比較すると、比較は失敗します。戻されるリストは、XmlDomFreeNodeList を使用して解放できます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
uri	IN	一致させる名前空間 URI（データ・エンコーディング）。すべての名前空間 URI を一致させるには「*」を指定します。
local	IN	一致させるローカル名（データ・エンコーディング）。すべてのローカル名を一致させるには「*」を指定します。

戻り値

(xmlnodelist *) 一致する要素のノード・リスト。

関連項目： XmlDomGetDocElemsByTag、XmlDomFreeNodeList

XmlDomGetTag

ノードの tagName を戻します。これはノードの名前と同義です。W3C のワーキング・グループでは、DOM1.0 仕様について、「Node インタフェースには汎用の nodeName 属性があるが、Element インタフェース上には tagName 属性がある。これらの 2 つの属性には同じ値が含まれている必要があるが、ワーキング・グループでは、DOM API が様々なユーザー層のニーズを満たす必要があることを考慮し、両方をサポートする価値がある」と考えています。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	Element ノード。

戻り値

(oratext *) 要素名（データ・エンコーディング)。

関連項目： XmlDomGetNodeName

XmlDomHasAttr

要素に指定された名前を持つ属性があるかどうかを判別します。属性がある場合は TRUE、ない場合は FALSE を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	Element ノード。
name	IN	属性名（データ・エンコーディング）。

戻り値

(boolean) 要素に指定された名前を持つ属性がある場合は TRUE。

関連項目： XmlDomHasAttrNS

XmlDomHasAttrNS

要素に指定された URI およびローカル名を持つ属性があるかどうかを判別します。属性がある場合は TRUE、ない場合は FALSE を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	Element ノード。
uri	IN	属性の名前空間 URI（データ・エンコーディング）。
local	IN	属性のローカル名（データ・エンコーディング）。

戻り値

(boolean) 要素に指定された URI/ ローカル名を持つ属性がある場合は TRUE。

関連項目： XmlDomHasAttr

XmlDomRemoveAttr

（名前で指定された）属性を削除します。削除された属性がデフォルト値を持つ場合は、そのデフォルト値を使用してすぐに再作成されます。属性は属性の要素のリストから削除されますが、属性ノード自体は破棄されません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
name	IN	属性名（データ・エンコーディング）。

関連項目： XmlDomRemoveAttrNS、XmlDomRemoveAttrNode

XmlDomRemoveAttrNS

（URI およびローカル名で指定された）属性を削除します。削除された属性がデフォルト値を持つ場合は、そのデフォルト値を使用してすぐに再作成されます。属性は属性の要素のリストから削除されますが、属性ノード自体は破棄されません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。

パラメータ	In/Out	説明
uri	IN	属性の名前空間 URI。
local	IN	属性のローカル名。

関連項目： XmlDomRemoveAttr、XmlDomRemoveAttrNode

XmlDomRemoveAttrNode

属性を要素から削除します。属性がデフォルト値を持つ場合は、そのデフォルト値（FALSE に設定された Specified）を使用してすぐに再作成されます。正常に終了した場合は削除された属性、エラーが発生した場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
oldAtrr	IN	削除する属性ノード。

戻り値

(xmlattrnode *) 置換された属性ノード（または NULL)。

関連項目： XmlDomRemoveAttr

XmlDomSetAttr

指定された名前および値（データ・エンコーディング内にある必要があります）を使用して、要素の新しい属性を作成します。指定された属性名を持つ属性がすでに存在する場合は、単純にその値が置換されます。名前および値は、検証、変換またはチェックされません。値は解析されないため、実体参照は拡張されません。属性の指定されたフラグが設定されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
name	IN	属性名（データ・エンコーディング）。
value	IN	属性値（データ・エンコーディング）。

関連項目： XmlDomSetAttrNS、XmlDomCreateAttr、XmlDomSetAttrValue、XmlDomRemoveAttr

XmlDomSetAttrNS

指定された URI、ローカル名および値（データ・エンコーディング内にある必要があります）を使用して、要素の新しい属性を作成します。指定された属性名を持つ属性がすでに存在する場合は、単純にその値が置換されます。名前および値は、検証、変換またはチェックされません。

値は解析されないため、実体参照は拡張されません。

属性の specified フラグが設定されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
uri	IN	属性の名前空間 URI（データ・エンコーディング）。
qname	IN	属性の修飾名（データ・エンコーディング）。
value	IN	属性値（データ・エンコーディング）。

関連項目： XmlDomSetAttr、XmlDomCreateAttr、XmlDomSetAttrValue、XmlDomRemoveAttr

XmlDomSetAttrNode

要素に新しい属性を追加します。指定された名前を持つ属性がすでに存在する場合は、その属性が置換され、oldNode によって古い属性が戻されます。新しい属性である場合、その属性が要素のリストに追加され、oldNode が NULL に設定されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
newAttr	IN	追加する属性ノード。

戻り値

(xmlattrnode *) 置換された属性ノード（または NULL）。

関連項目： XmlDomSetAttrNodeNS、XmlDomCreateAttr、XmlDomSetAttrValue

XmlDomSetAttrNodeNS

要素に新しい属性を追加します。newNode の URI およびローカル名を持つ属性がすでに存在する場合は、その属性が置換され、oldNode によって古い属性が戻されます。新しい属性である場合、その属性が要素のリストに追加され、oldNode が NULL に設定されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	要素ノード。
newAttr	IN	追加する属性ノード。

戻り値

(xmlattrnode *) 置換された属性ノード（または NULL）。

関連項目： XmlDomSetAttrNode、XmlDomCreateAttr、XmlDomSetAttrValue

Entity インタフェース

表 4-6 に、Entity インタフェースを介して使用できるメソッドを示します。

表 4-6 Entity メソッドの概要 : DOM パッケージ

ファンクション	概要
4-68 ページ「XmlDomGetEntityNotation」	エンティティの表記法を取得します。
4-69 ページ「XmlDomGetEntityPubID」	エンティティの公開識別子を取得します。
4-69 ページ「XmlDomGetEntitySysID」	エンティティのシステム識別子を取得します。
4-70 ページ「XmlDomGetEntityType」	エンティティの型を取得します。

XmlDomGetEntityNotation

解析対象外のエンティティの場合、（データ・エンコーディングの）表記法の名前を戻します。解析対象エンティティおよびそれ以外のノードのタイプの場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
ent	IN	エンティティ・ノード。

戻り値

(oratext *) エンティティの表記法（データ・エンコーディング。NULL の可能性があります）。

関連項目： XmlDomGetEntityPubID、XmlDomGetEntitySysID

XmlDomGetEntityPubID

(データ・エンコーディングの) エンティティの公開識別子を戻します。ノードがエンティティではないか、または定義済の公開識別子を持っていない場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
ent	IN	エンティティ・ノード。

戻り値

(oratext *) エンティティの公開識別子 (データ・エンコーディング。NULL の可能性があります)。

関連項目： XmlDomGetEntitySysID、XmlDomGetEntityNotation

XmlDomGetEntitySysID

(データ・エンコーディングの) エンティティのシステム識別子を戻します。ノードがエンティティではないか、または定義済のシステム識別子を持っていない場合は NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
ent	IN	エンティティ・ノード。

戻り値

(oratext *) エンティティのシステム識別子（データ・エンコーディング。NULL の可能性があります)。

関連項目： XmlDomGetEntityPubID、XmlDomGetEntityNotation

XmlDomGetEntityType

エンティティが汎用（TRUE）かパラメータ（FALSE）かを示すブール値を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
ent	IN	エンティティ・ノード。

戻り値

(boolean) 汎用エンティティの場合は TRUE、パラメータ・エンティティの場合は FALSE。

関連項目： XmlDomGetEntityPubID、XmlDomGetEntitySysID、XmlDomGetEntityNotation

NamedNodeMap インタフェース

表 4-7 に、NamedNodeMap インタフェースを介して使用できるメソッドを示します。

表 4-7 NamedNodeMap メソッドの概要 : DOM パッケージ

ファンクション	概要
4-71 ページ「XmlDomGetNamedItem」	リストから名前付きノードを戻します。
4-72 ページ「XmlDomGetNamedItemNS」	リスト（名前空間認識バージョン）から名前付きノードを戻します。
4-73 ページ「XmlDomGetNodeMapItem」	リスト内のノード数を戻します。
4-74 ページ「XmlDomGetNodeMapLength」	名前付きノード・マップの長さを戻します。
4-74 ページ「XmlDomRemoveNamedItem」	名前付きノード・マップからノードを削除します。
4-75 ページ「XmlDomRemoveNamedItemNS」	名前付きノード・マップ（名前空間認識バージョン）からノードを削除します。
4-76 ページ「XmlDomSetNamedItem」	名前付きノード・リストのノードを設定します。
4-77 ページ「XmlDomSetNamedItemNS」	名前付きノード・マップ（名前空間認識バージョン）のノードを設定します。

XmlDomGetNamedItem

名前（データ・エンコーディング内にある必要があります）で指定された NamedNodeMap から、項目を取得します。これは、名前空間を認識しないファンクションです。修飾名全体で（大 / 小文字を区別して）照合します。このファンクションは、一致する項目の索引も戻されるという点で、DOM 仕様とは異なります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
name	IN	取得するノードの名前。

戻り値

(xmlnode *) 指定した名前を持つ Node（または NULL)。

関連項目： XmlDomGetNamedItemNS、XmlDomGetNodeMapItem、XmlDomGetNodeMapLength

XmlDomGetNamedItemNS

URI およびローカル名（データ・エンコーディング内にある必要があります）で指定された NamedNodeMap から、項目を取得します。このファンクションは、一致する項目の索引も戻されるという点で、DOM 仕様とは異なります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
uri	IN	取得するノードの名前空間 URI（データ・エンコーディング)。
local	IN	取得するノードのローカル名（データ・エンコーディング)。

戻り値

(xmlnode *) 指定したローカル名および名前空間 URI を持つノード（または NULL)。

関連項目： XmlDomGetNamedItem、XmlDomGetNodeMapItem、XmlDomGetNodeMapLength

XmlDomGetNodeMapItem

名前（データ・エンコーディング内にある必要があります）で指定された NamedNodeMap から、項目を取得します。これは、名前空間を認識しないファンクションです。修飾名全体で（大 / 小文字を区別して）照合します。このファンクションは、一致する項目の索引も戻されるという点で、DOM 仕様とは異なります。W3C 仕様の名前付きの項目が使用されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
index	IN	0 マップの 0（ゼロ）ベースの索引

戻り値

(xmlnode *) マップ内の n 番目の位置にあるノード（または NULL）。

関連項目： XmlDomGetNamedItem、XmlDomSetNamedItem、XmlDomRemoveNamedItem、XmlDomGetNodeMapLength

XmlDomGetNodeMapLength

NamedNodeMap 内のノードの数（長さ）を戻します。ノードは索引によって参照されるため、有効な索引の範囲は 0（ゼロ）から length-1 です。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap

戻り値

(ub4) NamedNodeMap 内のノードの数。

関連項目： XmlDomGetNodeMapItem、XmlDomGetNamedItem

XmlDomRemoveNamedItem

名前で指定された NamedNodeMap からノードを削除します。これは、名前空間を認識しないファンクションです。修飾名全体で（大 / 小文字を区別して）照合します。削除されたノードが（指定されていない）デフォルト値を持つ属性の場合は、すぐに置換されます。削除されたノードが戻されます。削除が行われなかった場合は、NULL が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
name	IN	削除するノードの名前。

戻り値

(xmlnode *) このマップから削除されたノード。

関連項目： XmlDomRemoveNamedItemNS、XmlDomGetNamedItem、XmlDomGetNamedItemNS、XmlDomSetNamedItem、XmlDomSetNamedItemNS

XmlDomRemoveNamedItemNS

URI およびローカル名で指定された NamedNodeMap からノードを削除します。削除されたノードが（指定されていない）デフォルト値を持つ属性の場合は、すぐに置換されます。削除されたノードが戻されます。削除が行われなかった場合は、NULL が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
uri	IN	削除されるノードの名前空間 URI（データ・エンコーディング）。
local	IN	削除されるノードのローカル名（データ・エンコーディング）。

戻り値

(xmlnode *) このマップから削除されたノード。

関連項目： XmlDomRemoveNamedItem、XmlDomGetNamedItem、XmlDomGetNamedItemNS、XmlDomSetNamedItem、XmlDomSetNamedItemNS

XmlDomSetNamedItem

NamedNodeMap に新しいノードを追加します。指定された名前を持つノードがすでに存在する場合は、古いノードは置換され、戻されます。該当する名前付きノードが存在しない場合は、新しいノードがマップに追加され、古いノードが NULL に設定されます。これは、名前空間を認識しないファンクションです。修飾名全体で（大 / 小文字を区別して）照合します。ノードのタイプによっては決まった名前（Text、Comment など）があるため、同じタイプに別の名前を設定しようとすると、常に置換が行われます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
newNode	IN	マップに格納される新しいノード。

戻り値

(xmlnode *) 置換されたノード（または NULL)。

関連項目： XmlDomSetNamedItemNS、XmlDomGetNamedItem、XmlDomGetNamedItemNS、XmlDomGetNodeMapItem、XmlDomGetNodeMapLength

XmlDomSetNamedItemNS

NamedNodeMap に新しいノードを追加します。指定された URI およびローカル名を持つノードがすでに存在する場合、古いノードは置換され、戻されます。該当する名前付きノードが存在しない場合は、新しいノードがマップに追加され、古いノードが NULL に設定されます。ノードのタイプによっては決まった名前（Text、Comment など）があるため、同じタイプに別の名前を設定しようとすると、常に置換が行われます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
map	IN	NamedNodeMap
newNode	IN	マップに格納される新しいノード。

戻り値

(xmlnode *) 置換された Node（または NULL）。

関連項目： XmlDomSetNamedItem、XmlDomGetNamedItem、XmlDomGetNamedItemNS、XmlDomGetNodeMapItem、XmlDomGetNodeMapLength

Node インタフェース

表 4-8 に、Node インタフェースを介して使用できるメソッドを示します。

表 4-8 Text メソッドの概要 : DOM パッケージ

ファンクション	概要
4-80 ページ「XmlDomAppendChild」	子ノード・リストに新しい子ノードを追加します。
4-81 ページ「XmlDomCleanNode」	ノードを消去します（DOM 割当てを解放します）。
4-81 ページ「XmlDomCloneNode」	ノードを複製します。
4-82 ページ「XmlDomFreeNode」	XmlDomCreateXXX を使用して割り当てられたノードを解放します。
4-83 ページ「XmlDomGetAttrs」	ノードの属性を戻します。
4-83 ページ「XmlDomGetChildNodes」	ノードの子を戻します。
4-84 ページ「XmlDomGetDefaultNS」	ノードのデフォルトの名前空間を取得します。
4-85 ページ「XmlDomGetFirstChild」	ノードの最初の子ノードを戻します。
4-85 ページ「XmlDomGetFirstPfnsPair」	最初の接頭辞名前空間の組を取得します。
4-86 ページ「XmlDomGetLastChild」	ノードの最後の子ノードを戻します。
4-87 ページ「XmlDomGetNextPfnsPair」	後続の接頭辞名前空間の組を取得します。
4-87 ページ「XmlDomGetNextSibling」	ノードの直後の兄弟関係を戻します。
4-88 ページ「XmlDomGetNodeLocal」	ノードの修飾名のローカル部分を、NULL で終了する文字列として取得します。
4-89 ページ「XmlDomGetNodeLocalLen」	ノードの修飾名のローカル部分を、長さがエンコードされた文字列として取得します。
4-90 ページ「XmlDomGetNodeName」	ノードの名前を、NULL で終了する文字列として取得します。
4-91 ページ「XmlDomGetNodeNameLen」	ノードの名前を、長さがエンコードされた文字列として取得します。
4-92 ページ「XmlDomGetNodePrefix」	ノードの名前空間接頭辞を戻します。
4-92 ページ「XmlDomGetNodeType」	ノードの数値型コードを取得します。

表 4-8 Text メソッドの概要 : DOM パッケージ（続き）

ファンクション	概要
4-94 ページ「XmlDomGetNodeURI」	ノードの名前空間 URI を、NULL で終了する文字列として戻します。
4-95 ページ「XmlDomGetNodeURILen」	ノードの名前空間 URI を、長さがエンコードされた文字列として戻します。
4-96 ページ「XmlDomGetNodeValue」	ノードの値を、NULL で終了する文字列として取得します。
4-97 ページ「XmlDomGetNodeValueLen」	ノードの値を、長さがエンコードされた文字列として取得します。
4-98 ページ「XmlDomGetNodeValueStream」	ノード値のストリーム形式（チャンク）を取得します。
4-99 ページ「XmlDomGetOwnerDocument」	ノードの所有者ドキュメントを取得します。
4-99 ページ「XmlDomGetParentNode」	親ノードを取得します。
4-100 ページ「XmlDomGetPrevSibling」	ノードの直前の兄弟関係を戻します。
4-100 ページ「XmlDomGetSourceEntity」	入力ファイルが外部エンティティの場合、エンティティ・ノードを戻します。
4-101 ページ「XmlDomGetSourceLine」	ノードのソース行番号を戻します。
4-101 ページ「XmlDomGetSourceLocation」	ノードのソース位置（パス、URI など）を戻します。
4-61 ページ「XmlDomHasAttr」	名前付き属性は存在しますか。
4-102 ページ「XmlDomHasChildNodes」	ノードが子ノードを持っているかどうかをテストします。
4-103 ページ「XmlDomInsertBefore」	子ノード・リストに新しい子ノードを挿入します。
4-104 ページ「XmlDomNormalize」	隣接するテキスト・ノードをマージしてノードを正規化します。
4-104 ページ「XmlDomNumAttrs」	要素の属性の数を戻します。
4-105 ページ「XmlDomNumChildNodes」	ノードの子の数を戻します。
4-105 ページ「XmlDomPrefixToURI」	接頭辞の名前空間 URI を取得します。
4-106 ページ「XmlDomRemoveChild」	既存の子ノードを削除します。
4-107 ページ「XmlDomReplaceChild」	既存のノードの子ノードを置換します。
4-108 ページ「XmlDomSetDefaultNS」	ノードのデフォルトの名前空間を設定します。

表 4-8 Text メソッドの概要 : DOM パッケージ（続き）

ファンクション	概要
4-108 ページ「XmlDomSetNodePrefix」	ノードの名前空間接頭辞を設定します。
4-109 ページ「XmlDomSetNodeValue」	ノード値を設定します。
4-110 ページ「XmlDomSetNodeValueLen」	ノードの値を、長さがエンコードされた文字列として設定します。
4-111 ページ「XmlDomSetNodeValueStream」	ノード値のストリーム形式（チャンク）を設定します。
4-112 ページ「XmlDomValidate」	ノードを現行の DTD に対して検証します。

XmlDomAppendChild

親ノードの子リストの最後にノードを追加し、その新しいノードを戻します。newChild が DocumentFragment の場合は、すべての子が元の順序で追加されます。DocumentFragment ノード自体は追加されません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
parent	IN	新しいノードを受け取る親ノード。
newChild	IN	追加するノード。

戻り値

(xmlnode *) 追加されたノード。

関連項目： XmlDomInsertBefore、XmlDomReplaceChild

XmlDomCleanNode

DOM 自体によって割り当てられたノードの一部を解放しますが、子ノードに再帰したり、ノードの属性に影響することはありません。ノードの一部（名前など）を解放した後に、その一部（XmlDomGetNodeName など）を取得するための DOM コールは、NULL ポインタを戻す必要があります。DOM によって制御されるノードの一部、およびユーザーによって制御されるノードの一部を管理するために使用されます。正常な解放をコールすると、すべての割当ては DOM によって行われ、ユーザーの割当てのみが残ります。ユーザーは、ユーザー自身の割当てを解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	消去されるノード。

関連項目： XmlDomFreeNode

XmlDomCloneNode

ノードの複製を作成して戻します。複製ノードは親を持ちません。要素を複製すると、XML プロセッサによって生成されたデフォルト値の属性を含む、すべての属性およびその値がコピーされます。ただし、このメソッドでは、ノードに含まれるテキストは、子であるテキスト・ノードに含まれているため、ディープ・クローンの場合以外はコピーされません。他のタイプのノードを複製すると、単純にノードのコピーが戻されます。未指定の属性ノードの複製も指定されます。deep が TRUE の場合、ノードの子はすべて再帰的に複製され、複製されたノードは複製された子を持ちます。ディープ・クローン以外は子を持ちません。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
deep	IN	子を再帰的に複製する場合は TRUE。

戻り値

(xmlnode *) 複製（複製された）ノード。

関連項目： XmlDomImportNode

XmlDomFreeNode

XmlDomCreate*XXX* を使用して割り当てられたノードを解放します。ノードに対応するすべてのリソースを解放してから、ノード自体を解放します。ノードには、DOM で制御される部分とユーザーが制御する部分があります。DOM は誰がどのノードを所有するかを追跡するフラグを保持し、DOM 独自の割当てのみを解放します。ユーザーは、XmlDomFreeNode をコールする前に、ノードのユーザー独自の割当て部分を解放します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	解放される XML ノード。

関連項目： XmlDomCleanNode

XmlDomGetAttrs

要素ノードの属性の NamedNodeMap を戻します。属性がない場合は NULL を戻します。ノードのタイプがそれ以外の場合は、NULL が戻されます。要素に属性があっても、すべて削除されている場合は、空のリストが戻されます。したがって、リストがあっても、要素に属性があるとはかぎりません。XmlDomNumAttrs を使用してリストのサイズをチェックするか、先に XmlDomHasChildNodes を使用する必要があります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	XML 要素ノード。

戻り値

(xmlnamedmap *) ノードの属性の NamedNodeMap。

関連項目： XmlDomNumAttrs、XmlDomHasChildNodes

XmlDomGetChildNodes

ノードの子のリストを戻します。子がない場合は NULL を戻します。Element、Document、DTD および DocumentFragment のノードのみが子を持ちます。それ以外のタイプはすべて、NULL を戻します。

ノードに子があっても、それらがすべて削除されている場合は、空のリストが戻される可能性があります。つまり、リストが存在しても、メンバーは存在しません。したがって、リストのみがあっても、ノードが子を持っているとはかぎりません。XmlDomNumChildNodes を使用してリストのサイズをチェックするか、先に XmlDomHasChildNodes を使用する必要があります。

xmlnodelist 構造体は不透明なため、NodeList インタフェース内のファンクションでのみ操作できます。

戻されるリストは最新です。元のノードへのすべての変更が、すぐに反映されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnodelist *) ノードのすべての子を含む、最新の NodeList。

XmlDomGetDefaultNS

ノードのデフォルトの名前空間を取得します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	要素または属性 DOM ノード。

戻り値

(oratext *) ノードのデフォルトの名前空間（データ・エンコーディング。NULL の可能性があります)。

XmlDomGetFirstChild

ノードの最初の子ノードを戻します。ノードに子ノードが存在しない場合は、NULL を戻します。Element、Document、DTD および DocumentFragment の各ノードのみが子を持ちます。それ以外のタイプはすべて、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnode *) ノードの最初の子ノード。

関連項目： XmlDomGetLastChild、XmlDomHasChildNodes、XmlDomGetChildNodes、XmlDomNumChildNodes

XmlDomGetFirstPfnsPair

このファンクションを使用すると、実装では指定されたノードで使用できるすべての使用可能な URI の接頭辞バインドの反復を高速化できます。これにより、最初の URI の接頭辞マッピングの状態構造、接頭辞および URI が戻されます。状態構造は、残りの組の XmlDomGetNextPfnsPair に渡される必要があります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
prefix	OUT	最初のマッピングの接頭辞（データ・エンコーディング）。
uri	OUT	最初のマッピングの URI（データ・エンコーディング）。

戻り値

(xmlpfnspair *) 反復オブジェクト。接頭辞がない場合は NULL を戻します。

XmlDomGetLastChild

ノードの最後の子ノードを戻します。ノードに子ノードが存在しない場合は、NULL を戻します。Element、Document、DTD および DocumentFragment の各ノードのみが子を持ちます。それ以外のタイプはすべて、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnode *) ノードの最後の子ノード。

関連項目： XmlDomGetFirstChild、XmlDomHasChildNodes、XmlDomGetChildNodes、XmlDomNumChildNodes

XmlDomGetNextPfnsPair

このファンクションを使用すると、実装では指定されたノードで使用できるすべての使用可能な URI の接頭辞バインドの反復を高速化できます。XmlDomGetFirstPfnsPair からイテレータ構造を指定すると、次の URI の接頭辞マッピングが戻されます。NULL が戻されるまで、XmlDomGetNextPfnsPair のコールを繰り返します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
prefix	OUT	次のマッピングの接頭辞（データ・エンコーディング）。
uri	OUT	次のマッピングの URI（データ・エンコーディング）。

戻り値

(xmlpfnspair *) 反復オブジェクト。組がなくなった場合は NULL を戻します。

XmlDomGetNextSibling

DOM ツリーの同じレベルにあるノードに続いて、ノードを戻します。つまり、親ノードの子ノードごとに、その子ノードの直後の兄弟関係が、その子ノードに続く子ノードになります。子ノードが親ノードの最後の子ノードである場合、または子ノードが親ノードを持たない場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnode *) 同じレベルにあるノードの直後のノード。

関連項目： XmlDomGetPrevSibling

XmlDomGetNodeLocal

ノードの名前空間のローカル名を、NULL で終了する文字列として戻します。ノード名が完全に修飾されていない（接頭辞がない）場合、ローカル名はノード名と同じになります。

長さがエンコードされたバージョンは、XmlDomGetNodeLocalLen として使用できます。これは、ローカル名をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(oratext *) ノードのローカル名（データ・エンコーディング）。

関連項目： XmlDomGetNodeLocalLen、XmlDomGetNodePrefix、XmlDomGetNodeURI

XmlDomGetNodeLocalLen

ノードの名前空間のローカル名を、長さがエンコードされた文字列として戻します。ノード名が完全に修飾されていない（接頭辞がない）場合、ローカル名はノード名と同じになります。

NULL で終了するバージョンは、XmlDomGetNodeLocal として使用できます。これは、ローカル名を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、ノードのデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Get ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0（ゼロ）以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
buf	IN	入力バッファ（オプション）。
buflen	IN	入力バッファ長（オプション）。
len	OUT	ローカル名の長さ（文字数）。

戻り値

(oratext *) ノードのローカル名（データ・エンコーディング）。

関連項目： XmlDomGetNodeLocal、XmlDomGetNodePrefix、XmlDomGetNodeURILen

XmlDomGetNodeName

(データ・コーディングの) ノードの (完全修飾された) 名前を、NULL で終了する文字列として戻します。たとえば、bar¥0、foo:bar¥0 などです。

ノードのタイプによっては、"#text"、"#cdata-section"、"#comment"、"#document"、"#document-fragment" などの決まった名前があります。

ノード名は 1 度作成すると変更できないため、一致する SetNodeName ファンクションはありません。

長さに基づくバージョンは、XmlDomGetNodeNameLen として使用できます。これは、ノード名をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(oratext *) ノードの名前 (データ・エンコーディング)。

関連項目： XmlDomGetNodeNameLen

XmlDomGetNodeNameLen

(データ・コーディングの) ノードの (完全修飾された) 名前を、長さがエンコードされた文字列として戻します。たとえば、"bar", 3、"foo:bar", 7 などです。

ノードのタイプによっては、"#text"、"#cdata-section"、"#comment"、"#document"、"#document-fragment" などの決まった名前があります。

ノード名は 1 度作成すると変更できないため、一致する SetNodeName ファンクションはありません。

NULL で終了するバージョンは、XmlDomGetNodeName として使用できます。これは、ノード名を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、ノード名は、長さがエンコードされたデータとして内部的に格納されます。長さがエンコードされた Get*XXX* ファンクションを使用すると、名前をコピーしたり、名前を NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0 (ゼロ) 以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
buf	IN	入力バッファ (オプション)。
buflen	IN	入力バッファ長 (オプション)。
len	OUT	名前の長さ (文字数)。

戻り値

(oratext *) ノードの名前。名前の長さは 'len' で設定されます。

関連項目： XmlDomGetNodeName

XmlDomGetNodePrefix

ノードの名前空間接頭辞を（NULL で終了する文字列として）戻します。ノード名が完全に修飾されていない（接頭辞がない）場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(oratext *) ノードの名前空間接頭辞（データ・エンコーディング。NULL の可能性があります)。

XmlDomGetNodeType

ノードの型コードを戻します。型の名前および数値は DOM 仕様に適合します。

- 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

Oracle の拡張機能のその他のノードの型は、次のとおりです。

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

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnodetype) ノードの数値型コード。

XmlDomGetNodeURI

（データ・エンコーディングの）ノードの名前空間 URI を、NULL で終了する文字列として戻します。ノード名が修飾されていない（名前空間の接頭辞が含まれない）場合は、ノードが作成されたときに、デフォルトの名前空間が使用されます（NULL になる可能性があります）。

長さがエンコードされたバージョンは、XmlDomGetNodeURILen として使用できます。これは、URI をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(oratext *) ノードの名前空間 URI（データ・エンコーディング。NULL の可能性があります）。

関連項目： XmlDomGetNodeURILen、XmlDomGetNodePrefix、XmlDomGetNodeLocal

XmlDomGetNodeURILen

(データ・エンコーディングの) ノードの名前空間 URI を、長さがエンコードされた文字列として戻します。ノード名が修飾されていない (名前空間の接頭辞が含まれない) 場合は、ノードが作成されたときに、デフォルトの名前空間が使用されます (NULL になる可能性があります)。

NULL で終了するバージョンは、XmlDomGetNodeURI として使用できます。これは、URI 値を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、ノードのデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Get ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0 (ゼロ) 以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
buf	IN	入力バッファ (オプション)。
buflen	IN	入力バッファ長 (オプション)。
len	OUT	URI の長さ (文字数)。

戻り値

(oratext *) ノードの名前空間 URI (データ・エンコーディング。NULL の可能性があります)。

関連項目： XmlDomGetNodeURI、XmlDomGetNodePrefix、XmlDomGetNodeLocal

XmlDomGetNodeValue

ノードの（文字データに対応する）値を、NULL で終了する文字列として戻します。文字および汎用エンティティが置換されています。Attr、CDATA、Comment、ProcessingInstruction および Text の各ノードのみが値を持ちます。それ以外のタイプはすべて、NULL 値を持ちます。

長さがエンコードされたバージョンは、XmlDomGetNodeValueLen として使用できます。これは、ノード値をポインタおよび長さとして戻し、データが XMLType バックエンドのデータストアを使用するとわかっている場合に使用します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(oratext *) ノードの値。

関連項目： XmlDomSetNodeValue、XmlDomGetNodeValueLen

XmlDomGetNodeValueLen

ノードの（文字データに対応する）値を、長さがエンコードされた文字列として戻します。文字および汎用エンティティが置換されています。Attr、CDATA、Comment、PI および Text の各ノードのみが値を持ちます。それ以外のタイプはすべて、NULL 値を持ちます。

NULL で終了するバージョンは、XmlDomGetNodeValue として使用できます。これは、ノード値を NULL で終了する文字列として戻します。バックエンドのデータストアが XMLType であるとわかっている場合、ノードのデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Get ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

両方の入力バッファが NULL 以外で、入力バッファの長さが 0（ゼロ）以外の場合、値は入力バッファに格納されます。それ以外の場合、実装は独自のバッファを戻します。

実際の長さが buflen より大きい場合、切り捨てられた値がバッファにコピーされ、len が実際の長さを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
buf	IN	入力バッファ（オプション）。
buflen	IN	入力バッファ長（オプション）。
len	OUT	値の長さ（バイト数）。

戻り値

(oratext *) ノードの値。

関連項目： XmlDomSetNodeValueLen、XmlDomGetNodeValue

XmlDomGetNodeValueStream

ノードの大きいデータを戻し、分割してユーザーの出力ストリームに送信します。非常に大きいデータの場合、単一の連続するチャンクとして（効率的に）格納できるとはかぎりません。このファンクションは、そのタイプのチャンク・データにアクセスするために使用されます。XMLType のみがデータを（場合によって）チャンクします。XDK のデータは常に連続しています。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
ostream	IN	出力ストリーム・オブジェクト。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

関連項目： XmlDomSetNodeValueStream、XmlDomGetNodeValue、XmlDomGetNodeValueLen

XmlDomGetOwnerDocument

ノードに対応する Document ノードを戻します。各ノードは 1 つのドキュメントのみに属する場合もあれば、どのドキュメントにも対応しない場合もあります（たとえば XmlDomCreateElem などの直後）。元のドキュメント（ノード）が戻されます。孤立したノードの場合は、NULL が戻されます。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmldocnode *) ドキュメント・ノードが存在します。

XmlDomGetParentNode

ノードの親ノードを戻します。Attr、DocumentDocumentFragment、Entity および Notation 以外のノードのタイプはすべて、親を持ちます（この 5 つの例外は、常に NULL の親を持ちます）。ノードが作成された直後で、DOM ツリーにまだ追加されていない場合、またはノードが DOM ツリーから削除されている場合は、親も NULL になります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnode *) ノードの親ノード。

XmlDomGetPrevSibling

DOM ツリーの同じレベルにあるノードの直前のノードを戻します。つまり、親ノードの子ノードごとに、その子ノードの直前の兄弟関係が、その子ノードの前の子ノードになります。ノードが親ノードの最初の子ノードの場合は、NULL を戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlnode *) 同じレベルにあるノードの直前のノード。

関連項目： XmlDomGetNextSibling

XmlDomGetSourceEntity

包含によって特定のノードが作成される外部エンティティ・ノードを戻します。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(xmlentnode *) 入力が外部エンティティから行われる場合は、エンティティ・ノードを戻します。

XmlDomGetSourceLine

ノードが開始された元のソースの行番号を戻します。すべての入力で、最初の行は行 #1 になります。

構文

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

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(ub4) 元の入力ソースのノードの行番号。

XmlDomGetSourceLocation

ノードのソース位置（パス、URI など）を戻します。これは、データ・エンコーディング内ではなくコンパイラ・エンコーディング内にあります。

構文

```
oratext* XmlDomGetSourceLocation(
   xmlctx *xctx,
   xmlnode *node)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(oratext *) 入力ソースのフルパス（コンパイラ・エンコーディング）。

XmlDomHasAttrs

要素が属性を持っているかどうかをテストします。ソートの属性（名前空間または標準）が定義されている場合は TRUE を戻します。

構文

```
boolean XmlDomHasAttrs(
   xmlctx *xctx,
   xmlelemnode *elem)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	XML 要素ノード。

戻り値

(boolean) 要素が属性が持つ場合は TRUE。

XmlDomHasChildNodes

ノードが子ノードを持っているかどうかをテストします。Element、Document、DTD および DocumentFragment の各ノードのみが子を持ちます。XmlDomGetChildNodes がリストを戻した場合でも、ノードが実際に子ノードを持っているとはかぎりません。リストが空の場合もあるため、XmlDomGetChildNodes からの NULL 以外の戻り値は、テストとして使用できません。

構文

```
boolean XmlDomHasChildNodes(
   xmlctx *xctx,
   xmlnode *node)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(boolean) ノードが子を持つ場合は TRUE。

XmlDomInsertBefore

親ノード内の既存の子ノード refChild の前に newChild ノードを挿入します。refChild が NULL の場合は、XmlDomAppendChild ごとに親の子ノードに追加されます。それ以外の場合は、指定された親の子ノードである必要があります。newChild が DocumentFragment の場合は、refChild の前にすべての子が（同じ順序で）挿入されます。DocumentFragment ノード自体は挿入されません。newChild がすでに DOM ツリー内に存在する場合は、まずそれが現行の位置から削除されます。

構文

```
xmlnode* XmlDomInsertBefore(
   xmlctx *xctx,
   xmlnode *parent,
   xmlnode *newChild,
   xmlnode *refChild)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
parent	IN	新しい子ノードを受け取る親ノード。
newChild	IN	挿入するノード。
refChild	IN	参照ノード。

戻り値

(xmlnode *) 挿入されるノード。

関連項目： XmlDomAppendChild、XmlDomReplaceChild、XmlDomRemoveChild

XmlDomNormalize

要素をルートとしたサブツリーを正規化して、隣接する Text ノードの要素の子をマージします。隣接する Text ノードは、通常の解析時に作成されるのではなく、DOM コールによってドキュメントを操作した後にのみ作成されます。

構文

```
void XmlDomNormalize(
   xmlctx *xctx,
   xmlnode *node)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

XmlDomNumAttrs

要素の属性の数を戻します。XmlDomGetAttrs によってリストが戻される場合でも、属性が含まれているとはかぎりません。長さが 0（ゼロ）の空のリストの場合もあります。

構文

```
ub4 XmlDomNumAttrs(
   xmlctx *xctx,
   xmlelemnode *elem)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
elem	IN	XML 要素ノード。

戻り値

（ub4）ノードの属性の数。

XmlDomNumChildNodes

ノードの子ノードの数を戻します。Element、Document、DTD および DocumentFragment の各ノードのみが子を持ちます。それ以外のタイプは、0（ゼロ）を戻します。XmlDomGetChildNodes によってリストが戻される場合でも、子が含まれているとはかぎりません。長さが 0（ゼロ）の空のリストの場合もあります。

構文

```
ub4 XmlDomNumChildNodes(
   xmlctx *xctx,
   xmlnode *node)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。

戻り値

(ub4) ノードの子ノードの数。

XmlDomPrefixToURI

名前空間接頭辞およびノードを指定すると、その接頭辞にマップされた名前空間 URI を戻します。指定されたノードが一致する接頭辞を持たない場合は、その親ノードが試行されます。次に、その親ノードの親ノードが試行され、ルート・ノードまで繰り返し試行されます。接頭辞が定義されていない場合は、NULL を戻します。

構文

```
oratext* XmlDomPrefixToURI(
   xmlctx *xctx,
   xmlnode *node,
   oratext *prefix)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
prefix	IN	マップされる接頭辞。

戻り値

(oratext *) 接頭辞の URI（データ・エンコーディング。一致しない場合は NULL を戻します)。

XmlDomRemoveChild

親ノードの子ノード・リストからノードを削除して、戻します。ノードは孤立しており、削除後、ノードの親は NULL になります。

構文

```
xmlnode* XmlDomRemoveChild(
   xmlctx *xctx,
   xmlnode *oldChild)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
oldChild	IN	削除するノード。

戻り値

(xmlnode *) 削除されたノード。

関連項目： XmlDomAppendChild、XmlDomInsertBefore、XmlDomReplaceChild

XmlDomReplaceChild

oldChild の親の子ノード oldChild を新しいノード newChild で置換し、oldChild（NULL の親を持ち、孤立しています）を戻します。newChild が DocumentFragment の場合は、oldChild のかわりにすべての子が挿入されます。DocumentFragment ノード自体は挿入されません。newChild がすでに DOM ツリー内に存在する場合、まずそれが現行の位置から削除されます。

構文

```
xmlnode* XmlDomReplaceChild(
   xmlctx *xctx,
   xmlnode *newChild,
   xmlnode *oldChild)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
newChild	IN	置換される新しいノード。
oldChild	IN	置換される古いノード。

戻り値

(xmlnode *) 置換されたノード。

関連項目： XmlDomAppendChild、XmlDomInsertBefore、XmlDomRemoveChild

XmlDomSetDefaultNS

ノードのデフォルトの名前空間を設定します。

構文

```
void XmlDomSetDefaultNS(
   xmlctx *xctx,
   xmlnode *node,
   oratext *defns)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	要素または属性 DOM ノード。
defns	IN	ノードの新しいデフォルトの名前空間。

XmlDomSetNodePrefix

ノードの名前空間接頭辞を（NULL で終了する文字列として）設定します。接頭辞が定義されているかどうかは検証しません。新しい接頭辞と古いローカル名から、新しい修飾名が作成されます。新しい修飾名は DOM で制御され、ユーザーは管理できません。

構文

```
void XmlDomSetNodePrefix(
   xmlctx *xctx,
   xmlnode *node,
   oratext *prefix)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
prefix	OUT	新しい名前空間の接頭辞。

XmlDomSetNodeValue

ノードの値（文字データ）を、NULL で終了する文字列として設定します。値は NULL に設定できません。Attr、CDATA、Comment、PI および Text の各ノードのみが値を持ちます。それ以外のタイプの値を設定しようとしても、動作しません。新しい値は、データ・エンコーディング内にある必要があります。検証、変換またはチェックされません。

値はコピーされません。ポインタのみが保存されます。ユーザーがそのデータを永続的にして、解放します。

構文

```
xmlerr XmlDomSetNodeValue(
   xmlctx *xctx,
   xmlnode *node,
   oratext *value)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
value	IN	ノードの新しい値（データ・エンコーディング）。ユーザーが制御します。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

関連項目： XmlDomGetNodeValue、XmlDomSetNodeValueLen

XmlDomSetNodeValueLen

ノードの（文字データに対応する）値を、長さがエンコードされた文字列として設定します。

NULL で終了するバージョンは、XmlDomSetNodeValue として使用できます。これは、ノード値を NULL で終了する文字列として取ります。バックエンドのデータストアが XMLType であるとわかっている場合、ノードのデータは、長さがエンコードされたデータとして内部的に格納されます。長さに基づく Set ファンクションを使用すると、データをコピーしたり、データを NULL で終了する必要がなくなります。

構文

```
xmlerr XmlDomSetNodeValueLen(
   xmlctx *xctx,
   xmlnode *node,
   oratext *value,
   ub4 len)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
value	IN	ノードの新しい値（データ・エンコーディング）。ユーザーが制御します。
len	IN	値の長さ（バイト数）。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

関連項目： XmlDomSetNodeValueLen、XmlDomSetNodeValue

XmlDomSetNodeValueStream

ノードの大きい値（文字データ）を、入力ストリームとは別に設定します。非常に大きいデータの場合、単一の連続するチャンクとして（効率的に）格納できるとはかぎりません。このファンクションは、そのタイプのチャンク・データを格納するために使用されます。XMLType データのみに使用され、XDK のデータは常に連続しています。

構文

```
xmlerr XmlDomSetNodeValueStream(
   xmlctx *xctx,
   xmlnode *node,
   xmlistream *istream)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	XML ノード。
istream	IN	入力ストリーム・オブジェクト。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は、0（ゼロ）を戻します。

関連項目： XmlDomGetNodeValueStream、XmlDomSetNodeValue

XmlDomValidate

ルート・ノードを指定すると、現行の DTD に対して検証されます。

構文

```
xmlerr XmlDomValidate(
   xmlctx *xctx,
   xmlnode *node)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
node	IN	検証するノード。

戻り値

(xmlerr) エラー・コード。XMLERR_OK [0] はノードが有効なことを示します。

NodeList インタフェース

表 4-9 に、NodeList インタフェースを介して使用できるメソッドを示します。

表 4-9 NodeList メソッドの概要 : DOM パッケージ

ファンクション	概要
4-113 ページ「XmlDomFreeNodeList」	XmlDomGetElemsByTag などによって戻されるノード・リストを解放します。
4-114 ページ「XmlDomGetNodeListItem」	リスト内の n 番目のノードを戻します。
4-114 ページ「XmlDomGetNodeListLength」	ノード・リストの長さを戻します。

XmlDomFreeNodeList

XmlDomGetElemsByTag または関連するファンクションによって戻されたノード・リストを解放して、対応するすべてのリソースを解放します。DOM 固有のノード・リストの一部（ノードの子など）を指定すると、処理は行われません。

構文

```
void XmlDomFreeNodeList(
   xmlctx *xctx,
   xmlnodelist *list)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
list	IN	解放する NodeList。

関連項目： XmlDomGetElemsByTag、XmlDomGetElemsByTagNS、XmlDomGetChildrenByTag、XmlDomGetChildrenByTagNS

XmlDomGetNodeListItem

ノード・リストの n 番目のノードを戻します。最初の項目は、索引 0 です。

構文

```
xmlnode* XmlDomGetNodeListItem(
   xmlctx *xctx,
   xmlnodelist *list,
   ub4 index)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
list	IN	NodeList
index	IN	list への索引。

戻り値

(xmlnode *) ノード・リスト内の n 番目の位置にあるノード（または NULL）。

関連項目： XmlDomGetNodeListLength、XmlDomFreeNodeList

XmlDomGetNodeListLength

ノード・リスト内のノードの数（長さ）を戻します。ノードは索引によって参照されるため、有効な索引の範囲は 0（ゼロ）から length-1 です。

構文

```
ub4 XmlDomGetNodeListLength(
   xmlctx *xctx,
   xmlnodelist *list)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
list	IN	NodeList

戻り値

(ub4) ノード・リスト内のノードの数。

関連項目： XmlDomGetNodeListItem、XmlDomFreeNodeList

Notation インタフェース

表 4-10 に、Notation インタフェースを介して使用できるメソッドを示します。

表 4-10 NodeList メソッドの概要 : DOM パッケージ

ファンクション	概要
4-116 ページ「XmlDomGetNotationPubID」	表記法の公開識別子を取得します。
4-117 ページ「XmlDomGetNotationSysID」	表記法のシステム識別子を取得します。

XmlDomGetNotationPubID

（データ・エンコーディングの）表記法の公開識別子を戻します。ノードが表記法でないか、または定義済の公開識別子を持っていない場合は NULL を戻します。

構文

```
oratext* XmlDomGetNotationPubID(
   xmlctx *xctx,
   xmlnotenode *note)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
note	IN	Notation ノード。

戻り値

(oratext *) 表記法の公開識別子（データ・エンコーディング。NULL の可能性があります)。

関連項目： XmlDomGetNotationSysID

XmlDomGetNotationSysID

（データ・エンコーディングの）表記法のシステム識別子を戻します。ノードが表記法でないか、または定義済のシステム識別子を持っていない場合は NULL を戻します。

構文

```
oratext* XmlDomGetNotationSysID(
   xmlctx *xctx,
   xmlnotenode *note)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
note	IN	Notation ノード。

戻り値

(oratext *) 表記法のシステム識別子（データ・エンコーディング。NULL の可能性があります)。

関連項目： XmlDomGetNotationPubID

ProcessingInstruction インタフェース

表 4-11 に、ProcessingInstruction インタフェースを介して使用できるメソッドを示します。

表 4-11 ProcessingInstruction メソッドの概要 : DOM パッケージ

ファンクション	概要
4-118 ページ「XmlDomGetPIData」	処理命令のデータを取得します。
4-119 ページ「XmlDomGetPITarget」	PI のターゲットを取得します。
4-119 ページ「XmlDomSetPIData」	処理命令のデータを設定します。

XmlDomGetPIData

（データ・エンコーディングの）処理命令のコンテンツ（データ）を戻します。ノードが ProcessingInstruction ではない場合は、NULL を戻します。コンテンツは、ターゲットの後の最初の空白以外の文字から最後の "?>" までの部分です。

構文

```
oratext* XmlDomGetPIData(
   xmlctx *xctx,
   xmlpinode *pi)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
pi	IN	ProcessingInstruction ノード。

戻り値

(oratext *) 処理命令のデータ（データ・エンコーディング）。

関連項目： XmlDomGetPITarget、XmlDomSetPIData

XmlDomGetPITarget

処理命令のターゲット文字列を戻します。ノードが ProcessingInstruction ではない場合は、NULL を戻します。ターゲットは、ProcessingInstruction を開始するマークアップの後に続く最初のトークンです。データ部分はオプションですが、すべての ProcessingInstruction がターゲットを持つ必要があります。

構文

```
oratext* XmlDomGetPITarget(
   xmlctx *xctx,
   xmlpinode *pi)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
pi	IN	ProcessingInstruction ノード。

戻り値

(oratext *) 処理命令のターゲット（データ・エンコーディング）。

関連項目： XmlDomGetPIData、XmlDomSetPIData

XmlDomSetPIData

ProcessingInstruction のコンテンツ（データ・エンコーディング内にある必要があります）を設定します。データを NULL に設定することはできません。ノードが ProcessingInstruction ではない場合、処理は行われません。新しいデータは、検証、変換またはチェックされません。

構文

```
void XmlDomSetPIData(
   xmlctx *xctx,
   xmlpinode *pi,
   oratext *data)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
pi	IN	ProcessingInstruction ノード。
data	IN	ProcessingInstruction の新しいデータ（データ・エンコーディング）。

関連項目： XmlDomGetPITarget、XmlDomGetPIData

Text インタフェース

表 4-12 に、Text インタフェースを介して使用できるメソッドを示します。

表 4-12 Text メソッドの概要 : DOM パッケージ

ファンクション	概要
4-121 ページ「XmlDomSplitText」	テキスト・ノードを 2 つに分割します。

XmlDomSplitText

1 つのテキスト・ノードを 2 つに分割します。元のデータは、2 つのノードに分割されます。指定されたノードのタイプがテキストではないか、または offset が元のデータの範囲外にある場合は、処理を行わずに NULL を戻します。offset は 0（ゼロ）から始まり、バイト数ではなく文字数です。元のノードは保持され、データのみが切り捨てられます。元のデータの残りを含む新しいテキスト・ノードが作成され、元のノードの直後の兄弟関係として挿入されます。新しいテキスト・ノードが戻されます。

構文

```
xmltextnode* XmlDomSplitText(
   xmlctx *xctx,
   xmltextnode *textnode,
   ub4 offset)
```

パラメータ	In/Out	説明
xctx	IN	XML コンテキスト。
textnode	IN	Text ノード
offset	IN	0 テキストが分割される 0（ゼロ）ベースの文字カウント。

戻り値

(xmltextnode *) 新しいテキスト・ノード。

関連項目： XmlDomGetCharData、XmlDomAppendData、XmlDomInsertData、XmlDomDeleteData、XmlDomReplaceData

5

C 用の範囲 API パッケージ

範囲パッケージには、2 つのインタフェースの API が含まれます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

DocumentRange インタフェース

表 5-1 に DocumentRange インタフェースで使用できるメソッドの概要を示します。

表 5-1 DocumentRange メソッドの概要 : 範囲パッケージ

ファンクション	概要
5-2 ページ「XmlDomCreateRange」	範囲オブジェクトを作成します。

XmlDomCreateRange

範囲オブジェクトの作成に使用される、唯一の DocumentRange インタフェースのメソッド。

構文

```
xmlrange* XmlDomCreateRange(
   xmlctx *xctx,
   xmlrange *range,
   xmldocnode *doc);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	既存の NodeIterator、または新規 NodeIterator を割り当てる場合は NULL
doc	IN	新規の範囲が追加されるドキュメント

戻り値

(xmlrange *) 元の、または新規の範囲オブジェクト。

範囲インタフェース

表 5-2 に範囲インタフェースを通じて使用できるメソッドの概要を示します。

表 5-2 範囲メソッドの概要 : 範囲パッケージ

ファンクション	概要
5-4 ページ「XmlDomRangeClone」	範囲を複製します。
5-4 ページ「XmlDomRangeCloneContents」	範囲で選択された内容を複製します。
5-5 ページ「XmlDomRangeCollapse」	範囲をスタート・ポイントかエンド・ポイントまで縮小します。
5-6 ページ「XmlDomRangeCompareBoundaryPoints」	2 つの範囲の境界点を比べます。
5-7 ページ「XmlDomRangeDeleteContents」	範囲で選択された内容を削除します。
5-7 ページ「XmlDomRangeDetach」	範囲を分離します。
5-8 ページ「XmlDomRangeExtractContents」	範囲で選択された内容を抽出します。
5-8 ページ「XmlDomRangeGetCollapsed」	範囲が縮小されているかどうかを戻します。
5-9 ページ「XmlDomRangeGetCommonAncestor」	2 つの境界点の最も深い共通の祖先ノードを戻します。
5-9 ページ「XmlDomRangeGetDetached」	範囲が分離されているかどうかを戻します。
5-10 ページ「XmlDomRangeGetEndContainer」	範囲終了コンテナ・ノードを戻します。
5-10 ページ「XmlDomRangeGetEndOffset」	範囲終了オフセットを戻します。
5-11 ページ「XmlDomRangeGetStartContainer」	範囲開始コンテナ・ノードを戻します。
5-11 ページ「XmlDomRangeGetStartOffset」	範囲開始オフセットを戻します。
5-12 ページ「XmlDomRangeIsConsistent」	範囲が一貫しているかどうかを戻します。
5-12 ページ「XmlDomRangeSelectNode」	ノードを範囲として選択します。
5-13 ページ「XmlDomRangeSelectNodeContents」	ノード内容を選択するための範囲を定義します。
5-14 ページ「XmlDomRangeSetEnd」	エンド・ポイントを設定します。
5-14 ページ「XmlDomRangeSetEndBefore」	ノードの前にエンド・ポイントを設定します。

表 5-2 範囲メソッドの概要 : 範囲パッケージ（続き）

ファンクション	概要
5-15 ページ「XmlDomRangeSetStart」	スタート・ポイントを設定します。
5-16 ページ「XmlDomRangeSetStartAfter」	ノードの後にスタート・ポイントを設定します。
5-16 ページ「XmlDomRangeSetStartBefore」	ノードの前にスタート・ポイントを設定します。

XmlDomRangeClone

範囲を複製します。元の範囲で選択された内容に影響することなく、範囲を複製します。エラーの場合は NULL を戻します。

構文

```
xmlrange* XmlDomRangeClone(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(xmlrange *) 古い範囲を複製した新しい範囲。

XmlDomRangeCloneContents

範囲で選択された内容を複製します。複製しますが、範囲で選択された内容を削除しません。範囲の一貫性チェックを実行し、エラーがあれば retval をエラー・コードに設定します。

構文

```
xmlnode* XmlDomRangeCloneContents(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(xmlnode *) 複製された内容。

XmlDomRangeCollapse

範囲をスタート・ポイントかエンド・ポイントまで縮小します。縮小のポイントは、この範囲が追加されるドキュメント内の有効なポイントとみなされます。

構文

```
xmlerr XmlDomRangeCollapse(
   xmlctx *xctx,
   xmlrange *range,
   boolean tostart);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
tostart	IN	スタート・ポイント（TRUE）かエンド・ポイント（FALSE）のどちらまで縮小するか指示

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeCompareBoundaryPoints

2 つの異なる範囲の 2 つの境界点を比較します。範囲（range）の対応する境界点が 2 番目の範囲（srange）の対応する境界点の前、同じ位置、後のいずれに位置するかにより -1、0、1 を戻します。2 つの範囲が 2 つの異なるドキュメントに追加される場合、またはいずれかの範囲が分離される場合は ~(int)0 を戻します。

構文

```
sb4 XmlDomRangeCompareBoundaryPoints(
   xmlctx *xctx,
   xmlrange *range,
   xmlcmphow how,
   xmlrange *srange,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
how	IN	xmlcmphow 値。比較方法
srange	IN	比較対象の範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(sb4) strcmp のような比較結果。

XmlDomRangeDeleteContents

範囲で選択された内容を削除します。範囲の一貫性チェックを実行し、エラーがあれば retval をエラー・コードに設定します。

構文

```
xmlerr XmlDomRangeDeleteContents(
   xmlctx *xctx,
   xmlrange *range);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeDetach

範囲をドキュメントから分離させ、その範囲（range）を無効な状態にします。

構文

```
xmlerr XmlDomRangeDetach(
   xmlctx *xctx,
   xmlrange *range);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeExtractContents

範囲で選択された内容を抽出します。範囲で選択された内容を複製し、削除します。範囲の一貫性チェックを実行し、エラーがあれば retval をエラー・コードに設定します。

構文

```
xmlnode* XmlDomRangeExtractContents(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(xmlnode *) 抽出された内容。

XmlDomRangeGetCollapsed

範囲が縮小され、分離されない場合は TRUE を、それ以外の場合は FALSE を戻します。

構文

```
boolean XmlDomRangeGetCollapsed(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(boolean) 範囲が縮小されている場合は TRUE、それ以外の場合は FALSE。

XmlDomRangeGetCommonAncestor

範囲が分離されていない場合、範囲の 2 つの境界点で最も深い共通の祖先ノードを戻し、それ以外の場合は NULL を戻します。範囲は一貫した状態にあると仮定されます。

構文

```
xmlnode* XmlDomRangeGetCommonAncestor(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(xmlnode *) 最も深い共通の祖先ノード（または NULL)。

XmlDomRangeGetDetached

範囲が分離されているかどうかを戻します。範囲が分離され NULL 以外の場合は、TRUE を戻します。それ以外の場合は、FALSE を戻します。

構文

```
ub1 XmlDomRangeGetDetached(
   xmlctx *xctx,
   xmlrange *range);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト

戻り値

(ub1) 範囲が分離されている場合は TRUE、それ以外の場合は FALSE。

XmlDomRangeGetEndContainer

範囲が分離されていない場合は、範囲終了コンテナ・ノードを戻し、それ以外の場合は NULL を戻します。

構文

```
xmlnode* XmlDomRangeGetEndContainer(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(xmlnode *) 範囲終了コンテナ・ノード（または NULL)。

XmlDomRangeGetEndOffset

範囲が分離されていない場合、範囲終了オフセットを戻し、それ以外の場合は ~(ub4)0（最大値 ub4）を戻します。

構文

```
ub4 XmlDomRangeGetEndOffset(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(ub4) 範囲終了オフセット（または最大 ub4)。

XmlDomRangeGetStartContainer

範囲が有効な状態で分離されていない場合は、範囲開始コンテナ・ノードを戻し、それ以外の場合は NULL を戻します。

構文

```
xmlnode* XmlDomRangeGetStartContainer(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(xmlnode *) 範囲開始コンテナ・ノード。

XmlDomRangeGetStartOffset

範囲が分離されていない場合、範囲開始オフセットを戻し、それ以外の場合は ~(ub4)0（最大値 ub4）を戻します。

構文

```
ub4 XmlDomRangeGetStartOffset(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(ub4) 範囲開始オフセット（または最大 ub4）。

XmlDomRangeIsConsistent

範囲が一貫しているかどうかを戻します。範囲が一貫している場合は TRUE を戻します。両方のポイントは同じルート内にあり、スタート・ポイントはエンド・ポイントの前か同位置にあります。それ以外の場合は、FALSE を戻します。

構文

```
boolean XmlDomRangeIsConsistent(
   xmlctx *xctx,
   xmlrange *range,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
xerr	OUT	数値のリターン・コード

戻り値

(ub1) 範囲が一貫している場合は TRUE、それ以外の場合は FALSE。

XmlDomRangeSelectNode

このノードの親がコンテナ・ノードになり、その子の間で、オフセットがこのノードのオフセットと等しくなるように範囲のエンド・ポイントとスタート・ポイントを設定します。範囲は縮小されます。ノードはそのドキュメントの有効なノードであると仮定されます。範囲が分離されている場合、ノードは無視され、範囲が追加されます。

構文

```
xmlerr XmlDomRangeSelectNode(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeSelectNodeContents

範囲スタート・ポイントをノード内容の先頭に、範囲エンド・ポイントをノード内容の末尾に設定します。ノードは有効なドキュメント・ノードであると仮定されます。範囲が分離されている場合、ノードは無視され、範囲が追加されます。

構文

```
xmlerr XmlDomRangeSelectNodeContents(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeSetEnd

範囲のエンド・ポイントを設定します。現在のルート・コンテナ以外のルート・コンテナが範囲にある場合、範囲は新しい位置まで縮小されます。エンド・ポイントがスタート・ポイントの前に位置するように設定されている場合、範囲はその位置に縮小されます。このタイプが定義される説明に従って、xmlerr 値を戻します。範囲のスタート・ポイントが有効なスタート・ポイントであると仮定します。

構文

```
xmlerr XmlDomRangeSetEnd(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node,
   ub4 offset);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード
offset	IN	終了オフセット

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeSetEndBefore

範囲のエンド・ポイントをノードの前に設定します。現在のルート・コンテナ以外のルート・コンテナが範囲にある場合、範囲は新しい位置まで縮小されます。ビフォア・ノードによりエンド・ポイントがスタート・ポイントの前に位置するように設定されている場合、その範囲は新しい位置まで縮小されます。このタイプが定義される説明に従って、xmlerr 値を戻します。範囲のスタート・ポイントが有効なスタート・ポイントであると仮定します。

構文

```
xmlerr XmlDomRangeSetEndBefore(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeSetStart

範囲のスタート・ポイントを設定します。現在のルート・コンテナ以外のルート・コンテナが範囲にある場合、範囲は新しい位置まで縮小されます。スタート・ポイントがエンド・ポイントの後に位置するように設定されている場合、範囲はその位置まで縮小されます。このタイプが定義される説明に従って、xmlerr 値を戻します。範囲のエンド・ポイントが有効なエンド・ポイントであると仮定します。

構文

```
xmlerr XmlDomRangeSetStart(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node,
   ub4 offset);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード
offset	IN	開始オフセット

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeSetStartAfter

範囲のスタート・ポイントをノードの後に設定します。現在のルート・コンテナ以外のルート・コンテナが範囲にある場合、範囲は新しい位置まで縮小されます。アフター・ノードによりスタート・ポイントがエンド・ポイントの後に位置するように設定されている場合、その範囲は新しい位置まで縮小されます。このタイプが定義される説明に従って、xmlerr 値を戻します。範囲のエンド・ポイントが有効なエンド・ポイントであると仮定します。

構文

```
xmlerr XmlDomRangeSetStartAfter(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード

戻り値

(xmlerr) 数値のリターン・コード。

XmlDomRangeSetStartBefore

範囲のスタート・ポイントをノードの前に設定します。現在のルート・コンテナ以外のルート・コンテナが範囲にある場合、範囲はオフセットを 0 として新しい位置まで縮小されます。ビフォア・ノードによりスタート・ポイントがエンド・ポイントの後に位置するように設定されている場合、その範囲は新しい位置まで縮小されます。このタイプが定義される説明に従って、xmlerr 値を戻します。範囲のエンド・ポイントが有効なエンド・ポイントであると仮定します。

構文

```
xmlerr XmlDomRangeSetStartBefore(
   xmlctx *xctx,
   xmlrange *range,
   xmlnode *node);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
range	IN	範囲オブジェクト
node	IN	XML ノード

戻り値

(xmlerr) 数値のリターン・コード。

6

C 用の SAX API パッケージ

SAX はイベントベースの XML 解析用の標準インタフェースで、XML-DEV メーリング・リストのメンバーが共同開発したものです。SAX を使用するには、ファンクション・ポインタを使用して `xmlsaxcb` 構造を初期化し、`XmlLoadSax` コールのいずれかに渡します。ユーザー定義のコンテキスト構造に対するポインタを含めることもできます。そのコンテキスト・ポインタは、各 SAX ファンクションに渡されます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

SAX インタフェース

表 6-1 に SAX インタフェースを通じて使用できるメソッドの概要を示します。

表 6-1 SAX メソッドの概要

ファンクション	概要
6-3 ページ「XmlSaxAttributeDecl」	属性の宣言の SAX 通知を受け取ります。Oracle の拡張機能。
6-4 ページ「XmlSaxCDATA」	CDATA の SAX 通知を受け取ります。Oracle の拡張機能。
6-5 ページ「XmlSaxCharacters」	文字データの SAX 通知を受け取ります。
6-6 ページ「XmlSaxComment」	コメントの SAX 通知を受け取ります。
6-6 ページ「XmlSaxElementDecl」	要素の宣言の SAX 通知を受け取ります。Oracle の拡張機能。
6-7 ページ「XmlSaxEndDocument」	ドキュメントの終わりの SAX 通知を受け取ります。
6-7 ページ「XmlSaxEndElement」	要素の終わりの SAX 通知を受け取ります。
6-8 ページ「XmlSaxNotationDecl」	表記法の宣言の SAX 通知を受け取ります。
6-8 ページ「XmlSaxPI」	処理命令の SAX 通知を受け取ります。
6-9 ページ「XmlSaxParsedEntityDecl」	解析対象エンティティの宣言の SAX 通知を受け取ります。Oracle の拡張機能。
6-10 ページ「XmlSaxStartDocument」	ドキュメントの始まりの SAX 通知を受け取ります。
6-10 ページ「XmlSaxStartElement」	要素の始まりの SAX 通知を受け取ります。
6-11 ページ「XmlSaxStartElementNS」	名前空間を認識する要素の始まりの SAX 通知を受け取ります。
6-12 ページ「XmlSaxUnparsedEntityDecl」	解析対象外のエンティティ宣言の SAX 通知を受け取ります。
6-13 ページ「XmlSaxWhitespace」	無視できる（空白）データの SAX 通知を受け取ります。
6-14 ページ「XmlSaxXmlDecl」	XML 宣言の SAX 通知を受け取ります。Oracle の拡張機能。

XmlSaxAttributeDecl

このイベントは DTD の要素宣言をマークします。要素名と内容はデータ・エンコーディングに含まれます。属性は所属する要素の前に宣言できることに注意してください。

構文

```
xmlerr XmlSaxAttributeDecl(
   void *ctx,
   oratext *elem,
   oratext *attr,
   oratext *body)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
elem	IN	属性が宣言される要素。データ・エンコーディング
attr	IN	属性名。データ・エンコーディング
body	IN	属性宣言の本体

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxAttributeDecl

XmlSaxCDATA

このイベントでは CDATA は Text と異なるものとして処理されます。XmlSaxCDATA コールバックが指定されない場合、Text コールバックが呼び出されます。データはデータ・エンコーディングに含まれ、戻される長さはバイトではなく文字数です。無視できる（空白形式の）文字データの通知を受け取る、XmlSaxWhitespace も参照してください。

構文

```
xmlerr XmlSaxCDATA(
   void *ctx,
   oratext *ch,
   size_t len)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
ch	IN	CDATA のポインタ、データ・エンコーディング
len	IN	CDATA の長さ、文字数

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxWhitespace

XmlSaxCharacters

このイベントは Text か CDATA のいずれかの文字データをマークします。XmlSaxCDATA コールバックが指定されている場合、CDATA はそのコールバックに送られます。XmlSaxCDATA コールバックが指定されていない場合、Text と CDATA はいずれも XmlSaxCharacters コールバックに送られます。データはデータ・エンコーディングに含まれ、戻される長さはバイトではなく文字数です。無視できる（空白形式の）文字データの通知を受け取る、XmlSaxWhitespace も参照してください。

構文

```
xmlerr XmlSaxCharacters(
   void *ctx,
   oratext *ch,
   size_t len)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
ch	IN	データのポインタ、データ・エンコーディング
len	IN	データの長さ、文字数

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxWhitespace

XmlSaxComment

このイベントは XML ドキュメントのコメントをマークします。コメントのデータはデータ・エンコーディングに含まれます。SAX 標準ではなく、Oracle の拡張機能です。

構文

```
xmlerr XmlSaxComment(
   void *ctx,
   oratext *data)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
data	IN	コメントのデータ、データ・エンコーディング

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

XmlSaxElementDecl

このイベントは DTD の要素宣言をマークします。要素の名前と内容はデータ・エンコーディングに含まれます。

構文

```
xmlerr XmlSaxElementDecl(
   void *ctx,
   oratext *name,
   oratext *content)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
name	IN	要素名
content	IN	要素のコンテキスト・モデル

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxAttributeDecl

XmlSaxEndDocument

最後の SAX イベント。ドキュメントごとに 1 回コールされ、ドキュメントの終わりを示します。対応するイベントは XmlSaxStartDocument です。

構文

```
xmlerr XmlSaxEndDocument(
   void *ctx)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxStartDocument

XmlSaxEndElement

このイベントは要素の終わりをマークします。XmlSaxStartElement イベントまたは XmlSaxStartElementNS イベントに対応します。名前は要素の tagName（名前空間を認識した要素の場合は修飾名になる場合があります）であり、データ・エンコーディングに含まれます。

構文

```
xmlerr XmlSaxEndElement(
   void *ctx,
   oratext *name)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
name	IN	要素の終わりの名前、データ・エンコーディング

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxEndElement

XmlSaxNotationDecl

このイベントは DTD の表記の宣言をマークします。表記法名、公開識別子、およびシステム識別子はデータ・エンコーディングに含まれます。識別子はいずれもオプションで、NULL でもかまいません。

構文

```
xmlerr XmlSaxNotationDecl(
   void *ctx,
   oratext *name,
   oratext *pubId,
   oratext *sysId)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
name	IN	表記法名。データ・エンコーディング
pubId	IN	データ・エンコーディング時の表記の公開識別子、または NULL
sysId	IN	データ・エンコーディング時の表記法のシステム識別子、または NULL

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

XmlSaxPI

このイベントは ProcessingInstruction をマークします。ProcessingInstruction のターゲットとデータはデータ・エンコーディングに含まれます。ターゲットは必須ですが、データは NULL でもかまいません。

構文

```
xmlerr XmlSaxPI(
   void *ctx,
   oratext *target,
   oratext *data)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
target	IN	PI のターゲット、データ・エンコーディング
data	IN	データ・エンコーディング時の PI のデータ、または NULL

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

XmlSaxParsedEntityDecl

DTD の解析対象エンティティの宣言をマークします。解析対象エンティティの名前、公開識別子、システム識別子および表記法名はデータ・エンコーディングに含まれます。

構文

```
xmlerr XmlSaxParsedEntityDecl(
   void *ctx,
   oratext *name,
   oratext *value,
   oratext *pubId,
   oratext *sysId,
   boolean general)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
name	IN	エンティティ名。データ・エンコーディング
value	IN	エンティティの値。データ・エンコーディング
pubId	IN	データ・エンコーディング時のエンティティの公開識別子、または NULL
sysId	IN	エンティティのシステム識別子。データ・エンコーディング
general	IN	汎用エンティティの場合は TRUE、パラメータ・エンティティの場合は FALSE

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxUnparsedEntityDecl

XmlSaxStartDocument

最初の SAX イベント。ドキュメントごとに 1 回コールされ、ドキュメントの始まりを示します。対応するイベントは XmlSaxEndDocument です。

構文

```
xmlerr XmlSaxStartDocument(
   void *ctx)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxEndDocument

XmlSaxStartElement

このイベントは要素の始まりをマークします。これは元の名前空間を認識しないバージョンの SAX 1 であることに注意してください。XmlSaxStartElementNS が名前空間を認識するバージョンの SAX 2 です。両方が登録されている場合、NS バージョンのみがコールされます。要素名はすべての属性パートと同様に、データ・エンコーディングに含まれます。属性のマッピングの操作については、NamedNodeMap インタフェースのファンクションを参照してください。対応するファンクションは XmlSaxEndElement です（このファンクションの名前空間を認識するバージョンはありません）。

構文

```
xmlerr XmlSaxStartElement(
   void *ctx,
   oratext *name,
   xmlnodelist *attrs)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
name	IN	要素名。データ・エンコーディング
attrs	IN	要素の属性の NamedNodeMap

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxEndElement、第 4 章「C 用の DOM API パッケージ」の XmlDomGetNodeMapLength、および第 4 章「C 用の DOM API パッケージ」の XmlDomGetNamedItem

XmlSaxStartElementNS

このイベントは要素の始まりをマークします。名前空間を認識するバージョンの新しい SAX 2 であることに注意してください。XmlSaxStartElement は名前空間を認識しないバージョンの SAX 1 です。両方が登録されている場合、NS バージョンのみがコールされます。要素の修飾名、ローカル名、名前空間 URI はすべての属性パートと同様に、データ・エンコーディングに含まれます。属性マップの操作については、NamedNodeMap インタフェースのファンクションを参照してください。対応するファンクションは XmlSaxEndElement です（このファンクションの名前空間を認識するバージョンはありません）。

構文

```
xmlerr XmlSaxStartElementNS(
   void *ctx,
   oratext *qname,
   oratext *local,
   oratext *nsp,
   xmlnodelist *attrs)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
qname	IN	要素の修飾名。データ・エンコーディング
local	IN	要素の名前空間のローカル名。データ・エンコーディング
nsp	IN	要素の名前空間 URI。データ・エンコーディング
attrs	IN	要素の属性の NodeList、または NULL

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxStartElement、XmlSaxEndElement、第 4 章「C 用の DOM API パッケージ」の XmlDomGetNodeMapLength、および第 4 章「C 用の DOM API パッケージ」の XmlDomGetNamedItem

XmlSaxUnparsedEntityDecl

DTD の解析対象外のエンティティ宣言をマークします。解析対象エンティティについては、XmlSaxParsedEntityDecl を参照してください。解析対象外のエンティティ名、公開識別子、システム識別子および表記法名はデータ・エンコーディングに含まれます。

構文

```
xmlerr XmlSaxUnparsedEntityDecl(
   void *ctx,
   oratext *name,
   oratext *pubId,
   oratext *sysId,
   oratext *note)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
name	IN	エンティティ名。データ・エンコーディング
pubId	IN	データ・エンコーディングに含まれたエンティティの公開識別子、または NULL
sysId	IN	エンティティのシステム識別子。データ・エンコーディング
note	IN	エンティティの表記法名。データ・エンコーディング

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxParsedEntityDecl

XmlSaxWhitespace

このイベントは改行、行間のインデントなどの無視できる空白データをマークします。対応するファンクションは、通常の文字データの通知を受け取る XmlSaxCharacters です。データはデータ・エンコーディングに含まれ、戻される長さはバイトではなく文字数です。

構文

```
xmlerr XmlSaxWhitespace(
   void *ctx,
   oratext *ch,
   size_t len)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
ch	IN	データのポインタ、データ・エンコーディング
len	IN	データの長さ、文字数

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

関連項目： XmlSaxCharacters

XmlSaxXmlDecl

このイベントは XML 宣言をマークします。XmlSaxStartDocument は常に最初のイベントです。このコールバックが登録され XMLDecl が存在する場合、XmlSaxXmlDecl が 2 番目のイベントになります。エンコーディング・フラグにより、エンコーディングが指定されているかどうかが示されます。ドキュメントのデフォルトのエンコーディング指定はオーバーライドされる（あるいは間違っている）場合があり、またなんらかの方式で入力がデータ・エンコーディングに変換されるため、ドキュメントで指定される実際のエンコーディングは指定されません。スタンドアロン・フラグの場合、フラグが指定されていない場合は -1 が戻されます。それ以外の場合、FALSE は 0、TRUE は 1 が戻されます。

構文

```
xmlerr XmlSaxXmlDecl(
   void *ctx,
   oratext *version,
   boolean encoding,
   sword standalone)
```

パラメータ	IN/OUT	説明
ctx	IN	ユーザーの SAX コンテキスト
version	IN	XMLDecl のバージョン文字列。データ・エンコーディング
encoding	IN	エンコーディングが指定されているかどうかを指定
standalone	IN	スタンドアロン・ドキュメントの値。指定されない場合は < 0

戻り値

(xmlerr) エラー・コード、成功の場合は XMLERR_OK [0]。

7

C 用のスキーマ API パッケージ

この XML スキーマ・バリデータの C 実装は、W3C の XML Schema 仕様、Rev REC-xmlschema-1-20010502 に準拠します。これにより、スキーマ・バリデータが複数のスキーマ・ドキュメントを 1 つのスキーマに編集するのに必要な動作が実装されます。この結果のスキーマを使用して、特定のインスタンス・ドキュメントを検証することができます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

スキーマ・インタフェース

表 7-1 に Schema インタフェースで使用できるメソッドを示します。

表 7-1 スキーマ・メソッドの概要

ファンクション	概要
7-3 ページ「XmlSchemaClean」	スキーマ・コンテキストにロードされたスキーマをクリーン・アップし、スキーマ・コンテキストを再利用します。
7-3 ページ「XmlSchemaCreate」	スキーマ・コンテキストを作成し、戻します。
7-4 ページ「XmlSchemaDestroy」	スキーマ・コンテキストを破棄します。
7-4 ページ「XmlSchemaErrorWhere」	エラーが発生した場所を戻します。
7-5 ページ「XmlSchemaLoad」	スキーマ・ドキュメントをロードします。
7-5 ページ「XmlSchemaLoadedList」	ロードされたスキーマ・ドキュメントのサイズとリスト、またはいずれかを戻します。
7-6 ページ「XmlSchemaSetErrorHandler」	スキーマ・コンテキストに、エラー・メッセージ・ハンドラとその関連コンテキストを設定します。
7-7 ページ「XmlSchemaSetValidateOptions」	次の検証セッションで使用されるオプションを設定します。
7-7 ページ「XmlSchemaTargetNamespace」	指定されたスキーマ・ドキュメントのターゲットの名前空間を戻します。
7-8 ページ「XmlSchemaUnload」	スキーマ・ドキュメントをアンロードします。
7-9 ページ「XmlSchemaValidate」	要素ノードをスキーマについて検証します。
7-10 ページ「XmlSchemaVersion」	このスキーマ実装のバージョンを戻します。

XmlSchemaClean

スキーマ・コンテキストにロードされたスキーマをクリーン・アップし、スキーマ・コンテキストを再利用します。

構文

```
void XmlSchemaClean(
   xsdctx *sctx);
```

パラメータ	IN/OUT	説明
sctx	IN	クリーン・アップされるスキーマ・コンテキスト

関連項目： XmlSchemaCreate、XmlSchemaDestroy

XmlSchemaCreate

その他のバリデータ API で使用されるスキーマ・コンテキストを戻します。これは XmlSchemaDestroy と合わせて使用する必要があります。

構文

```
xsdctx *XmlSchemaCreate(
   xmlctx *xctx,
   xmlerr *err,
   list);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
err	OUT	戻されるエラー・コード
list	IN	可変引数の NULL で終了するリスト

戻り値

(xsdctx *) スキーマ・コンテキスト。

関連項目： 第 9 章「C 用の XML API パッケージ」の XmlSchemaDestroy、XmlCreate

XmlSchemaDestroy

スキーマ・コンテキストを破棄し、そのすべてのリソースを解放します。

構文

```
void XmlSchemaDestroy(
   xsdctx *sctx);
```

パラメータ	IN/OUT	説明
sctx	IN	解放されるスキーマ・コンテキスト

関連項目： XmlSchemaCreate

XmlSchemaErrorWhere

エラーが発生した場所（行番号、パス）を戻します。

構文

```
xmlerr XmlSchemaErrorWhere(
   xsdctx *sctx,
   ub4 *line,
   oratext **path);
```

パラメータ	IN/OUT	説明
sctx	IN	スキーマ・コンテキスト
line	IN/OUT	エラーが発生した行番号
path	IN/OUT	エラーが発生した URL またはファイル領域

戻り値

(xmlerr) エラー・コード。

関連項目： XmlSchemaSetErrorHandler

XmlSchemaLoad

次の検証セッションで使用されるスキーマ・ドキュメントをロードします。ロードされるスキーマ・ドキュメントが有効である間は、スキーマ・ドキュメントをスキーマに追加的にロードできます。最後にロードされたスキーマが無効と判断された場合は、XmlSchemaClean をコールしてスキーマ・コンテキストをクリーン・アップし、適切に修正した最新のスキーマを含むすべてのコンテキストを再ロードする必要があります。

構文

```
xmlerr XmlSchemaLoad(
   xsdctx *sctx,
   oratext *uri,
   list);
```

パラメータ	IN/OUT	説明
sxctx	IN	スキーマ・コンテキスト
uri	IN	スキーマ・ドキュメントの URL。コンパイラのエンコーディング
list	IN	可変引数の NULL で終了するリスト

戻り値

(xmlerr) 数値のエラー・コード、成功時には XMLERR_OK[0] を戻します。

関連項目： XmlSchemaUnload、XmlSchemaLoadedList

XmlSchemaLoadedList

list が NULL の場合、ロードされたスキーマ・ドキュメントのサイズのみを戻します。list が NULL 以外の場合、ユーザー側のポインタ・バッファで URL ポインタのリストが戻されます。十分な大きさのバッファを用意するのは、ユーザー側の責任であることに注意してください。

構文

```
ub4 XmlSchemaLoadedList(
   xsdctx *sctx,
   oratext **list);
```

パラメータ	IN/OUT	説明
sctx	IN	スキーマ・コンテキスト
list	IN	ポインタ・バッファのアドレス

戻り値

(ub4) リスト・サイズ。

関連項目： XmlSchemaLoad、XmlSchemaUnload

XmlSchemaSetErrorHandler

スキーマ・コンテキストに、エラー・メッセージ・ハンドラとその関連コンテキストを設定します。エラーに関する有益な位置情報を取り出すために、エラー・ハンドラ・コンテキストにスキーマ・コンテキストのアドレスを指定する必要があります。

構文

```
xmlerr XmlSchemaSetErrorHandler(
   xsdctx *sctx,
   XML_ERRMSG_F(
      (*errhdl),
      ectx,
      msg,
      err),
   void *errctx);
```

パラメータ	IN/OUT	説明
sctx	IN	スキーマ・コンテキスト
errhdl	IN	エラー・メッセージ・ハンドラ
errctx	IN	エラー・ハンドラのコンテキスト

戻り値

(xmlerr) エラー・コード。

関連項目： XmlSchemaCreate、XmlSchemaErrorWhere、および第 3 章「C 用のコールバック API パッケージ」の XML_ERRMSG_F

XmlSchemaSetValidateOptions

次の検証セッションで使用されるオプションを設定します。以前に設定されたオプションは、上書きまたはリセットされるまで有効です。

構文

```
xmlerr XmlSchemaSetValidateOptions(
   xsdctx *sctx,
   list);
```

パラメータ	IN/OUT	説明
sctx	IN	スキーマ・コンテキスト
list	IN	可変引数の NULL で終了するリスト

戻り値

(xmlerr) 数値のエラー・コード、成功時には XMLERR_OK[0] を戻します。

関連項目： XmlSchemaValidate

XmlSchemaTargetNamespace

URI で識別され、指定されたスキーマ・ドキュメントのターゲットの名前空間を戻します。現在ロードされているすべてのスキーマ・ドキュメントは問い合わせできます。現在ロードされているスキーマ・ドキュメントには、XmlSchemaLoad でロードされたドキュメント、および schemaLocation ヒントまたは noNamespaceSchemaLocation ヒントでロードされたドキュメントが含まれます。

構文

```
oratext *XmlSchemaTargetNamespace(
   xsdctx *sctx,
   oratext *uri);
```

パラメータ	IN/OUT	説明
sctx	IN	XML コンテキスト
uri	IN	問合せるスキーマ・ドキュメントの URL

戻り値

(oratext *) ターゲットの名前空間文字列。ドキュメントが指定されていない場合は NULL。

関連項目： XmlSchemaLoadedList

XmlSchemaUnload

バリデータからスキーマ・ドキュメントをアンロードします。以前にロードされたすべてのスキーマ・ドキュメントは、アンロードされるまでロードされた状態です。ロードされたすべてのスキーマ・ドキュメントをアンロードするには、URI が NULL になるように設定します（XmlSchemaClean と同じ意味）。指定のスキーマに関連する子スキーマも、すべてアンロードされることに注意してください。この実装では、次の場面のみがサポートされます。

- ロード、ロード、…
- ロード、ロード、ロード、アンロード、アンロード、アンロード、クリーン、これを反復。

ロード、ロード、アンロード、ロード、…はサポートされません。

構文

```
xmlerr XmlSchemaUnload(
   xsdctx *sctx,
   oratext *uri,
   list);
```

パラメータ	IN/OUT	説明
sctx	IN	スキーマ・コンテキスト
uri	IN	スキーマ・ドキュメントの URL。コンパイラのエンコーディング
list	IN	可変引数の NULL で終了するリスト

戻り値

(xmlerr) 数値のエラー・コード、成功時には XMLERR_OK[0] を戻します。

関連項目： XmlSchemaLoad、XmlSchemaLoadedList

XmlSchemaValidate

要素ノードをスキーマについて検証します。現在のセッションで使用されるスキーマには、XmlSchemaLoad で指定されたスキーマ・ドキュメント、およびインスタンス・ドキュメントで schemaLocation または noNamespaceSchemaLocation でヒントとして指定されたスキーマ・ドキュメントがすべて含まれます。このルーチンが呼び出された後、ロードされたすべてのスキーマ・ドキュメントはロードされた状態が続き、XmlSchemaLoadedList を使用して問い合せできます。ただし、これらは引き続き非アクティブな状態です。次の検証セッションでは、非アクティブなスキーマ・ドキュメントをアクティブにできます。それには XmlSchemaLoad でスキーマ・ドキュメントを指定するか、または新しいインスタンス・ドキュメントの schemaLocation または noNamespaceSchemaLocation で、それらをヒントとして指定します。スキーマ・ドキュメントとそのすべての子孫（ネスト化方式で追加またはインポートされたドキュメント）をアンロードする場合は、XmlSchemaUnload を使用します。

構文

```
xmlerr XmlSchemaValidate(
   xsdctx *sctx,
   xmlctx *xctx,
   xmlelemnode *elem);
```

パラメータ	IN/OUT	説明
sctx	IN	スキーマ・コンテキスト
xctx	IN	最上位の XML コンテキスト。
elem	IN	ドキュメント内の検証される要素ノード

戻り値

(xmlerr) 数値のエラー・コード、成功時には XMLERR_OK[0] を戻します。

関連項目： XmlSchemaSetValidateOptions

XmlSchemaVersion

このスキーマ実装のバージョンを戻します。

構文

```
oratext *XmlSchemaVersion();
```

戻り値

(oratext *) バージョン文字列（コンパイラのエンコーディング）。

8

C 用の横断 API パッケージ

横断パッケージには 4 つのインタフェースの API が含まれます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

DocumentTraversal インタフェース

表 8-1 に DocumentTraversal インタフェースで使用できるメソッドの概要を示します。

表 8-1 DocumentTraversal メソッドの概要 : 横断パッケージ

ファンクション	概要
8-2 ページ「XmlDomCreateNodeIter」	ノードのイテレータ・オブジェクトを作成します。
8-3 ページ「XmlDomCreateTreeWalker」	ツリー・ウォーカ・オブジェクトを作成します。

XmlDomCreateNodeIter

DocumentTraversal インタフェースの 2 つあるメソッドの 1 つであり、NodeIterator オブジェクトの作成に使用されます。このメソッドは、タイプを除いては戻されるオブジェクト XmlDomCreateTreeWalker と同一です。

whatToShow 引数はフラグ・ビットのマスクであり、各ノード・タイプに 1 つずつ割り当てられます。値 XMLDOM_SHOW_ALL では、すべてのノード・タイプが渡されますが、それ以外の場合はビットが設定されているタイプのみが渡されます。

Entity 参照拡張機能は、entrefExpansion フラグにより制御されます。TRUE の場合、実態参照は最終内容と置き換えられます。FALSE の場合、実態参照はノードとして残されます。

構文

```
xmliter* XmlDomCreateNodeIter(
   xmlctx *xctx,
   xmliter *iter,
   xmlnode *root,
   xmlshowbits whatToShow,
   XMLDOM_ACCEPT_NODE_F(
      (*nodeFilter),
      xctx,
      node),
   boolean entrefExpand);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
iter	IN	設定する場合は既存の NodeIterator、作成する場合は NULL
xerr	IN	NodeIterator のルート・ノード
whatToShow	IN	XMLDOM_SHOW_*XXX* フラグ・ビットのマスク
nodeFilter	IN	使用されるノード・フィルタ、ノード・フィルタがない場合は NULL
xerr	IN	実態参照ノードを拡張するかどうかを指定

戻り値

(xmliter *) 元の、または新規の NodeIterator オブジェクト。

関連項目： XmlDomCreateTreeWalker

XmlDomCreateTreeWalker

DocumentTraversal インタフェースの 2 つのメソッドのうち 1 つで、TreeWalker オブジェクトの作成に使用されます。このメソッドは、タイプを除いては戻されるオブジェクト XmlDomCreateNodeIter と同一です。

whatToShow 引数はフラグ・ビットのマスクであり、各モード・タイプに 1 つずつ割り当てられます。値 XMLDOM_SHOW_ALL では、すべてのノード・タイプが渡されます。それ以外の場合はビットが設定されているタイプのみが渡されます。

Entity 参照拡張機能は、entrefExpansion フラグにより制御されます。TRUE の場合、実態参照は最終内容と置き換えられます。FALSE の場合、実態参照はノードとして残されます。

構文

```
xmlwalk* XmlDomCreateTreeWalker(
   xmlctx *xctx,
   xmlwalk* walker,
   xmlnode *root,
   xmlshowbits whatToShow,
   XMLDOM_ACCEPT_NODE_F(
      (*nodeFilter),
      xctx,
```

```
    node),
  boolean entrefExpansion);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	設定する場合は既存の TreeWalker、作成する場合は NULL
xerr	IN	TreeWalker のルート・ノード
whatToShow	IN	XMLDOM_SHOW_*XXX* フラグ・ビットのマスク
nodeFilter	IN	使用されるノード・フィルタ、ノード・フィルタがない場合は NULL
xerr	IN	実態参照ノードを拡張するかどうかを指定

戻り値

(xmlwalk *) 新規 TreeWalker オブジェクト。

関連項目： XmlDomCreateNodeIter

NodeFilter インタフェース

表 8-2 に NodeFilter インタフェースで使用できるメソッドの概要を示します。

表 8-2 NodeFilter メソッドの概要 : 横断パッケージ

ファンクション	概要
8-5 ページ「XMLDOM_ACCEPT_NODE_F」	ユーザー定義のフィルタリング・アクションをノードで実行します。

XMLDOM_ACCEPT_NODE_F

NodeFilter インタフェースの唯一のメソッド。ノードとフィルタが指定された場合、実行するフィルタリング・アクションを決定します。

このファンクション・ポインタは、必要に応じてノード・イテレータ / ツリー・ウォーカ・メソッドに渡されます。

xmlerr の値は次のとおりです。

- XMLERR_OK ノードを受け取ります。NodeIterator または TreeWalker に定義されたナビゲーション・メソッドはこのノードを戻します。
- XMLERR_FILTER_REJECT ノードを拒否します。NodeIterator または TreeWalker に定義されたナビゲーション・メソッドはこのノードを戻しません。TreeWalker の場合、このノードの子も拒否されます。NodeIterator はこれを XMLDOM_FILTER_SKIP のシノニムとして処理します。
- XMLERR_FILTER_SKIP この単一ノードをスキップします。NodeIterator または TreeWalker に定義されたナビゲーション・メソッドはこのノードを戻しません。NodeIterator と TreeWalker のいずれの場合も、このノードの子は考慮されます。

構文

```
#define XMLDOM_ACCEPT_NODE_F(func, xctx, node)
xmlerr func(
   xmlctx *xctx,
   xmlnode *node)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
node	IN	テストするノード

戻り値

(xmlerr) フィルタリング結果。

NodeIterator インタフェース

表 8-3 に NodeIterator インタフェースで使用できるメソッドの概要を示します。

表 8-3 NodeIterator メソッドの概要 : 横断パッケージ

ファンクション	概要
8-7 ページ「XmlDomIterDetach」	ノード・イテレータを分離します（停止します）。
8-8 ページ「XmlDomIterNextNode」	イテレータの次のノードを戻します。
8-9 ページ「XmlDomIterPrevNode」	イテレータの前のノードを戻します。

XmlDomIterDetach

反復しているセットから NodeIterator を分離して、すべてのリソースを解放し、イテレータを INVALID 状態に移行させます。分離が呼び出された後、XmlDomIterNextNode または XmlDomIterPrevNode をコールすると例外 XMLERR_ITER_DETACHED が発生します。

構文

```
xmlerr XmlDomIterDetach(
   xmlctx *xctx,
   xmliter *iter);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
iter	IN	ノード・イテレータ・オブジェクト

関連項目： XmlDomIterNextNode、XmlDomIterPrevNode

XmlDomIterNextNode

セットの次のノードを戻し、セットのイテレータの位置を進めます。ノード・イテレータが作成された後、最初に XmlDomIterNextNode をコールするとセットの最初のノードが戻されます。参照ノード（現在のイテレータの位置）は削除されないものと想定されます。それ以外の場合、基本の DOM ツリーに変更を加えてもイテレータは無効になりません。

構文

```
xmlnode* XmlDomIterNextNode(
   xmlctx *xctx,
   xmliter *iter,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
iter	IN	ノード・イテレータ・オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 反復しているセットの次のノード（または NULL）。

関連項目： XmlDomIterPrevNode、XmlDomIterDetach

XmlDomIterPrevNode

セットの前のノードを戻し、セットのイテレータの位置を後方に移動します。

構文

```
xmlnode* XmlDomIterPrevNode(
   xmlctx *xctx,
   xmliter *iter,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
iter	IN	ノード・イテレータ・オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 反復しているセットの前のノード（または NULL）。

関連項目： XmlDomIterNextNode、XmlDomIterDetach

TreeWalker インタフェース

表 8-4 に TreeWalker インタフェースで使用できるメソッドの概要を示します。

表 8-4 TreeWalker メソッドの概要 : 横断パッケージ

ファンクション	概要
8-11 ページ「XmlDomWalkerFirstChild」	現在のノードで最初に参照できる子を戻します。
8-11 ページ「XmlDomWalkerGetCurrentNode」	現在のノードを戻します。
8-12 ページ「XmlDomWalkerGetRoot」	ルート・ノードを戻します。
8-13 ページ「XmlDomWalkerLastChild」	現在のノードで最後に参照される子を戻します。
8-13 ページ「XmlDomWalkerNextNode」	次に参照できるノードを戻します。
8-14 ページ「XmlDomWalkerNextSibling」	直後の兄弟関係ノードを戻します。
8-15 ページ「XmlDomWalkerParentNode」	親ノードを戻します。
8-15 ページ「XmlDomWalkerPrevNode」	前のノードを戻します。
8-16 ページ「XmlDomWalkerPrevSibling」	直前の兄弟関係ノードを戻します。
8-17 ページ「XmlDomWalkerSetCurrentNode」	現在のノードを設定します。
8-17 ページ「XmlDomWalkerSetRoot」	ルート・ノードを設定します。

XmlDomWalkerFirstChild

現在のノードで最初に参照できる子に TreeWalker を移動し、新しいノードを戻します。現在のノードに参照可能な子が存在しない場合、NULL を戻し、現在のノードを維持します。

構文

```
xmlnode* XmlDomWalkerFirstChild(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 最初に参照できる子（または NULL)。

関連項目： XmlDomWalkerLastChild

XmlDomWalkerGetCurrentNode

現在のノードを戻します（取得します)。エラーの場合は NULL を戻します。

構文

```
xmlnode* XmlDomWalkerGetCurrentNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 現在のノード。

XmlDomWalkerGetRoot

ルート・ノードを戻します（取得します）。エラーの場合は NULL を戻します。現在のノードは、ノードが属するサブツリーと一緒にルート・ノード下から削除できるため、ウォーカ内の現在のルート・ノードでは現在のノードとの関連性が失われる場合があります。
TreeWalker の反復は現在のノードに基づきます。ただしルート・ノードは反復の空間を定義します。このファンクションは、ルート・ノード（祖先）が現在のノードとまだ関連性があるかどうかをチェックします。関連性がある場合、このルート・ノードを戻します。それ以外の場合、現在のルートが属するツリーのルートを検出し、このルートをウォーカのルート・ノードとして設定し、戻します。ウォーカが NULL ポインタの場合、NULL を戻します。

構文

```
xmlnode* XmlDomWalkerGetRoot(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) ルート・ノード。

XmlDomWalkerLastChild

現在のノードで最後に参照される子に TreeWalker を移動し、新しいノードを戻します。現在のノードに参照可能な子が存在しない場合、NULL を戻し、現在のノードを維持します。

構文

```
xmlnode* XmlDomWalkerLastChild(
   xmlctx *xctx,
   xmlwalk *walker,
    xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 最後に参照される子（または NULL）。

XmlDomWalkerNextNode

TreeWalker を、現在のノードの次に参照可能なノードにドキュメント順に移動し、新しいノードを戻します。現在のノードに次のノードが存在しない場合、または次のノードの検索で、TreeWalker のルート・ノードより上の検索が試行された場合は、NULL を戻し、現在のノードを維持します。

構文

```
xmlnode* XmlDomWalkerNextNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 次のノード（または NULL）。

関連項目： XmlDomWalkerPrevNode、XmlDomWalkerNextSibling、XmlDomWalkerPrevSibling

XmlDomWalkerNextSibling

現在のノードの直後の兄弟関係に TreeWalker を移動し、新しいノードを戻します。現在のノードに参照可能な直後の兄弟関係が存在しない場合、NULL を戻し、現在のノードを維持します。

構文

```
xmlnode* XmlDomWalkerNextSibling(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 直後の兄弟関係（または NULL）。

関連項目： XmlDomWalkerNextNode、XmlDomWalkerPrevNode、XmlDomWalkerPrevSibling

XmlDomWalkerParentNode

現在のノードで参照可能な最も近い祖先ノードに移動し、その祖先モードを戻します。親ノードの検索で TreeWalker のルート・ノードよりも上の検索が試行された場合、または参照可能な祖先モードが見つからなかった場合、このメソッドは現在の位置を維持し、NULL を戻します。

構文

```
xmlnode* XmlDomWalkerParentNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 親ノード（または NULL)。

XmlDomWalkerPrevNode

TreeWalker を、現在のノードの前の参照可能なノードにドキュメント順に移動し、新しいノードを戻します。現在のノードに前のノードが存在しない場合、または前のノード検索で、TreeWalker のルート・ノードより上の検索が試行された場合、NULL を戻し、現在のノードを維持します。

構文

```
xmlnode* XmlDomWalkerPrevNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 前のノード（または NULL）。

関連項目： XmlDomWalkerNextNode、XmlDomWalkerNextSibling、XmlDomWalkerPrevSibling

XmlDomWalkerPrevSibling

現在のノードの直前の兄弟関係に TreeWalker を移動し、新しいノードを戻します。現在のノードに参照可能な直前の兄弟関係が存在しない場合、NULL を戻し、現在のノードを維持します。

構文

```
xmlnode* XmlDomWalkerPrevSibling(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 直前の兄弟関係（または NULL）。

関連項目： XmlDomWalkerNextNode、XmlDomWalkerPrevNode、XmlDomWalkerNextSibling

XmlDomWalkerSetCurrentNode

新しい現在のノードを設定し、このノードを戻します。またルート・ノードが新しい現在のノードの祖先かどうかをチェックします。祖先ではない場合、現在のノードを設定せず、NULL を戻して、retval を XMLDOM_WALKER_BAD_NEW_CUR に設定します。エラーの場合は NULL を戻します。

構文

```
xmlnode* XmlDomWalkerSetCurrentNode(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlnode *node,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
node	IN	新しい現在のノード
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 新しい現在のノード。

XmlDomWalkerSetRoot

ルート・ノードを設定します。新しいルート・ノードが現在のノードの祖先である場合は、このノードを戻します。祖先でない場合はエラーを送り、現在のルート・ノードが現在のノードの祖先かどうかをチェックします。祖先の場合、現在のノードを戻します。それ以外の場合、現在のノードが所属するツリーのルートにルート・ノードを設定し、そのルートを戻します。ウォーカまたはルート・ノード・パラメータが NULL ポインタの場合、NULL を戻します。

構文

```
xmlnode* XmlDomWalkerSetRoot(
   xmlctx *xctx,
   xmlwalk *walker,
   xmlnode *node,
   xmlerr *xerr);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト
walker	IN	TreeWalker オブジェクト
node	IN	新しいルート・ノード
xerr	OUT	数値のリターン・エラー・コード

戻り値

(xmlnode *) 新しいルート・ノード。

9

C 用の XML API パッケージ

この XML プロセッサ（またはパーサー）の C 実装は、W3C の XML 仕様（改訂 REC-xml-19980210）に準拠しており、XML データを読み込むために XML プロセッサに必要な動作、およびアプリケーションに提供する必要がある情報を含んでいます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XML インタフェース

表 9-1 に、XML インタフェースで使用できるメソッドの概要を示します。

表 9-1 XML のメソッドの概要

ファンクション	概要
9-3 ページ「XmlAccess」	URL のアクセス・メソッド・コールバックを設定します。
9-4 ページ「XmlCreate」	XML Developer's Toolkit の xmlctx を作成します。
9-6 ページ「XmlCreateDTD」	DTD を作成します。
9-7 ページ「XmlCreateDocument」	ドキュメント（ノード）を作成します。
9-7 ページ「XmlDestroy」	xmlctx を破棄します。
9-8 ページ「XmlFreeDocument」	ドキュメントを解放します（すべてのリソースを解放します）。
9-8 ページ「XmlGetEncoding」	XML コンテキストで使用されているデータのエンコーディングを戻します。
9-9 ページ「XmlHasFeature」	DOM 機能が実装されているかどうかを判別します。
9-10 ページ「XmlIsSimple」	シングルバイト（単純な）キャラクタ・セット・フラグを戻します。
9-10 ページ「XmlIsUnicode」	XmlIsUnicode（単純な）キャラクタ・セット・フラグを戻します。
9-11 ページ「XmlLoadDom」	XML 文書をロード（解析）して DOM を生成します。
9-13 ページ「XmlLoadSax」	XML 文書をロード（解析）して SAX イベントを生成します。
9-13 ページ「XmlLoadSaxVA」	XML 文書をロード（解析）して SAX イベント [varargs] を生成します。
9-14 ページ「XmlSaveDom」	XML 文書を保存（シリアル化、フォーマット）します。
9-15 ページ「XmlVersion」	XDK のバージョン文字列を戻します。

XmlAccess

特定の URL アクセス・メソッドのデータをロードするために使用するオープン / 読込み / クローズ・コールバックを設定します。HTTP、FTP などのための組込みデータ・ロード・ファンクションをオーバーライドするか、または新しい型（UNKNOWN など）を処理するためのファンクションを提供します。

構文

```
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));
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
access	IN	URL アクセス・メソッド。
userctx	IN	コールバックに渡されるユーザー定義のコンテキスト。
openf	IN	アクセスをオープンするコールバック・ファンクション。
readf	IN	アクセスを読み込むコールバック・ファンクション。
closef	IN	アクセスをクローズするコールバック・ファンクション。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

関連項目： XmlLoadDom、XmlLoadSax

XmlCreate

XML Developer's Toolkit の xmlctx を作成します。xmlctx の XDK および XMLType の両方に共通のプロパティは次のとおりです。

- ("data_encoding", データ・エンコーディングの名前) DOM および SAX によって XML データの表示に使用されるエンコーディング。指定されていない場合、デフォルトは UTF-8（または、EBCDIC プラットフォームの場合、UTF-E）です。EBCDIC または ISO-8859 などのシングルバイト・エンコーディングの方が、UTF-8 のようなマルチバイト・エンコーディングよりも高速です。Unicode（UTF-16）は、より多くのメモリーを消費しますが、マルチバイトよりもパフォーマンスが優れています。
- ("data_lid", データ・エンコーディング言語識別子) NLS の lx_langid として指定されているデータ・エンコーディング。対応する NLS グローバル地域も指定する必要があります。
- ("default_input_encoding", デフォルトの入力エンコーディングの名前) 入力ドキュメントのエンコーディングが BOM、XMLDecl、プロトコル・ヘッダーなどによって自動的に判別できない場合、このエンコーディングが使用されます。
- ("default_input_lid", デフォルトの入力エンコーディング言語識別子) NLS の lx_langid として指定されているデフォルトの入力エンコーディング。対応する NLS グローバル地域も指定する必要があります。
- ("error_language", エラー言語または language.encoding) エラー・メッセージの作成に使用される言語（およびオプションでエンコーディング）。デフォルトは UTF-8 エンコーディングを使用した「American」です。言語のみを指定する場合、言語名のみ指定し、他は指定しません（"American"）。エンコーディングも指定するには、ドットに続いてエンコーディングの Oracle 名称を追加します（"American.WE8ISO8859P1"）。
- ("error_handler", ファンクション・ポインタ、XML_ERRMSG_F を参照) エラー時のデフォルトの動作では、フォーマット済のメッセージが stderr に出力されます。エラー・ハンドラを提供すると、フォーマット済のメッセージは出力するかわりにハンドラに渡されます。
- ("error_context", エラー・ハンドラのユーザー定義によるコンテキスト) エラー・ハンドラ・ファンクションに渡されるコンテキスト・ポインタです。意味はユーザーが定義します。ここでは指定するのみで、エラー発生時に渡されます。

- ("input_encoding", 強制入力エンコーディングの名前) 入力ドキュメントの強制入力エンコーディング。ドキュメントの XMLDecl などをオーバーライドし、必ず指定したエンコーディングで解析するために使用します。**この機能の使用はお薦めしません。** BOM、XMLDecl などが存在する場合、これらは正確であるため、通常の場合は必要ありません。
- ("input_lid", INLID, POINTER) 強制入力エンコーディング。
- ("lpu_context", lpu context) URL データのロードおよびアクセス・メソッドのフックに使用される LPU コンテキスト。指定しない場合は作成されます。
- ("lml_context", LMLCTX, POINTER) 低レベルのメモリー割当てに使用される LML コンテキスト。指定しない場合は作成されます。外部からは、エンドユーザーは memory_alloc、memory_free などを設定する必要があります。
- ("memory_alloc", 低レベルのメモリー割当てファンクション) 低レベルのメモリー割当てファンクション（malloc を使用しない場合)。これを指定する場合、対応する解放ファンクションも指定する必要があります。XML_ALLOC_F を参照してください。
- ("memory_free", 低レベルのメモリー解放ファンクション) 低レベルのメモリー解放ファンクション（free を使用しない場合)。alloc ファンクションに対応します。
- ("memory_context", ユーザー定義のメモリー・コンテキスト) alloc および free ファンクションに渡されるユーザー定義のメモリー・コンテキスト。定義および用途はユーザーが決定できます。ここでは単に設定し、コールバックに渡されます。
- ("nls_global_area", NLS グローバル地域 , lx_glo) NLS の言語識別子としてエンコーディングが指定されている場合、対応する NLS グローバル地域も指定する必要があります。
- XDK には、XDK 型の xmlctx にのみ適用される独自のプロパティが存在します（前述のプロパティはすべて一般的なもので、すべての xmlctx に適用されます)。
- ("input_buffer_size", 文字単位で示した入力バッファのサイズ) 基本的な I/O バッファ・サイズ。デフォルトは 256K、最小値は 4K で最大値は 4MB です。エンコーディングにより、これらのバッファが 1 つ、2 つまたは 3 つ必要になります。サイズはバイト数ではなく文字数です。バッファが Unicode データを保持する場合、サイズは倍の大きさになります。
- ("memory_block_size", バイト単位で示したメモリー割当て単位のサイズ) 高レベルのメモリー・パッケージが低レベルのアロケータから要求するチャンク・サイズ。つまり、メモリー割当ての基本単位です。デフォルトは 64K、最小値は 16K で最大値は 256MB です。

構文

```
xmlctx *XmlCreate(
   xmlerr *err,
   oratext *name,
   list);
```

パラメータ	IN/OUT	説明
err	OUT	戻されたエラー・コード。
access	IN	コンテキスト名（デバッグ目的）。
list	IN	変数引数の NULL で終了するリスト。

戻り値

(xmlctx *) 作成された xmlctx。エラー発生時には err が設定された状態で NULL が戻されます。

関連項目： 第 3 章「C 用のコールバック API パッケージ」の XmlDestroy、XML_ERRMSG_F

XmlCreateDTD

DTD を作成します。

構文

```
xmldocnode* XmlCreateDTD(
   xmlctx *xctx
   oratext *qname,
   oratext *pubid,
   oratext *sysid,
   xmlerr *err)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
qname	IN	修飾名。
pubid	IN	外部サブセットの公開識別子。
sysid	IN	外部サブセットのシステム識別子。
err	OUT	戻されたエラー・コード。

戻り値

(xmldtdnode *) 新しい DTD ノード。

XmlCreateDocument

最初のトップレベルの DOCUMENT ノードおよびサポートするインフラストラクチャを作成します。修飾名を指定した場合、その名前の要素が作成され、ドキュメントのルート要素に設定されます。

構文

```
xmldocnode* XmlCreateDocument(
   xmlctx *xctx,
   oratext *uri,
   oratext *qname,
   xmldtdnode *dtd,
   xmlerr *err)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
uri	IN	作成するルート要素の名前空間の URI、または NULL。
qname	IN	ルート要素の修飾名、または存在しない場合は NULL。
dtd	IN	関連付けられた DTD ノード。
err	OUT	戻されたエラー・コード。

戻り値

(xmldocnode *) 新しい Document オブジェクト。

XmlDestroy

xmlctx を破棄します。

構文

```
void XmlDestroy(
   xmlctx *xctx)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。

関連項目： XmlCreate

XmlFreeDocument

XmlCreateDocument または Load ファンクションによって作成されたドキュメントを破棄します。さらに、ドキュメントに関連付けられている、無効となったすべてのリソースを解放します。

構文

```
void XmlFreeDocument(
   xmlctx *xctx,
   xmldocnode *doc)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
doc	IN	解放するドキュメント。

関連項目： XmlCreateDocument、XmlLoadDom

XmlGetEncoding

XML コンテキストで使用されているデータのエンコーディングを戻します。通常、データのエンコーディングはユーザーが選択するため、このファンクションは必要ありません。ただし、データ・エンコーディングが指定されておらず、デフォルトが使用されている場合、このファンクションを使用して、デフォルトのエンコーディング名を戻すことができます。

構文

```
oratext *XmlGetEncoding(
   xmlctx *xctx);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。

戻り値

(oratext *) データ・エンコーディングの名前。

関連項目： 第 4 章「C 用の DOM API パッケージ」の XmlDomGetDecl、XmlIsSimple、XmlIsUnicode

XmlHasFeature

DOM 機能が実装されているかどうかを判別します。指定されたバージョンで機能が実装されている場合は TRUE、実装されていない場合は FALSE を戻します。

レベル 1 の場合、パッケージの正当な値は「HTML」および「XML」です（大 / 小文字は区別されません）。バージョンは文字列「1.0」です。バージョンが指定されていない場合は、この機能のすべてのバージョンがサポートされるため、TRUE を戻します。

- DOM 1.0 の機能は「XML」および「HTML」です。
- DOM 2.0 の機能は「Core」、「XML」、「HTML」、「Views」、「StyleSheets」、「CSS」、「CSS2」、「Events」、「UIEvents」、「MouseEvents」、「MutationEvents」、「HTMLEvents」、「Range」、「Traversal」です。

構文

```
boolean XmlHasFeature(
   xmlctx *xctx,
   oratext *feature,
   oratext *version)
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
feature	IN	テストする機能のパッケージ名。
version	IN	テストするパッケージ名のバージョン番号。

戻り値

(boolean) 機能が実装されているかどうか。

XmlIsSimple

コンテキストのデータ・エンコーディングが単純であるかどうか、つまり、ASCII または EBCDIC などのように各文字がシングルバイトであるかどうかを示すフラグを戻します。

構文

```
boolean XmlIsSimple(
   xmlctx *xctx);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。

戻り値

(boolean) データ・エンコーディングが単純な場合は TRUE、そうでない場合は FALSE。

関連項目： XmlGetEncoding、XmlIsUnicode

XmlIsUnicode

コンテキストのデータ・エンコーディングが Unicode、つまり UTF-16 で、各文字が 2 バイトであるかどうかを示すフラグを戻します。

構文

```
boolean XmlIsUnicode(
   xmlctx *xctx);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。

戻り値

(boolean) データ・エンコーディングが Unicode の場合は TRUE、そうでない場合は FALSE。

関連項目： XmlGetEncoding、XmlIsSimple

XmlLoadDom

入力ソースから XML 文書をロード（解析）して DOM を生成します。正常に終了した場合はルート・ドキュメント・ノード、エラーが発生した場合は（err を設定した状態で）NULL を戻します。

このファンクションは、2 つの固定引数の xmlctx とエラー・リターン・コード、0 個以上の (プロパティ , 値) ペア、および NULL を取ります。

SOURCE: 入力ソースは、次のいずれかの相互に排他的なプロパティの 1 つによって設定されます（1 つを選択します）。

- ("uri", ドキュメントの URI) [コンパイラのエンコーディング]
- ("file", ドキュメントのファイル・システム・パス) [コンパイラのエンコーディング]
- ("buffer", バッファのアドレス , "buffer_length", バッファ内のバイト数)
- ("stream", ストリーム・オブジェクトのアドレス , "stream_context", ストリーム・オブジェクトのコンテキストへのポインタ)
- ("stdio", FILE* ストリーム)

PROPERTIES: 追加プロパティは次のとおりです。

- ("dtd", DTD ノード) ドキュメントの DTD。
- ("base_uri", ドキュメントのベース URI) URI 以外のソースからロードされたドキュメントの場合に、有効なベース URI を設定します。ドキュメントのベース URI は、相対 URI の include、import などの解決に必要です。
- ("input_encoding", エンコーディング名) 強制入力エンコーディング [名前]
- ("default_input_encoding", encoding_name) ドキュメントが自己記述的でない場合（BOM、プロトコル・ヘッダー、XMLDecl などがない場合）に使用するデフォルトの入力エンコーディング。
- ("schema_location", string) ドキュメントのスキーマの schemaLocation。ドキュメントをデータベースにロードする際に最適のレイアウトを判別するために使用されます。
- ("validate", ブール値) TRUE の場合、DTD 検証を有効にします。デフォルトで、整形式のみ確認されます。スキーマの検証は別に扱われます。
- ("discard_whitespace", ブール値) TRUE の場合、入力ドキュメントの要素間の書式化用の空白（改行およびインデント）は削除されます。デフォルトでは、すべての入力文字が維持されます。
- ("dtd_only", ブール値) TRUE の場合、完全なドキュメントではなく、外部 DTD を解析します。

- ("stop_on_warning", ブール値) TRUE の場合、警告はエラーと同様に扱われ、解析、検証などが即時停止されます。デフォルトでは、警告が発行されても処理は継続されます。
- ("warn_duplicate_entity", ブール値) TRUE の場合、エンティティが複数回宣言されると、警告が発行されます。デフォルトでは、最初の宣言が受け入れられ、その後の宣言は暗黙的に無視されます。
- ("no_expand_char_ref", ブール値) TRUE の場合、文字参照は DOM データ内で拡張されません。通常、文字参照は表現する文字によって置換されます。ただし、ドキュメントが保存されると、それらの文字エンティティは再表示されません。ロードおよび保存の過程でこれらを保持するには、拡張しないようにします。
- ("no_check_chars", ブール値) TRUE の場合、XML [2] Char production のテストを省略し、すべての入力文字が有効として受け入れられます。

構文

```
xmldocnode *XmlLoadDom(
   xmlctx *xctx,
   xmlerr *err,
   list);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
err	OUT	戻されたエラー・コード。
list	IN	変数引数の NULL で終了するリスト。

戻り値

(xmlctx *) 成功時はドキュメント・ノードが戻されます。失敗時には、err が設定された状態で NULL が戻されます。

関連項目： XmlSaveDom

XmlLoadSax

入力ソースから XML 文書をロード（解析）し、SAX イベントのセットを（ユーザー・コールバックとして）生成します。入力ソースおよびプロパティの基本セットは、XmlLoadDom と同じです。

構文

```
xmlerr XmlLoadSax(
   xmlctx *xctx,
   xmlsaxcb *saxcb,
   void *saxctx,
   list);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
saxcb	IN	SAX コールバック構造。
saxctx	IN	SAX コールバックに渡されるコンテキスト。
list	IN	変数引数の NULL で終了するリスト。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

XmlLoadSaxVA

入力ソースから XML 文書をロード（解析）し、SAX イベントのセットを（ユーザー・コールバックとして）生成します。入力ソースおよびプロパティの基本セットは、XmlLoadDom と同じです。

構文

```
xmlerr XmlLoadSaxVA(
   xmlctx *xctx,
   xmlsaxcb *saxcb,
   void *saxctx,
   va_list va);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
saxcb	IN	SAX コールバック構造。
saxctx	IN	SAX コールバックに渡されるコンテキスト。
va	IN	変数引数の NULL で終了するリスト。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

XmlSaveDom

ドキュメントまたはサブツリーを指定された宛先にシリアル化し、書き込まれたバイト数を戻します。宛先が指定されていない場合、フォーマット後のサイズのみ戻し、出力は戻しません。

出力エンコーディングが指定されている場合、ドキュメントは出力時に再エンコードされます。そうでない場合、既存のエンコーディングのままです。

最上位はインデントされたステップ×レベルの空白、次のレベルはステップ×（レベル＋ 1）の空白で、以下同様です。

バッファに保存する際、バッファがオーバーフローすると 0（ゼロ）が戻され、エラーが XMLERR_SAVE_OVERFLOW に設定されます。

DESTINATION: 出力先は、次のいずれかの相互に排他的なプロパティの 1 つによって設定されます（1 つを選択します）。

- ("uri", ドキュメントの URI) POST、PUT? [コンパイラのエンコーディング]
- （"file", ドキュメントのファイル・システム・パス) [コンパイラのエンコーディング]
- ("buffer", バッファのアドレス , "buffer_length", バッファ内のバイト数)
- ("stream", ストリーム・オブジェクトのアドレス , "stream_context", ストリーム・オブジェクトのコンテキストへのポインタ)

PROPERTIES: 追加プロパティは次のとおりです。

- ("output_encoding", エンコーディング名) ドキュメントの最終エンコーディングの名前。指定された場合を除き、xmlctx と同じエンコーディングが使用されます。
- ("indent_step", 符号なし) 出力の各レベルをインデントする空白数。デフォルトは 4 です。0（ゼロ）はインデントなしを示します。
- ("indent_level", 符号なし) 最初のインデント・レベル。デフォルトは 0（ゼロ）で、インデントなし、左揃えを示します。

- ("xmldecl", ブール値) 出力ドキュメントに XMLDecl を含めます。通常、XMLDecl は、完全なドキュメントの出力です（ルート・ノードは DOC)。
- ("bom", ブール値) 出力ドキュメントに BOM を入力します。通常、BOM は特定のエンコーディング（UTF-16）にのみ必要で、他のエンコーディング（UTF-8）の場合はオプションです。オプションの BOM を出力します。
- ("prune", ブール値) 出力を UNIX の 'find' コマンドのようにプルーニングします。子には派生せず、指定された 1 つのノードのみ出力します。

構文

```
ubig_ora XmlSaveDom(
   xmlctx *xctx,
   xmlerr *err,
   xmlnode *root,
   list);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。
err	OUT	失敗した場合、エラー・コード。
root	IN	保存するルート・ノードまたはサブツリー。
list	IN	変数引数の NULL で終了するリスト。

戻り値

(ubig_ora) 出力先に書き込まれたバイト数。

関連項目： XmlLoadDom

XmlVersion

XDK のバージョン文字列を戻します。

構文

```
oratext *XmlVersion();
```

戻り値

(oratext *) バージョン文字列。

10

C 用の XPath API パッケージ

XPath メソッドは、XPath に関連のある型およびインタフェースを処理します。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XPath インタフェース

表 10-1 に、XPath インタフェースで使用できるメソッドの概要を示します。

表 10-1 XPath のメソッドの概要

ファンクション	概要
10-3 ページ「XmlXPathCreateCtx」	XPath コンテキストを作成します。
10-3 ページ「XmlXPathDestroyCtx」	XPath コンテキストを破棄します。
10-4 ページ「XmlXPathEval」	XPath 式を評価します。
10-4 ページ「XmlXPathGetObjectBoolean」	XPath オブジェクトのブール値を取得します。
10-5 ページ「XmlXPathGetObjectFragment」	XPath オブジェクトのフラグメント値を取得します。
10-5 ページ「XmlXPathGetObjectNSetNode」	ノードセット型 XPath オブジェクトからノードを取得します。
10-6 ページ「XmlXPathGetObjectNSetNum」	ノードセット型 XPath オブジェクトからノードの数を取得します。
10-6 ページ「XmlXPathGetObjectNumber」	XPath オブジェクトから数値を取得します。
10-7 ページ「XmlXPathGetObjectString」	XPath オブジェクトから文字列を取得します。
10-7 ページ「XmlXPathGetObjectType」	XPath オブジェクト型を取得します。
10-8 ページ「XmlXPathParse」	XPath 式を解析します。

XmlXPathCreateCtx

XPath コンテキストを作成します。

構文

```
xpctx* XmlXPathCreateCtx(
   xmlctx *xsl,
   oratext *baseuri,
   xmlnode *ctxnode,
   ub4 ctxpos,
   ub4 ctxsize);
```

パラメータ	IN/OUT	説明
xsl	IN	xmldoc オブジェクトとしての XSL スタイルシート。
baseuri	IN	ドキュメントで使用されているベース URI（存在する場合）。
ctxnode	IN	現在のコンテキストの位置。
ctxpos	IN	現在のコンテキストのサイズ。
ctxsize	IN	現在のコンテキスト・ノード。

戻り値

（xpctx *）XPath コンテキスト。エラー発生時には NULL が戻されます。

XmlXPathDestroyCtx

XPath コンテキストを破棄します。

構文

```
void XmlXPathDestroyCtx(
   xpctx *xslxpctx);
```

パラメータ	IN/OUT	説明
xslxpctx	IN	XPath コンテキスト・オブジェクト。

XmlXPathEval

XPath 式を評価します。

構文

```
xpobj *XmlXPathEval(
   xpctx *xctx,
   xpexpr *exprtree,
   xmlerr *err);
```

パラメータ	IN/OUT	説明
xctx	IN	XPath コンテキスト。
exprtree	IN	解析した XPath 式ツリー。
err	OUT	エラー・コード。

戻り値

(xpobj *) 結果の XPath オブジェクト。エラー発生時には NULL が戻されます。

XmlXPathGetObjectBoolean

XPath オブジェクトのブール値を取得します。

構文

```
boolean XmlXPathGetObjectBoolean(
   xpobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。

戻り値

(boolean) 真理値。

関連項目： XmlXPathGetObjectType、XmlXPathGetObjectNSetNum、XmlXPathGetObjectNSetNode、XmlXPathGetObjectNumber、XmlXPathGetObjectBoolean

XmlXPathGetObjectFragment

XPath オブジェクトのブール値を取得します。

構文

```
xmlnode* XmlXPathGetObjectFragment(
   xpobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。

戻り値

(boolean) 真理値。

関連項目： XmlXPathGetObjectType、XmlXPathGetObjectNSetNum、XmlXPathGetObjectNSetNode、XmlXPathGetObjectNumber、XmlXPathGetObjectBoolean

XmlXPathGetObjectNSetNode

ノードセット型 XPath オブジェクトからノードを取得します。

構文

```
xmlnode *XmlXPathGetObjectNSetNode(
   xpobj *obj,
   ub4 i);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。
i	IN	ノードセットのノードのインデックス。

戻り値

(xmlnode *) オブジェクト型または値。

関連項目： XmlXPathGetObjectType、XmlXPathGetObjectNSetNum、XmlXPathGetObjectString、XmlXPathGetObjectNumber、XmlXPathGetObjectBoolean

XmlXPathGetObjectNSetNum

ノードセット型 XPath オブジェクトからノードの数を取得します。

構文

```
ub4 XmlXPathGetObjectNSetNum(
   xpobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。

戻り値

(ub4) ノードセット内のノードの数。

関連項目： XmlXPathGetObjectType、XmlXPathGetObjectNSetNode、XmlXPathGetObjectString、XmlXPathGetObjectNumber、XmlXPathGetObjectBoolean

XmlXPathGetObjectNumber

XPath オブジェクトから数値を取得します。

構文

```
double XmlXPathGetObjectNumber(
   xpobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。

戻り値

(double) 数値。

関連項目： XmlXPathGetObjectType、XmlXPathGetObjectNSetNum、XmlXPathGetObjectNSetNode、XmlXPathGetObjectString、XmlXPathGetObjectBoolean

XmlXPathGetObjectString

XPath オブジェクトから文字列を取得します。

構文

```
oratext *XmlXPathGetObjectString(
   xpobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。

戻り値

(oratext *) 文字列。

関連項目： XmlXPathGetObjectType、XmlXPathGetObjectNSetNum、XmlXPathGetObjectNSetNode、XmlXPathGetObjectNumber、XmlXPathGetObjectBoolean

XmlXPathGetObjectType

XPath オブジェクト型を取得します。

構文

```
xmlxslobjtype XmlXPathGetObjectType(
   xpobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	XPath オブジェクト。

戻り値

(xmlxslobjtype) オブジェクトの型コード。

関連項目： XmlXPathGetObjectNSetNum、XmlXPathGetObjectNSetNode、XmlXPathGetObjectString、XmlXPathGetObjectNumber、XmlXPathGetObjectBoolean

XmlXPathParse

XPath 式を解析します。

構文

```
xpexpr* XmlXPathParse(
   xpctx *xctx,
   oratext *expr,
   xmlerr * err);
```

パラメータ	IN/OUT	説明
xctx	IN	XPath コンテキスト・オブジェクト。
expr	IN	XPath 式。
err	OUT	エラー・コード。

戻り値

(xpexpr *) XPath 式解析ツリー。エラー発生時には NULL が戻されます。

11

C 用の XPointer API パッケージ

XPointer パッケージには、3 つのインタフェースの API が含まれています。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XPointer インタフェース

表 11-1 に、XPointer インタフェースで使用できるメソッドの概要を示します。

表 11-1 XPointer メソッドの概要 : XPointer パッケージ

ファンクション	概要
11-2 ページ「XmlXPointerEval」	xpointer 文字列を評価します。

XmlXPointerEval

xpointer 文字列を解析および評価し、ドキュメント内での位置を計算します。

構文

```
xmlxptrlocset* XmlXPointerEval(
   xmldocnode* doc,
   oratext* xptrstr);
```

パラメータ	IN/OUT	説明
doc	IN	対応する DOM ツリーのドキュメント・ノード。
xptrstr	IN	xpointer 文字列。

戻り値

(xmlxptrlocset *) 計算した位置セット。

XPtrLoc インタフェース

表 11-2 に、XPtrLoc インタフェースで使用できるメソッドの概要を示します。

表 11-2 XPtrLoc メソッドの概要 : XPointer パッケージ

ファンクション	概要
11-3 ページ「XmlXPtrLocGetNode」	XPtrLoc から Xml ノードを戻します。
11-4 ページ「XmlXPtrLocGetPoint」	XPtrLoc から Xml ポイントを戻します。
11-4 ページ「XmlXPtrLocGetRange」	XPtrLoc から Xml の範囲を戻します。
11-5 ページ「XmlXPtrLocGetType」	XPtrLoc の型を戻します。
11-5 ページ「XmlXPtrLocToString」	位置から文字列を戻します。

XmlXPtrLocGetNode

位置からノードを戻します。

構文

```
xmlnode* XmlXPtrLocGetNode(
   xmlxptrloc* loc);
```

パラメータ	IN/OUT	説明
loc	IN	位置。

戻り値

(xmlnode *) 位置の Node。

XmlXPtrLocGetPoint

位置からポイントを戻します。

構文

```
xmlpoint* XmlXPtrLocGetPoint(
   xmlxptrloc* loc);
```

パラメータ	IN/OUT	説明
loc	IN	位置。

戻り値

(xmlpoint *) 位置のポイント。

XmlXPtrLocGetRange

位置から範囲を戻します。

構文

```
xmlrange* XmlXPtrLocGetRange(
   xmlxptrloc* loc);
```

パラメータ	IN/OUT	説明
loc	IN	位置。

戻り値

(xmlrange *) 位置の範囲。

XmlXPtrLocGetType

位置の型を戻します。

構文

```
xmlxptrloctype XmlXPtrLocGetType(
   xmlxptrloc* loc);
```

パラメータ	IN/OUT	説明
loc	IN	位置。

戻り値

(xmlxptrloctype) 位置の型。

XmlXPtrLocToString

位置から文字列を戻します。

- ノード名 : コンテナ・ノードの名前。

構文

```
oratext* XmlXPtrLocToString(
   xmlxptrloc* loc);
```

パラメータ	IN/OUT	説明
loc	IN	位置。

戻り値

(oratext *) 文字列。

XPtrLocSet インタフェース

表 11-3 に、XPtrLocSet インタフェースで使用できるメソッドの概要を示します。

表 11-3 XPtrLocSet メソッドの概要 : XPointer パッケージ

ファンクション	概要
11-6 ページ「XmlXPtrLocSetFree」	位置セットを解放します。
11-7 ページ「XmlXPtrLocSetGetItem」	XPtrLocSet 内の idx の位置を戻します。
11-7 ページ「XmlXPtrLocSetGetLength」	XPtrLocSet の長さを戻します。

XmlXPtrLocSetFree

XPointer または XPtrLocSet インタフェースによって戻されたすべての位置セットに対し、ユーザーが、このファンクションをコールする必要があります。

構文

```
void XmlXPtrLocSetFree(
   xmlxptrlocset* locset);
```

パラメータ	IN/OUT	説明
locset	IN	位置セット。

XmlXPtrLocSetGetItem

位置セット内の idx の位置を戻します。最初の位置は 1 です。

構文

```
xmlxptrloc* XmlXPtrLocSetGetItem(
   xmlxptrlocset* locset,
   ub4 idx);
```

パラメータ	IN/OUT	説明
locset	IN	位置セット。
idx	IN	位置のインデックス。

戻り値

(xmlxptrloc *) idx の位置。

XmlXPtrLocSetGetLength

位置セット内の位置の数を戻します。

構文

```
ub4 XmlXPtrLocSetGetLength(
   xmlxptrlocset* locset);
```

パラメータ	IN/OUT	説明
locset	IN	位置セット。

戻り値

(ub4) locset 内のノードの数。

12

C 用の XSLT API パッケージ

XSLT パッケージは、XSL 処理に関連のある型およびメソッドを実装します。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XSLT インタフェース

表 12-1 に、XSLT インタフェースで使用できるメソッドの概要を示します。

表 12-1 XSLT メソッドの概要

ファンクション	概要
12-3 ページ「XmlXslCreate」	XSL コンテキストを作成します。
12-3 ページ「XmlXslDestroy」	XSL コンテキストを破棄します。
12-4 ページ「XmlXslGetBaseURI」	XSL Processors のベース URI を取得します。
12-4 ページ「XmlXslGetOutput」	XSL の結果のフラグメントを取得します。
12-5 ページ「XmlXslGetStylesheetDom」	XSL スタイルシート・ドキュメントを取得します。
12-5 ページ「XmlXslGetTextParam」	XSL テキストのパラメータ値を取得します。
12-6 ページ「XmlXslProcess」	インスタンス・ドキュメントを XSL 処理します。
12-6 ページ「XmlXslResetAllParams」	XSL Processors のパラメータをリセットします。
12-7 ページ「XmlXslSetOutputDom」	XSL コンテキストの出力 DOM を設定します。
12-7 ページ「XmlXslSetOutputEncoding」	XSL コンテキストの出力エンコーディングを設定します。
12-8 ページ「XmlXslSetOutputMethod」	XSL コンテキストの出力メソッドを設定します。
12-8 ページ「XmlXslSetOutputSax」	XSL コンテキストの出力 SAX を設定します。
12-9 ページ「XmlXslSetOutputStream」	XSL コンテキストの出力ストリームを設定します。
12-9 ページ「XmlXslSetTextParam」	XSL コンテキストの出力テキスト・パラメータを設定します。

XmlXslCreate

XSLT コンテキストを作成します。

構文

```
xslctx *XmlXslCreate(
   xmlctx *ctx,
   xmldocnode *xsl,
   oratext *baseuri,
   xmlerr *err);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。
xsl	IN	XSL スタイルシート・ドキュメント・オブジェクト。
baseuri	IN	ドキュメントの挿入およびインポートに使用するベース URI。
err	IN/OUT	戻されたエラー・コード。

戻り値

(xslctx *) XSLT コンテキスト。

関連項目： XmlXslDestroy

XmlXslDestroy

XSL コンテキストを破棄します。

構文

```
xmlerr XmlXslDestroy(
   xslctx *ctx);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト。

戻り値

(xmlerr) エラー・コード。

関連項目： XmlXslCreate

XmlXslGetBaseURI

XSL Processors のベース URI を取得します。

構文

```
oratext *XmlXslGetBaseURI(
   xslctx *ctx);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。

戻り値

(oratext *) ベース URI。

XmlXslGetOutput

XSL の結果のフラグメントを取得します。

構文

```
xmlfragnode *XmlXslGetOutput(
   xslctx *ctx);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。

戻り値

(xmlfragnode *) 結果のフラグメント。

XmlXslGetStylesheetDom

XSL スタイルシート・ドキュメントを取得します。

構文

```
xmldocnode *XmlXslGetStylesheetDom(
   xslctx *ctx);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。

戻り値

(xmldocnode *) スタイルシート・ドキュメント。

XmlXslGetTextParam

XSL テキストのパラメータ値を取得します。

構文

```
oratext *XmlXslGetTextParam(
   xslctx *ctx,
   oratext *name);
```

パラメータ	IN/OUT	説明
ctx	IN	XML コンテキスト・オブジェクト。
name	IN	最上位のパラメータ値の名前。

戻り値

(oratext *) パラメータ値。

関連項目： XmlXslSetTextParam

XmlXslProcess

インスタンス・ドキュメントを XSL 処理します。

構文

```
xmlerr XmlXslProcess(
   xslctx *ctx,
   xmldocnode *xml,
   boolean normalize);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。
xml	IN	処理するインスタンス・ドキュメント。
normalize	IN	TRUE の場合は、ドキュメントを正規化するよう XSL Processors に強制します。

戻り値

(xmlerr) エラー・コード。

XmlXslResetAllParams

追加された最上位のパラメータをすべてリセットします。

構文

```
xmlerr XmlXslResetAllParams(
   xslctx *ctx);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

関連項目： XmlXslSetTextParam

XmlXslSetOutputDom

xslctx の出力 DOM を設定します。

構文

```
xmlerr XmlXslSetOutputDom(
   xslctx *ctx,
   xmldocnode *doc);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。
doc	IN	出力ノード。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

XmlXslSetOutputEncoding

xslctx の出力エンコーディングを設定します。

構文

```
xmlerr XmlXslSetOutputEncoding(
   xslctx *ctx,
   oratext* encoding);
```

パラメータ	IN/OUT	説明
ctx	IN	XML コンテキスト・オブジェクト。
encoding	IN	出力のエンコーディング。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

XmlXslSetOutputMethod

xslctx の出力メソッドを設定します。

構文

```
xmlerr XmlXslSetOutputMethod(
   xslctx *ctx,
   xmlxslomethod method);
```

パラメータ	IN/OUT	説明
ctx	IN	XML コンテキスト・オブジェクト。
encoding	IN	XSL の出力メソッド。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

XmlXslSetOutputSax

xslctx の出力 SAX を設定します。

構文

```
xmlerr XmlXslSetOutputSax(
   xslctx *ctx,
   xmlsaxcb* saxcb,
   void *saxctx);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。
saxcb	IN	SAX コールバック・オブジェクト。
saxctx	IN	SAX コールバック・コンテキスト。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

XmlXslSetOutputStream

構文

```
xmlerr XmlXslSetOutputStream(
   xslctx *ctx,
   xmlostream *stream);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。
stream	IN	出力ストリーム・オブジェクト。

戻り値

(xmlxsl) エラー・コード。正常に終了した場合は XMLXSL_SUCC [0] が戻されます。

XmlXslSetTextParam

xslctx の出力テキスト・パラメータを設定します。

構文

```
xmlerr XmlXslSetTextParam(
   xslctx *ctx,
   oratext *name,
   oratext *value);
```

パラメータ	IN/OUT	説明
ctx	IN	XSL コンテキスト・オブジェクト。
name	IN	最上位のパラメータの名前。
value	IN	最上位のパラメータの値。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

関連項目： XmlXslGetTextParam

13

C 用の XSLTVM API パッケージ

XSLTVM パッケージでは、1999 年 11 月 16 日の W3C 勧告で指定された XSL Transformation（XSLT）言語を実装します。XSLTVM パッケージには、2 つのインタフェースが含まれています。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XSLTVM の使用

XSLT 仮想マシンは、コンパイル済の XSLT コードを実行するよう設計された CPU のソフトウェアによる実装です。仮想マシンの概念は、XSLT スタイルシートを、バイトコードのシーケンスまたは「XSLT CPU」のマシン用手順にコンパイルするコンパイラに基づいています。バイトコード・プログラムは、2 バイト単位のプラットフォームに依存しないシーケンスです。これは、異なる XSLTVM 上で格納、キャッシュおよび実行できます。XSLTVM は、バイトコード・プログラムを使用して、インスタンス XML 文書を変換します。この方法では、コンパイル（設計）時と実行時の計算を明確に分離し、手順間でデータを変換するための統一された手段を指定します。

API パッケージを使用する一般的な場面には、次の手順が含まれます。

1. XML メタ・コンテキスト・オブジェクトを作成 / 使用します。

   ```
   xctx = XmlCreate(,...);
   ```

2. XSLT コンパイラ・オブジェクトを作成 / 使用します。

   ```
   comp = XmlXvmCreateComp(xctx);
   ```

3. XSLT スタイルシートをコンパイルし、結果のバイトコードをキャッシュします。

   ```
   code = XmlXvmCompileFile(comp, xslFile, baseuri, flags, );
   ```

4. XSLTVM オブジェクトを作成 / 使用します。XSLTVM がスタックのオーバーフローのメッセージで終了する場合、またはより小さいメモリー・フットプリントが必要な場合、明示的スタック・サイズの設定が必要となります（`XmlXvmCreate` を参照）。

   ```
   vm = XmlXvmCreate(xctx, "StringStack", 32, "NodeStack", 24,
   NULL);
   ```

5. XSLTVM オブジェクトにスタイルシート・バイトコードを設定します。

   ```
   len = XmlXvmGetBytecodeLength(code, ); err =
   XmlXvmSetBytecodeBuffer(vm, code, len);
   ```

6. インスタンス XML 文書を変換します。

   ```
   err = XmlXvmTransformFile(vm, xmlFile, baseuri);
   ```

7. クリーン・アップします。

   ```
   XmlXvmDestroy(vm);

   XmlXvmDestroyComp(comp);

   XmlDestroy(xctx);
   ```

XSLTC インタフェース

表 13-1 に、XSLTVM インタフェースで使用できるメソッドの概要を示します。

表 13-1 XSLTC メソッドの概要 : XSLTVM パッケージ

ファンクション	概要
13-4 ページ「XmlXvmCompileBuffer」	バッファからバイトコードに XSLT スタイルシートをコンパイルします。
13-5 ページ「XmlXvmCompileDom」	DOM からバイトコードに XSLT スタイルシートをコンパイルします。
13-6 ページ「XmlXvmCompileFile」	ファイルからバイトコードに XSLT スタイルシートをコンパイルします。
13-7 ページ「XmlXvmCompileURI」	URI からバイトコードに XSLT スタイルシートをコンパイルします。
13-8 ページ「XmlXvmCompileXPath」	XPath 式をコンパイルします。
13-8 ページ「XmlXvmCreateComp」	XSLT コンパイラを作成します。
13-9 ページ「XmlXvmDestroyComp」	XSLT コンパイラ・オブジェクトを破棄します。
13-9 ページ「XmlXvmGetBytecodeLength」	バイトコードの長さを戻します。

XmlXvmCompileBuffer

バッファからバイトコードに XSLT スタイルシートをコンパイルします。コンパイラ・フラグは、次のいずれか 1 つ以上になります。

- XMLXVM_DEBUG: バイトコードにデバッグ情報を含めるよう、コンパイラに強制します。
- XMLXVM_STRIPSPACE: <xsl:strip-space elements="*"/> と同等です。

生成されたバイトコードはコンパイラ・バッファに常駐し、次のスタイルシートがコンパイルされるか、コンパイラ・オブジェクトが削除されると解放されます。したがって、バイトコードを再利用する場合、別の場所にコピーする必要があります。

構文

```
ub2 *XmlXvmCompileBuffer(
   xmlxvmcomp *comp,
   oratext *buffer,
   ub4 length,
   oratext *baseURI,
   xmlxvmflag flags,
   xmlerr *error);
```

パラメータ	IN/OUT	説明
comp	IN	コンパイラ・オブジェクト。
buffer	IN	スタイルシート・ドキュメントが含まれているバッファへのポインタ。
length	IN	バイト単位で示したスタイルシート・ドキュメントの長さ。
baseuri	IN	ドキュメントのベース URI。
flags	IN	現在のコンパイルのフラグ。
error	OUT	戻されたエラー・コード。

戻り値

(ub2 *) バイトコード。エラー発生時には NULL が戻されます。

関連項目： XmlXvmCompileFile、XmlXvmCompileURI、XmlXvmCompileDom

XmlXvmCompileDom

DOM からバイトコードに XSLT スタイルシートをコンパイルします。コンパイラ・フラグは、次のいずれか 1 つ以上になります。

- XMLXVM_DEBUG: バイトコードにデバッグ情報を含めるよう、コンパイラに強制します。
- XMLXVM_STRIPSPACE: <xsl:strip-space elements="*"/> と同等です。

生成されたバイトコードはコンパイラ・バッファに常駐し、次のスタイルシートがコンパイルされるか、コンパイラ・オブジェクトが削除されると解放されます。したがって、バイトコードを再利用する場合、別の場所にコピーする必要があります。

構文

```
ub2 *XmlXvmCompileDom(
   xmlxvmcomp *comp,
   xmldocnode *root,
   xmlxvmflag flags,
   xmlerr *error);
```

パラメータ	IN/OUT	説明
comp	IN	コンパイラ・オブジェクト。
root	IN	スタイルシート DOM のルート要素。
flags	IN	現在のコンパイルのフラグ。
error	OUT	戻されたエラー・コード。

戻り値

(ub2 *) バイトコード。エラー発生時には NULL が戻されます。

関連項目： XmlXvmCompileFile、XmlXvmCompileBuffer、XmlXvmCompileURI

XmlXvmCompileFile

ファイルからバイトコードに XSLT スタイルシートをコンパイルします。コンパイラ・フラグは、次のいずれか 1 つ以上になります。

- XMLXVM_DEBUG: バイトコードにデバッグ情報を含めるよう、コンパイラに強制します。
- XMLXVM_STRIPSPACE: <xsl:strip-space elements="*"/> と同等です。

生成されたバイトコードはコンパイラ・バッファに常駐し、次のスタイルシートがコンパイルされるか、コンパイラ・オブジェクトが削除されると解放されます。したがって、バイトコードを再利用する場合、別の場所にコピーする必要があります。

構文

```
ub2 *XmlXvmCompileFile(
   xmlxvmcomp *comp,
   oratext *path,
   oratext *baseURI,
   xmlxvmflag flags,
   xmlerr *error);
```

パラメータ	IN/OUT	説明
comp	IN	コンパイラ・オブジェクト。
path	IN	XSL スタイルシート・ファイルのパス。
baseuri	IN	ドキュメントのベース URI。
flags	IN	現在のコンパイルのフラグ。
error	OUT	戻されたエラー・コード。

戻り値

(ub2 *) バイトコード。エラー発生時には NULL が戻されます。

関連項目： XmlXvmCompileURI、XmlXvmCompileBuffer、XmlXvmCompileDom

XmlXvmCompileURI

URI からバイトコードに XSLT スタイルシートをコンパイルします。コンパイラ・フラグは、次のいずれか 1 つ以上になります。

- XMLXVM_DEBUG: バイトコードにデバッグ情報を含めるよう、コンパイラに強制します。
- XMLXVM_STRIPSPACE: <xsl:strip-space elements="*"/> と同等です。

生成されたバイトコードはコンパイラ・バッファに常駐し、次のスタイルシートがコンパイルされるか、コンパイラ・オブジェクトが削除されると解放されます。したがって、バイトコードを再利用する場合、別の場所にコピーする必要があります。

構文

```
ub2 *XmlXvmCompileURI(
   xmlxvmcomp *comp,
   oratext *uri,
   xmlxvmflag flags,
   xmlerr *error);
```

パラメータ	IN/OUT	説明
comp	IN	コンパイラ・オブジェクト。
uri	IN	XSL スタイルシートが含まれているファイルの URI。
flags	IN	現在のコンパイルのフラグ。
error	OUT	戻されたエラー・コード。

戻り値

(ub2 *) バイトコード。エラー発生時には NULL が戻されます。

関連項目： XmlXvmCompileFile、XmlXvmCompileBuffer、XmlXvmCompileDom

XmlXvmCompileXPath

XPath 式をコンパイルします。オプションの pfxmap は、名前空間の接頭辞を XPath 式内の URI にマップするために使用されます。これは、NULL で終了する接頭辞や URI 値などの配列です。

構文

```
ub2 *XmlXvmCompileXPath(
   xmlxvmcomp *comp,
   oratext *xpath,
   oratext **pfxmap,
   xmlerr *error);
```

パラメータ	IN/OUT	説明
comp	IN	コンパイラ・オブジェクト。
xpath	IN	XPath 式。
pfxmap	IN	接頭辞と URI のマッピングの配列。
error	OUT	戻されたエラー・コード。

戻り値

(ub2 *) XPath 式のバイトコード。エラー発生時には NULL が戻されます。

XmlXvmCreateComp

XSLT コンパイラ・オブジェクトを作成します。XSLT コンパイラは、XSLT スタイルシートをバイトコードにコンパイルするために使用します。

構文

```
xmlxvmcomp *XmlXvmCreateComp(
   xmlctx *xctx);
```

パラメータ	IN/OUT	説明
xctx	IN	XML コンテキスト。

戻り値

(xmlxvmcomp *) XSLT コンパイラ・オブジェクト。エラー発生時には NULL が戻されます。

関連項目： XmlXvmDestroyComp

XmlXvmDestroyComp

XSLT コンパイラ・オブジェクトを破棄します。

構文

```
void XmlXvmDestroyComp(
   xmlxvmcomp *comp);
```

パラメータ	IN/OUT	説明
comp	IN	XSLT コンパイラ・オブジェクト。

関連項目： XmlXvmCreateComp

XmlXvmGetBytecodeLength

バイトコードの長さは、バイトコードをコピーする場合、または XSLTVM で設定する場合に必要です。

構文

```
ub4 XmlXvmGetBytecodeLength(
   ub2 *bytecode,
   xmlerr *error);
```

パラメータ	IN/OUT	説明
bytecode	IN	バイトコードのバッファ。
error	OUT	戻されたエラー・コード。

戻り値

(ub4) バイト単位で示したバイトコードの長さ。

XSLTVM インタフェース

表 13-2 に、XSLTVM インタフェースで使用できるメソッドの概要を示します。

表 13-2 XSLTVM メソッドの概要 : XSLTVM パッケージ

ファンクション	概要
13-11 ページ「XMLXVM_DEBUG_F」	XMLXSLTVM デバッグ・ファンクション。
13-12 ページ「XmlXvmCreate」	XSLT 仮想マシンを作成します。
13-13 ページ「XmlXvmDestroy」	XSLT 仮想マシンを破棄します。
13-13 ページ「XmlXvmEvaluateXPath」	コンパイル済の XPath 式を評価します。
13-14 ページ「XmlXvmGetObjectBoolean」	XPath オブジェクトのブール値を取得します。
13-14 ページ「XmlXvmGetObjectNSetNode」	ノードセット型 XPath オブジェクトからノードを取得します。
13-15 ページ「XmlXvmGetObjectNSetNum」	ノードセット型 XPath オブジェクトからノードの数を取得します。
13-15 ページ「XmlXvmGetObjectNumber」	XPath オブジェクトから数値を取得します。
13-16 ページ「XmlXvmGetObjectString」	XPath オブジェクトから文字列を取得します。
13-16 ページ「XmlXvmGetObjectType」	XPath オブジェクトの型を取得します。
13-17 ページ「XmlXvmGetOutputDom」	出力 DOM を戻します。
13-17 ページ「XmlXvmResetParams」	スタイルシートの最上位テキストのパラメータをリセットします。
13-18 ページ「XmlXvmSetBaseURI」	XLTVM のベース URI を設定します。
13-18 ページ「XmlXvmSetBytecodeBuffer」	コンパイルしたバイトコードを設定します。
13-19 ページ「XmlXvmSetBytecodeFile」	ファイルからコンパイルしたバイトコードを設定します。
13-20 ページ「XmlXvmSetBytecodeURI」	コンパイルしたバイトコードを設定します。
13-20 ページ「XmlXvmSetDebugFunc」	デバッグ用のコールバック・ファンクションを設定します。
13-21 ページ「XmlXvmSetOutputDom」	出力ドキュメント・ノードを XSLTVM に設定します。
13-22 ページ「XmlXvmSetOutputEncoding」	XSLTVM 出力のエンコーディングを設定します。

表 13-2 XSLTVM メソッドの概要 : XSLTVM パッケージ（続き）

ファンクション	概要
13-22 ページ「XmlXvmSetOutputSax」	出力 SAX を XSLTVM に設定します。
13-23 ページ「XmlXvmSetOutputStream」	ユーザー定義のストリームを XSLTVM 出力に設定します。
13-23 ページ「XmlXvmSetTextParam」	スタイルシートの最上位テキストのパラメータを設定します。
13-24 ページ「XmlXvmTransformBuffer」	メモリー内の XML 文書に対してコンパイルした XSLT スタイルシートを実行します。
13-24 ページ「XmlXvmTransformDom」	DOM として XML 文書に対してコンパイルした XSLT スタイルシートを実行します。
13-25 ページ「XmlXvmTransformFile」	ファイル内の XML 文書に対してコンパイルした XSLT スタイルシートを実行します。
13-26 ページ「XmlXvmTransformURI」	URI の XML 文書に対してコンパイルした XSLT スタイルシートを実行します。

XMLXVM_DEBUG_F

XSLTVM のデバッグ・コールバック・ファンクション。

構文

```
#define XMLXVM_DEBUG_F(func, line, file, obj, n)
void func(
   ub2 line,
   oratext *file,
   xvmobj *obj,
   ub4 n)
```

パラメータ	IN/OUT	説明
line	IN	ソース・スタイルシートの行番号。
file	IN	スタイルシートのファイル名。
obj	IN	現行の VM オブジェクト。
n	IN	現行ノードのインデックス。

関連項目： XmlXvmSetDebugFunc

XmlXvmCreate

XSLT 仮想マシンを作成します。この API を使用して、次の中から 0（ゼロ）以上の XSLTVM プロパティを設定できます。

- `"VMStack", size`: 主要な VM スタックの size[Kbyte] を設定します。デフォルトのサイズは 4K です。
- `"NodeStack", size`: ノードスタックの size[Kbyte] を設定します。デフォルトのサイズは 16K です。
- `"StringStack", size`: 文字列スタックの size[Kbyte] を設定します。デフォルトのサイズは 64K です。

スタック・サイズが指定されていない場合は、デフォルトのサイズが使用されます。XSLTVM がスタックのオーバーフローのメッセージで終了する場合、またはより小さいメモリー・フットプリントが必要な場合、明示的スタック・サイズの設定が必要となります。

構文

```
xmlxvm *XmlXvmCreate(
   xmlctx *xctx,
   list);
```

パラメータ	IN/OUT	説明
`xctx`	`IN`	XML コンテキスト。
`list`	`IN`	設定するプロパティの `NULL` で終了するリスト（空でも可）。

戻り値

`(xmlxvm *)` XSLT 仮想マシン・オブジェクト。エラー発生時には `NULL` が戻されます。

関連項目： XmlXvmDestroy

XmlXvmDestroy

XSLT 仮想マシンを破棄します。

構文

```
void XmlXvmDestroy(
   xmlxvm *vm);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。

関連項目： XmlXvmCreate

XmlXvmEvaluateXPath

コンパイル済の XPath 式を評価します。

構文

```
xvmobj *XmlXvmEvaluateXPath(
   xmlxvm *vm,
   ub2 *bytecode,
   ub4 ctxpos,
   ub4 ctxsize,
   xmlnode *ctxnode);
```

パラメータ	IN/OUT	説明
vm	IN	XSLTVM オブジェクト。
bytecode	IN	XPath 式のバイトコード。
ctxpos	IN	現在のコンテキストの位置。
ctxsize	IN	現在のコンテキストのサイズ。
ctxnode	IN	現在のコンテキスト・ノード。

戻り値

(xvmobj *) XPath オブジェクト。

XmlXvmGetObjectBoolean

XPath オブジェクトのブール値を取得します。

構文

```
boolean XmlXvmGetObjectBoolean(
   xvmobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	オブジェクト。

戻り値

(boolean) XPath オブジェクトの値。

関連項目： XmlXvmGetObjectType、XmlXvmGetObjectNSetNum、XmlXvmGetObjectNSetNode、XmlXvmGetObjectNumber、XmlXvmGetObjectBoolean

XmlXvmGetObjectNSetNode

ノードセット型 XPath オブジェクトからノードを取得します。

構文

```
xmlnode *XmlXvmGetObjectNSetNode(
   xvmobj *obj,
   ub4 i);
```

パラメータ	IN/OUT	説明
obj	IN	オブジェクト。
i	IN	ノードセットのノードのインデックス。

戻り値

(xmlnode *) オブジェクト型または値。

関連項目： XmlXvmGetObjectType、XmlXvmGetObjectNSetNum、XmlXvmGetObjectString、XmlXvmGetObjectNumber、XmlXvmGetObjectBoolean

XmlXvmGetObjectNSetNum

ノードセット型 XPath オブジェクトからノードの数を取得します。

構文

```
ub4 XmlXvmGetObjectNSetNum(
   xvmobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	オブジェクト。

戻り値

(ub4) ノードセット内のノードの数。

関連項目： XmlXvmGetObjectType、XmlXvmGetObjectNSetNode、XmlXvmGetObjectString、XmlXvmGetObjectNumber、XmlXvmGetObjectBoolean

XmlXvmGetObjectNumber

XPath オブジェクトから数値を取得します。

構文

```
double XmlXvmGetObjectNumber(
   xvmobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	オブジェクト。

戻り値

(double) 数値。

関連項目： XmlXvmGetObjectType、XmlXvmGetObjectNSetNum、XmlXvmGetObjectNSetNode、XmlXvmGetObjectString、XmlXvmGetObjectBoolean

XmlXvmGetObjectString

XPath オブジェクトから文字列を取得します。

構文

```
oratext *XmlXvmGetObjectString(
   xvmobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	オブジェクト。

戻り値

(oratext *) 文字列。

関連項目： XmlXvmGetObjectType、XmlXvmGetObjectNSetNum、XmlXvmGetObjectNSetNode、XmlXvmGetObjectNumber、XmlXvmGetObjectBoolean

XmlXvmGetObjectType

XPath オブジェクトの型を取得します。

構文

```
xmlxvmobjtype XmlXvmGetObjectType(
   xvmobj *obj);
```

パラメータ	IN/OUT	説明
obj	IN	オブジェクト。

戻り値

(xmlxvmobjtype) オブジェクトの型コード。

関連項目： XmlXvmGetObjectNSetNum、XmlXvmGetObjectNSetNode、XmlXvmGetObjectString、XmlXvmGetObjectNumber、XmlXvmGetObjectBoolean

XmlXvmGetOutputDom

結果の DOM ツリーのルート・ノードを戻します（存在する場合）。VM が DOM ツリーを出力するための設定を行う変換の前に、XmlXvmSetOutputDom を使用する必要があります（デフォルトの VM 出力はストリームです）。

構文

```
xmlfragnode *XmlXvmGetOutputDom(
   xmlxvm *vm);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。

戻り値

(xmlfragnode *) 出力 DOM。SAX または Stream 出力の場合は NULL が戻されます。

関連項目： XmlXvmSetOutputDom

XmlXvmResetParams

スタイルシートの最上位のパラメータをデフォルト値にリセットします。

構文

```
void XmlXvmResetParams(
   xmlxvm *vm);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。

XmlXvmSetBaseURI

XSLTVM のベース URI を設定します。baseuri は、document または XmlXvmTransformFile を使用して XML 文書を変換する際に文書をロードするパスを、VM が構成するために使用されます。

構文

```
xmlerr XmlXvmSetBaseURI(
   xmlxvm *vm,
   oratext *baseuri);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
baseuri	IN	ドキュメントの読込みおよび書込みに使用する VM のベース URI。

戻り値

(xmlerr) エラー・コード。

XmlXvmSetBytecodeBuffer

コンパイルしたバイトコードをバッファから設定します。事前に設定されているバイトコードは置換されます。スタイルシートのバイトコードが設定されていない場合、XML 変換は実行できません。VM はバイトコードを内部バッファにコピーしないため、VM が使用を完了するまでバイトコードを解放しないでください。

構文

```
xmlerr XmlXvmSetBytecodeBuffer(
   xmlxvm *vm,
   ub2 *buffer,
   size_t buflen);
```

パラメータ	IN/OUT	説明
vm	IN	XSLT VM のコンテキスト。
buffer	IN	ユーザーのバッファ。
buflen	IN	バイト単位で示したバッファ・サイズ。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

関連項目： XmlXvmSetBytecodeFile、XmlXvmSetBytecodeURI

XmlXvmSetBytecodeFile

コンパイルしたバイトコードをファイルから設定します。事前に設定されているバイトコードは置換されます。スタイルシートのバイトコードが設定されていない場合、XML 変換は実行できません。

構文

```
xmlerr XmlXvmSetBytecodeFile(
   xmlxvm *vm,
   oratext *path);
```

パラメータ	IN/OUT	説明
vm	IN	XSLT VM のコンテキスト。
path	IN	バイトコード・ファイルのパス。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

関連項目： XmlXvmSetBytecodeURI、XmlXvmSetBytecodeBuffer

XmlXvmSetBytecodeURI

コンパイルしたバイトコードを URI から設定します。事前に設定されているバイトコードは置換されます。スタイルシートのバイトコードが設定されていない場合、XML 変換は実行できません。

構文

```
xmlerr XmlXvmSetBytecodeURI(
   xmlxvm *vm,
   oratext *uri);
```

パラメータ	IN/OUT	説明
vm	IN	XSLT VM のコンテキスト。
uri	IN	バイトコード・ファイルのパス。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

関連項目： XmlXvmSetBytecodeFile、XmlXvmSetBytecodeBuffer

XmlXvmSetDebugFunc

このユーザー・コールバック・ファンクションは、実行が XSLT スタイルシートの新しい行に到達するたびに、VM によって起動されます。VM は、ユーザーに、スタイルシートのファイル名、行番号、現在のコンテキスト・ノードセットおよびノードセット内の現在のノード・インデックスを渡します。重要 : スタイルシートは、XMLXVM_DEBUG フラグを使用してコンパイルする必要があります。

構文

```
#define XMLXVM_DEBUG_FUNC(func)
void func (ub2 line, oratext *filename, xvmobj *obj, ub4 n)
xmlerr XmlXvmSetDebugFunc(
   xmlxvm *vm,
   XMLXVM_DEBUG_FUNC(debugcallback));
```

パラメータ	IN/OUT	説明
vm	IN	XSLT VM のコンテキスト。
func	IN	コールバック・ファンクション。

戻り値

(xmlerr) 数値のエラー・コード。正常に終了した場合は XMLERR_OK [0] が戻されます。

XmlXvmSetOutputDom

出力 DOM を XSLTVM に設定します。xmldocnode==NULL の場合、結果の DOM ツリーは VM オブジェクトに属し、新しく変換が実行されるか、または VM オブジェクトが削除されると、DOM ツリーは削除されます。結果の DOM ツリーを長期間使用する必要がある場合、xmldocnode を作成し、VM オブジェクトで設定する必要があります。

構文

```
xmlerr XmlXvmSetOutputDom(
   xmlxvm *vm,
   xmldocnode *doc);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
doc	IN	空のドキュメント。

戻り値

(xmlerr) エラー・コード。

XmlXvmSetOutputEncoding

XSLTVM ストリーム出力のエンコーディングを設定します。入力（データ）のエンコーディングが、この API によって設定されたエンコーディングと異なる場合、エンコーディングの変換が実行されます。この API は、XSLT スタイルシートに設定されているエンコーディング（設定されている場合）をオーバーライドします。

構文

```
xmlerr XmlXvmSetOutputEncoding(
   xmlxvm *vm,
   oratext *encoding);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
encoding	IN	出力のエンコーディング。

戻り値

(xmlerr) エラー・コード。

XmlXvmSetOutputSax

出力 SAX を XSLTVM に設定します。SAX コールバック・インタフェース・オブジェクトが提供されている場合、VM はユーザーが指定したコールバック・ファンクションを使用して、結果のドキュメントを SAX イベントの形式で出力します。

構文

```
xmlerr XmlXvmSetOutputSax(
   xmlxvm *vm,
   xmlsaxcb *saxcb,
   void *saxctx);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
saxcb	IN	SAX コールバック・オブジェクト。
saxctx	IN	SAX コンテキスト。

戻り値

(xmlerr) エラー・コード。

XmlXvmSetOutputStream

XSLTVM 出力をユーザー定義のストリームに設定します。デフォルトの XSLTVM 出力はストリームです。この API は、書込み時に、ユーザーの指定した API によってデフォルトのストリームをオーバーライドします。

構文

```
xmlerr XmlXvmSetOutputStream(
   xmlxvm *vm,
   xmlostream *ostream);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
ostream	IN	ストリーム・オブジェクト。

戻り値

(xmlerr) エラー・コード。

XmlXvmSetTextParam

スタイルシートの最上位テキストのパラメータを設定します。XSLT スタイルシートで設定されているパラメータ値は上書きされます。変換のたびに最上位のパラメータはスタイルシートの値にリセットされるため、この API を再度コールする必要があります。

構文

```
xmlerr XmlXvmSetTextParam(
   xmlxvm *vm,
   oratext *name,
   oratext *value);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
name	IN	最上位のパラメータの名前。
value	IN	最上位のパラメータの値。

戻り値

(xmlerr) エラー・コード。正常に終了した場合は XMLERR_SUCC [0] が戻されます。

XmlXvmTransformBuffer

メモリー内の XML 文書に対してコンパイルした XSLT スタイルシートを実行します。このコールの前に、XmlXvmSetBytecodeXXX を使用して、コンパイル済の XSLT スタイルシート（バイトコード）を設定する必要があります。

構文

```
xmlerr XmlXvmTransformBuffer(
   xmlxvm *vm,
   oratext *buffer,
   ub4 length,
   oratext *baseURI);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
buffer	IN	XML 文書が含まれている、NULL で終了するバッファ。
length	IN	XML 文書の長さ。
baseURI	IN	XML 文書のベース URI。

戻り値

(xmlerr) エラー・コード。

関連項目： XmlXvmTransformFile、XmlXvmTransformURI、XmlXvmTransformDom

XmlXvmTransformDom

DOM として XML 文書に対してコンパイルした XSLT スタイルシートを実行します。このコールの前に、XmlXvmSetBytecodeXXX を使用して、コンパイル済の XSLT スタイルシート（バイトコード）を設定する必要があります。

構文

```
xmlerr XmlXvmTransformDom(
   xmlxvm *vm,
   xmldocnode *root);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
root	IN	XML 文書の DOM のルート要素。

戻り値

(xmlerr) エラー・コード。

関連項目： XmlXvmTransformFile、XmlXvmTransformURI、XmlXvmTransformBuffer

XmlXvmTransformFile

ファイル内の XML 文書に対してコンパイルした XSLT スタイルシートを実行します。このコールの前に、XmlXvmSetBytecodeXXX を使用して、コンパイル済の XSLT スタイルシート（バイトコード）を設定する必要があります。

構文

```
xmlerr XmlXvmTransformFile(
   xmlxvm *vm,
   oratext *path,
   oratext *baseURI);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
path	IN	変換する XML 文書のパス。
baseURI	IN	XML 文書のベース URI。

戻り値

(xmlerr) エラー・コード。

関連項目： XmlXvmTransformURI、XmlXvmTransformBuffer、XmlXvmTransformDom

XmlXvmTransformURI

URI の XML 文書に対してコンパイルした XSLT スタイルシートを実行します。このコールの前に、XmlXvmSetBytecodeXXX を使用して、コンパイル済の XSLT スタイルシート（バイトコード）を設定する必要があります。

構文

```
xmlerr XmlXvmTransformURI(
   xmlxvm *vm,
   oratext *uri);
```

パラメータ	IN/OUT	説明
vm	IN	VM オブジェクト。
uri	IN	変換する XML 文書の URI。

戻り値

(xmlerr) エラー・コード。

関連項目： XmlXvmTransformFile、XmlXvmTransformBuffer、XmlXvmTransformDom

第 II 部

C++ 用の XML API

第 V 部に含まれる章は、次のとおりです。

14

C++ 用の Ctx API パッケージ

Ctx は TCtx XML コンテキスト関連の型とインタフェースの名前空間です。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

Ctx のデータ型

表 14-1 に Ctx パッケージのデータ型の概要を示します。

表 14-1 データ型の概要 : Ctx パッケージ

データ型	説明
14-2 ページ「encoding」	サポートされる唯一のエンコーディング。
14-2 ページ「encodings」	エンコーディングの配列。

encoding

サポートされる唯一のエンコーディング。

定義

```
typedef struct encoding {
   oratext *encname;
   oratext *encvalue;
} encoding;
```

encodings

エンコーディングの配列。

定義

```
typedef struct encodings {
   unsigned num;
   encoding *enc;
} encodings;
```

MemAllocator インタフェース

表 14-2 に MemAllocator インタフェースで使用できるメソッドの概要を示します。

表 14-2 MemAllocator メソッドの概要 : Ctx パッケージ

ファンクション	概要
14-3 ページ「alloc」	特定のサイズのメモリーを割り当てます。
14-4 ページ「dealloc」	引数で示されたメモリーの割当てを解除します。
14-4 ページ「~MemAllocator」	仮想デストラクタ。実際のデストラクタに対するインタフェース・レベルのハンドル。

alloc

これはユーザー定義のアロケータ・ファンクションのプロトタイプを定義する、仮想メンバー・ファンクションです。

構文

```
virtual void* alloc(
   ub4 size) = 0;
```

パラメータ	説明
size	メモリー・サイズ。

dealloc

これはユーザー定義のデアロケータ・ファンクションのプロトタイプを定義する、仮想メンバー・ファンクションです。このようなデアロケータは、alloc メンバー・ファンクションで割り当てられたメモリーの割当てを解除するためにサポートされます。

構文

```
virtual void dealloc(
   void* ptr) = 0;
```

パラメータ	説明
ptr	それまでに割り当てられたメモリーのポインタ

~MemAllocator

名前や実装を把握せずに呼び出すことが可能な、実際のデストラクタに対するインタフェース・レベルのハンドルを提供します。

構文

```
virtual ~MemAllocator() {}
```

TCtx インタフェース

表 14-3 に TCtx インタフェースで使用できるメソッドの概要を示します。

表 14-3 TCtx メソッドの概要 : Ctx パッケージ

ファンクション	概要
14-5 ページ「TCtx」	クラス・コンストラクタです。
14-6 ページ「getEncoding」	XML コンテキストで使用されるデータ・エンコーディングを取得します。
14-7 ページ「getErrHandler」	ユーザーが提供するエラー・ハンドラを取得します。
14-7 ページ「getMemAllocator」	メモリー・アロケータを取得します。
14-7 ページ「isSimple」	データ・エンコーディングがシンプルかどうかを示すフラグを取得します。
14-8 ページ「isUnicode」	データ・エンコーディングが Unicode かどうかを示すフラグを取得します。
14-8 ページ「~TCtx」	デストラクタ。空白を消去し、実装を破棄します。

TCtx

TCtx コンストラクタ。コンテキスト・オブジェクトの作成に失敗した場合、XmlException を発生させます。

構文	説明
`TCtx() throw (XMLException)`	このコンストラクタはコンテキスト・オブジェクトを作成し、デフォルト値のパラメータを使用してコンテキスト・オブジェクトを初期化します。

`TCtx(`
`  oratext* name,`
`  ErrorHandler* errh = NULL,`
`  MemAllocator* memalloc = NULL,`
`  encodings* encs = NULL)`
`throw (XMLException)`
	このコンストラクタはコンテキスト・オブジェクトを作成し、ユーザーが指定したパラメータ値を使用してコンテキスト・オブジェクトを初期化します。

構文	説明
TCtx(oratext* name, up4 inpblksize, ErrorIfs* errh = NULL, MemAllocator* memalloc = NULL, encodings* encs = NULL) throw (XMLException)	このコンストラクタはコンテキスト・オブジェクトを作成し、ユーザーが指定したパラメータ値を使用してコンテキスト・オブジェクトを初期化します。入力ソースからのメモリー・ブロック・サイズについては、追加パラメータをとります。

パラメータ	説明
name	ユーザー定義のコンテキスト名
errh	ユーザー定義のエラー・ハンドラ
memalloc	ユーザー定義のメモリー・アロケータ
encs	ユーザー指定のエンコーディング
inpblksize	入力ソースのメモリー・ブロック・サイズ

戻り値

(TCtx) コンテキスト・オブジェクト。

getEncoding

XML コンテキストで使用されるデータ・エンコーディングを戻します。通常、データ・コーディングはユーザーにより選択されるため、このファンクションは必要とされません。ただし、データ・エンコーディングが指定されず、デフォルトが使用できる場合、このファンクションを使用してそのデフォルト・エンコーディングの名前を戻すことができます。

構文

```
oratext* getEncoding() const;
```

戻り値

(oratext *) データ・エンコーディングの名前。

getErrHandler

このメンバー・ファンクションは、コンテキストが作成されたときにユーザーが指定したエラー・ハンドラを戻し、ユーザーが何も指定していない場合は NULL を戻します。

構文

```
ErrorHandler* getErrHandler() const;
```

戻り値

(ErrorHandler *) エラー・ハンドラ・オブジェクトのポインタ、または NULL。

getMemAllocator

このメンバー・ファンクションは、コンテキストが作成されたときにユーザーが指定したメモリー・アロケータ、またはデフォルト・メモリー・アロケータを戻します。C レベルのすべてのメモリー割当てについて、このメモリー・アロケータが使用される必要があります。

構文

```
MemAllocator* getMemAllocator() const;
```

戻り値

(MemAllocator*) メモリー・アロケータ・オブジェクトのポインタ。

isSimple

コンテキストのデータ・エンコーディングが「シンプル」かどうか、すなわち ASCII や EBCDIC のように 1 文字に 1 バイトが使用されているかどうかを示すフラグを戻します。

構文

```
boolean isSimple() const;
```

戻り値

(boolean) データ・エンコーディングが「シンプル」の場合は TRUE、それ以外は FALSE。

isUnicode

コンテキストのデータ・エンコーディングが、1 文字に 2 バイトを使用する Unicode、UTF-16 かどうかを示すフラグを戻します。

構文

```
boolean isUnicode() const;
```

戻り値

(boolean) データ・エンコーディングが Unicode の場合は TRUE、それ以外は FALSE。

~TCtx

デストラクタ。コンテキスト・オブジェクトが不要になった場合に、ユーザーがコールします。

構文

```
~Tctx();
```

15

C++ 用の DOM API パッケージ

このパッケージのインタフェースは、`http://www.w3c.org/TR/DOM-Level-2-Core/core.html` に基づいた DOM レベル 2 Core インタフェースに相当します。

この章の内容は次のとおりです。

- NamedNodeMapRef インタフェース
- NodeFilter インタフェース
- NodeIterator インタフェース
- NodeListRef インタフェース
- NodeRef インタフェース
- NotationRef インタフェース
- ProcessingInstructionRef インタフェース
- Range インタフェース
- RangeException インタフェース
- TextRef インタフェース
- TreeWalker インタフェース

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

DOM の使用

DOM インタフェースは、DOM 仕様の様々な実装に対する汎用参照として表されます。DOM インタフェースはノード別にパラメータ化されるため、各種の専門化およびインスタンス化がサポートされます。そのうち最も重要なのは xmlnode であり、これは現在の C 実装に対応します。

これらの汎用参照には、NULL のような値は含まれません。どの実装でも、状態のない参照を作成することはできません。状態がないことを通知する必要がある場合、例外が発生します。

多くのメソッドでは、DOM ツリーの構成が間違っている場合は SYNTAX_ERR 例外が発生し、不正なパラメータまたは予期せぬ NULL ポインタが使用されている場合は UNDEFINED_ERR 例外が発生します。特定のメソッドでのみ発生するエラーである場合は、メソッド・シグネチャには反映されません。

実際の DOM ツリーはコンテキスト（TCtx）に依存しません。ただし、現行の xmlctx ベースの実装で DOM ツリーを操作するには、現行のコンテキスト（TCtx）へのアクセス権が必要です。そのためには、コンテキスト・ポインタを DOMImplRef のコンストラクタに渡します。マルチスレッド環境では、DOMImplRef は必ずスレッド・コンテキスト内で作成されるため、正しいコンテキストに対するポインタが含まれます。

DOMImplRef により、DOM ツリーを作成できます。DomImplRef は、DomImplRef の通常の非コピー・コンストラクタが起動された場合に作成される、実際の DOMImplementation オブジェクトに対する参照です。これは、DOM ツリーを共有する必要があるマルチスレッド環境で適切に機能し、各スレッドに個別の TCtx が関連付けられます。これは、シングル・スレッド環境でも同じく適切に機能します。

DOMString は、Oracle 実装でサポートされる唯一のエンコーディングです。Oracle の拡張機能により、他のエンコーディングがサポートされます。oratext* データ型は、すべてのエンコーディングに使用されます。

DOM のデータ型

表 15-1 では、DOM パッケージのデータ型を示しています。

表 15-1 データ型の概要 : DOM パッケージ

データ型	説明
15-4 ページ「AcceptNodeCodes」	ノード・フィルタによって戻される値を定義します。
15-5 ページ「CompareHowCode」	比較の型を定義します。
15-5 ページ「DOMNodeType」	ノードの型を定義します。
15-6 ページ「DOMExceptionCode」	DOM 例外のコードを定義します。
15-6 ページ「WhatToShowCode」	フィルタ処理のコードを定義します。
15-7 ページ「RangeExceptionCode」	DOM 範囲例外のコードです。

AcceptNodeCodes

ノード・フィルタによって戻される値を定義します。ノード・イテレータおよびツリー・ウォーカにより使用されます。

定義

```
typedef enum AcceptNodeCode {
   FILTER_ACCEPT    = 1,
   FILTER_REJECT    = 2,
   FILTER_SKIP      = 3
    } AcceptNodeCode;
```

CompareHowCode

比較の型を定義します。

定義

```
typedef enum CompareHowCode {
   START_TO_START = 0,
   START_TO_END = 1,
   END_TO_END = 2,
   END_TO_START = 3 }
CompareHowCode;
```

DOMNodeType

ノードの型を定義します。

定義

```
typedef enum DOMNodeType {
   UNDEFINED_NODE = 0,
   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
} DOMNodeType;
```

DOMExceptionCode

DOM 例外のコードを定義します。

定義

```
typedef enum DOMExceptionCode {
   UNDEFINED_ERR               = 0,
   INDEX_SIZE_ERR              = 1,
   DOMSTRING_SIZE_ERR          = 2,
   HIERARCHY_REQUEST_ERR       = 3,
   WRONG_DOCUMENT_ERR          = 4,
   INVALID_CHARACTER_ERR       = 5,
   NO_DATA_ALLOWED_ERR         = 6,
   NO_MODIFICATION_ALLOWED_ERR = 7,
   NOT_FOUND_ERR               = 8,
   NOT_SUPPORTED_ERR           = 9,
   INUSE_ATTRIBUTE_ERR         = 10,
   INVALID_STATE_ERR           = 11,
   SYNTAX_ERR                  = 12,
   INVALID_MODIFICATION_ERR    = 13,
   NAMESPACE_ERR               = 14,
   INVALID_ACCESS_ERR          = 15
} DOMExceptionCode;
```

WhatToShowCode

フィルタ処理のコードを定義します。

定義

```
typedef unsigned long WhatToShowCode;
   const unsigned long SHOW_ALL = 0xFFFFFFFF; c
   onst unsigned long SHOW_ELEMENT = 0x00000001;
   const unsigned long SHOW_ATTRIBUTE = 0x00000002;
   const unsigned long SHOW_TEXT = 0x00000004;
   const unsigned long SHOW_CDATA_SECTION = 0x00000008;
   const unsigned long SHOW_ENTITY_REFERENCE = 0x00000010;
   const unsigned long SHOW_ENTITY = 0x00000020;
   const unsigned long SHOW_PROCESSING_INSTRUCTION = 0x00000040;
   const unsigned long SHOW_COMMENT = 0x00000080;
   const unsigned long SHOW_DOCUMENT = 0x00000100;
   const unsigned long SHOW_DOCUMENT_TYPE = 0x00000200;
   const unsigned long SHOW_DOCUMENT_FRAGMENT = 0x00000400;
   const unsigned long SHOW_NOTATION = 0x00000800;
```

RangeExceptionCode

DOM 範囲例外のコードです。

定義

```
typedef enum RangeExceptionCode {
   RANGE_UNDEFINED_ERR         = 0,
   BAD_BOUNDARYPOINTS_ERR      = 1,
   INVALID_NODE_TYPE_ERR       = 2
} RangeExceptionCode;
```

AttrRef インタフェース

表 15-2 では、AttrRef インタフェースで使用できるメソッドを示しています。

表 15-2 TreeWalker メソッドの概要 : DOM パッケージ

ファンクション	概要
15-8 ページ「AttrRef」	コンストラクタです。
15-9 ページ「getName」	属性の名前を戻します。
15-9 ページ「getOwnerElement」	属性の元の要素を戻します。
15-9 ページ「getSpecified」	属性が明示的に作成されているかどうかを示すブール値を戻します。
15-10 ページ「getValue」	属性値を戻します。
15-10 ページ「setValue」	属性値を設定します。
15-10 ページ「~AttrRef」	デフォルトのパブリック・デストラクタです。

AttrRef

クラス・コンストラクタです。

構文	説明

`AttrRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createAttribute をコールした後に、指定の属性ノードに対する参照を作成します。

`AttrRef(` `  const AttrRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(AttrRef) Node 参照オブジェクト

getName

(データ・エンコーディング内の) 属性の完全修飾名を、NULL で終了する文字列で戻します。

構文

```
oratext* getName() const;
```

戻り値

(oratext *) 属性名

getOwnerElement

属性の元の要素を戻します。

構文

```
Node* getOwnerElement();
```

戻り値

(Node*) 属性の元の要素ノード。

getSpecified

属性の指定値を戻します。元のドキュメントで属性に値が明示的に指定されている場合は TRUE、それ以外の場合は FALSE を戻します。ノードが属性でない場合は、FALSE を戻します。ユーザーが DOM を介して属性値を設定した場合、指定値は TRUE となります。

構文

```
boolean getSpecified() const;
```

戻り値

(boolean) 属性の指定値

getValue

（データ・エンコーディング内の）属性の値（文字データ）を、NULL で終了する文字列で戻します。文字および汎用エンティティは置換されます。

構文

```
oratext* getValue() const;
```

戻り値

(oratext*) 属性値

setValue

指定された属性値をデータに設定します。新しい値はデータ・エンコーディングに含まれる必要があります。検証、変換、確認は行われません。新しい値を設定した後、属性の指定されたフラグは TRUE となります。

構文

```
void setValue(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	属性の新しい値

~AttrRef

これはデフォルトのデストラクタです。

構文

```
~AttrRef();
```

CDATASectionRef インタフェース

表 15-3 では、CDATASectionRef インタフェースで使用できるメソッドを示しています。

表 15-3 CDATASectionRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-11 ページ「CDATASectionRef」	コンストラクタです。
15-11 ページ「~CDATASectionRef」	デフォルトのパブリック・デストラクタです。

CDATASectionRef

クラス・コンストラクタです。

構文	説明

`CDATASectionRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	`createCDATASection` をコールした後に、指定の CDATA ノードに対する参照を作成します。

`CDATASectionRef(` `  const CDATASectionRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
`node_ref`	コンテキストを提供するための参照
`nptr`	参照されるノード

戻り値

`(CDATASectionRef)` Node 参照オブジェクト

~CDATASectionRef

これはデフォルトのデストラクタです。

構文

```
~CDATASectionRef();
```

CharacterDataRef インタフェース

表 15-4 では、CharacterDataRef インタフェースで使用できるメソッドを示しています。

表 15-4 CharacterDataRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-12 ページ「appendData」	ノードの現行データの最後にデータを追加します。
15-13 ページ「deleteData」	ノードのデータの一部を削除します。
15-13 ページ「freeString」	substringData によって割り当てられた文字列の割当てを解除します。
15-14 ページ「getData」	ノードのデータを戻します。
15-14 ページ「getLength」	ノードのデータの長さを戻します。
15-14 ページ「insertData」	ノードの現行データに文字列を挿入します。
15-15 ページ「replaceData」	ノードのデータの一部を置換します。
15-16 ページ「setData」	ノードのデータを設定します。
15-16 ページ「substringData」	ノードのデータの部分文字列を取得します。

appendData

CharacterData ノードのデータの最後に文字列を追加します。追加されたデータはデータ・エンコーディングに含まれる必要があります。検証、変換、確認は行われません。

構文

```
void appendData(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	追加するデータ

deleteData

CharacterData ノードのデータから一定範囲の文字を削除します。オフセットは 0（ゼロ）から始まるため、オフセット 0（ゼロ）はデータの最初を参照します。オフセットおよびカウントはどちらも、バイト数ではなく文字数で表されます。オフセットとカウントの合計がデータ長を超える場合、オフセットからデータの最後までのすべての文字が削除されます。

構文

```
void deleteData(
   ub4 offset,
   ub4 count)
throw (DOMException);
```

パラメータ	説明
offset	削除の開始点となる文字オフセット
count	削除する文字数

freeString

substringData() によって割り当てられた文字列の割当てを解除します。これは Oracle の拡張機能です。

構文

```
void freeString(
   oratext* str);
```

パラメータ	説明
str	文字列

getData

（データ・エンコーディング内の）CharacterData ノードのデータ（テキスト、コメントまたは CDATA の型）を戻します。

構文

```
oratext* getData() const;
```

戻り値

(oratext*) ノードのデータ

getLength

CharacterData ノードのデータ（Text、Comment または CDATA の型）の長さを、バイト数ではなく文字数で戻します。

構文

```
ub4 getLength() const;
```

戻り値

(ub4) ノードのデータの長さ（文字数。バイト数ではない）

insertData

CharacterData ノードのデータの指定位置に文字列を挿入します。挿入されたデータはデータ・エンコーディングに含まれる必要があります。検証、変換、確認は行われません。オフセットは、バイト数ではなく文字数で指定されます。オフセットは 0（ゼロ）から始まるため、オフセット 0（ゼロ）の位置に挿入するとデータが追加されます。

構文

```
void insertData(
   ub4 offset,
   oratext* data)
throw (DOMException);
```

パラメータ	説明
offset	挿入の開始点となる文字オフセット
data	挿入するデータ

replaceData

CharacterData ノードのデータの一定範囲の文字を新規文字列で置換します。オフセットは 0（ゼロ）から始まるため、オフセット 0（ゼロ）はデータの最初を参照します。置換データはデータ・エンコーディングに含まれる必要があります。検証、変換、確認は行われません。オフセットおよびカウントはどちらも、バイト数ではなく文字数で表されます。オフセットとカウントの合計が長さを超える場合、データの最後までのすべての文字が置換されます。

構文

```
void replaceData(
   ub4 offset,
   ub4 count,
   oratext* data)
throw (DOMException);
```

パラメータ	説明
offset	offset
count	置換する文字数。
data	データ

setData

CharacterData ノードのデータ（テキスト、コメントまたは CDATA の型）を設定し、古いデータを置換します。新しいデータの検証、変換、確認は行われません。データ・エンコーディングに含まれる必要があります。

構文

```
void setData(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	データ

substringData

CharacterData ノードから一定範囲の文字データ（Text、Comment または CDATA の型）を戻します。データはデータ・エンコーディングに含まれるため、オフセットおよびカウントはバイト数ではなく文字数で表されます。文字列の先頭はオフセット 0（ゼロ）です。オフセットとカウントの合計が長さを超える場合、データの最後までのすべての文字が戻されます。部分文字列は、コンテキストにより管理されるメモリーに永続的に割り当てられており、freeString によって明示的に割当て解除される必要があります。

構文

```
oratext* substringData(
   ub4 offset,
   ub4 count)
throw (DOMException);
```

パラメータ	説明
offset	offset
count	抽出する文字数

戻り値

(oratext *) 指定された部分文字列

CommentRef インタフェース

表 15-5 では、CommentRef インタフェースで使用できるメソッドを示しています。

表 15-5 CommentRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-17 ページ「CommentRef」	コンストラクタです。
15-18 ページ「~CommentRef」	デフォルトのパブリック・デストラクタです。

CommentRef

クラス・コンストラクタです。

構文	説明

`CommentRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createComment をコールした後に、指定のコメント・ノードに対する参照を作成します。

`CommentRef(` `  const CommentRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(CommentRef) Node 参照オブジェクト

~CommentRef

これはデフォルトのデストラクタです。

構文

```
~CommentRef();
```

DOMException インタフェース

表 15-6 では、DOMException インタフェースで使用できるメソッドを示しています。

表 15-6 DOMException メソッドの概要 : DOM パッケージ

ファンクション	概要
15-19 ページ「getDOMCode」	例外に埋め込まれた DOM 例外コードを取得します。
15-19 ページ「getMesLang」	エラー・メッセージの現在の言語エンコーディングを取得します。
15-20 ページ「getMessage」	Oracle XML エラー・メッセージを取得します。

getDOMCode

これは、実行時の例外状況の DOM 例外コード（DOMExceptionCode で定義）を戻す実装で定義済みのメンバー・ファンクションのプロトタイプを定義した仮想メンバー・ファンクションです。

構文

```
virtual DOMExceptionCode getDOMCode() const = 0;
```

戻り値

(DOMExceptionCode) 例外コード

getMesLang

XmlException から継承された仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext*) エラー・メッセージの現在の言語（エンコーディング）

getMessage

XmlException から継承された仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext *) エラー・メッセージ

DOMImplRef インタフェース

表 15-7 では、DOMImplRef インタフェースで使用できるメソッドを示しています。

表 15-7 DOMImplRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-21 ページ「DOMImplRef」	コンストラクタです。
15-22 ページ「createDocument」	ドキュメント参照を作成します。
15-23 ページ「createDocumentType」	DTD 参照を作成します。
15-23 ページ「getImplementation」	ドキュメントに対応する DOMImplementation オブジェクトを取得します。
15-24 ページ「getNoMod」	no modification allowed フラグ値を取得します。
15-24 ページ「hasFeature」	DOM 機能が実装されているかどうかを判別します。
15-25 ページ「setContext」	ノードに別のコンテキストを設定します。
15-25 ページ「~DOMImplRef」	デフォルトのパブリック・デストラクタです。

DOMImplRef

クラス・コンストラクタです。

構文	説明

`DOMImplRef(`
`  Context* ctx_ptr,`
`  DOMImplementation< Node>* impl_ptr);`
	指定されたコンテキストにおいて、DOMImplementation オブジェクトに対する参照オブジェクトを作成します。インプリメンテーション・オブジェクトに対する参照を戻します。

`DOMImplRef(` `  const DOMImplRef< Context, Node>& iref);`	インプリメンテーション・オブジェクトに対する他の参照を作成する際に必要です。削除フラグはコピーされません。

`DOMImplRef(`
`  const DOMImplRef< Context, Node>& iref,`
`  Context* ctx_ptr);`
	別のコンテキストにおいてインプリメンテーション・オブジェクトに対する参照を作成する際に必要です。削除フラグはコピーされません。

パラメータ	説明
ctx_ptr	コンテキスト・ポインタ
impl_ptr	実装

戻り値

(DOMImplRef) インプリメンテーション・オブジェクトに対する参照

createDocument

ドキュメント参照を作成します。

構文

```
DocumentRef< Node>* createDocument(
   oratext* namespaceURI,
   oratext* qualifiedName,
   DocumentTypeRef< Node>& doctype)
throw (DOMException);
```

パラメータ	説明
namespaceURI	ルート要素の名前空間 URI
qualifiedName	ルート要素の修飾名
doctype	対応する DTD ノード

戻り値

(DocumentRef< Node>*) ドキュメント参照

createDocumentType

DTD 参照を作成します。

構文

```
DocumentTypeRef< Node>* createDocumentType(
   oratext* qualifiedName,
   oratext* publicId,
   oratext* systemId)
throw (DOMException);
```

パラメータ	説明
qualifiedName	修飾名
publicId	外部サブセットの公開識別子
systemId	外部サブセットのシステム識別子

戻り値

(DocumentTypeRef< Node>*) DTD 参照

getImplementation

このドキュメントの作成に使用された DOMImplementation オブジェクトを戻します。DOMImplementation オブジェクトが破棄された場合、これに対応するドキュメント・ツリーもすべて破棄されます。

構文

```
DOMImplementation< Node>* getImplementation() const;
```

戻り値

(DOMImplementation) DOMImplementation 参照オブジェクト

getNoMod

no modification allowed フラグ値を取得します。これは Oracle の拡張機能です。

構文

```
boolean getNoMod() const;
```

戻り値

フラグの値が TRUE の場合は TRUE、FALSE の場合は FALSE

hasFeature

DOM 機能が実装されているかどうかを判別します。機能が指定バージョンで実装されている場合は TRUE、実装されていない場合は FALSE を戻します。

レベル 1 では、パッケージの正当な値は「HTML」および「XML」（大 / 小文字は区別されない）であり、バージョンは文字列「1.0」です。バージョンが指定されていない場合は、この機能のすべてのバージョンがサポートされるため、TRUE を戻します。

DOM 1.0 の機能は「XML」および「HTML」です。

DOM 2.0 の機能は、「Core」、「XML」、「HTML」、「Views」、「StyleSheets」、「CSS」、「CSS2」、「Events」、「UIEvents」、「MouseEvents」、「MutationEvents」、「HTMLEvents」、「Range」、「Traversal」です。

構文

```
boolean hasFeature(
   oratext* feature,
   oratext* version);
```

パラメータ	説明
feature	機能のパッケージ名
version	パッケージのバージョン

戻り値

(boolean) 機能が実装されているかどうか

setContext

別のコンテキストにおいてノード参照を作成する際に必要です。

構文

```
void setContext(
   NodeRef< Node>& nref,
   Context* ctx_ptr);
```

パラメータ	説明
nref	参照ノード
ctx_ptr	コンテキスト・ポインタ

~DOMImplRef

これはデフォルトのデストラクタです。インプリメンテーション・オブジェクトに対する参照を消去します。通常は環境によってコールされます。ただし、必要に応じてユーザーが直接コールできます。

構文

```
~DOMImplRef();
```

DOMImplementation インタフェース

表 15-8 では、DOMImplementation インタフェースで使用できるメソッドを示しています。

表 15-8 DOMImplementation メソッドの概要 : DOM パッケージ

ファンクション	概要
15-26 ページ「DOMImplementation」	コンストラクタです。
15-27 ページ「getNoMod」	nomodificationallowed フラグ値を取得します。
15-27 ページ「~DOMImplementation」	デフォルトのパブリック・デストラクタです。

DOMImplementation

DOMImplementation オブジェクトを作成します。パラメータ値に no modifications allowed フラグを設定します。

構文

```
DOMImplementation(
   boolean no_mod);
```

パラメータ	説明
no_mod	変更が許可されているか（FALSE）許可されていないか（TRUE）

戻り値

(DOMImplementation) インプリメンテーション・オブジェクト

getNoMod

no modification allowed フラグ値を取得します。これは Oracle の拡張機能です。

構文

```
boolean getNoMod() const;
```

戻り値

フラグの値が TRUE の場合は TRUE、FALSE の場合は FALSE

~DOMImplementation

これはデフォルトのデストラクタです。このオブジェクトに対応するすべての DOM ツリーを削除します。

構文

```
~DOMImplementation();
```

DocumentFragmentRef インタフェース

表 15-9 では、DocumentFragmentRef インタフェースで使用できるメソッドを示しています。

表 15-9 DocumentFragmentRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-28 ページ「DocumentFragmentRef」	コンストラクタです。
15-29 ページ「~DocumentFragmentRef」	デフォルトのパブリック・デストラクタです。

DocumentFragmentRef

クラス・コンストラクタです。

構文	説明

`DocumentFragmentRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createDocumentFragment をコールした後に、指定のフラグメント・ノードに対する参照を作成します。

`DocumentFragmentRef(` `  const DocumentFragmentRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(DocumentFragmentRef) Node 参照オブジェクト

~DocumentFragmentRef

これはデフォルトのデストラクタです。

構文

```
~DocumentFragmentRef() {}
```

DocumentRange インタフェース

表 15-10 では、DocumentRange インタフェースで使用できるメソッドを示しています。

表 15-10 DocumentRange メソッドの概要 : DOM パッケージ

ファンクション	概要
15-30 ページ「DocumentRange」	コンストラクタです。
15-30 ページ「createRange」	新しい範囲オブジェクトを作成します。
15-31 ページ「destroyRange」	範囲オブジェクトを破棄します。
15-31 ページ「~DocumentRange」	デフォルトのデストラクタです。

DocumentRange

ファクトリを作成します。

構文

```
DocumentRange();
```

戻り値

(DocumentRange) 新しいファクトリ・オブジェクト

createRange

新しい範囲オブジェクトを作成します。

構文

```
Range< Node>* createRange(
   DocumentRef< Node>& doc);
```

パラメータ	説明
doc	ドキュメント・ノードに対する参照

戻り値

(Range*) 新しい範囲に対するポインタ

destroyRange

範囲オブジェクトを破棄します。

構文

```
void destroyRange(
   Range< Node>* range)
throw (DOMException);
```

パラメータ	説明
range	範囲

~DocumentRange

デフォルトのデストラクタです。

構文

```
~DocumentRange();
```

DocumentRef インタフェース

表 15-11 では、DocumentRef インタフェースで使用できるメソッドを示しています。

表 15-11 DocumentRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-33 ページ「DocumentRef」	コンストラクタです。
15-33 ページ「createAttribute」	属性ノードを作成します。
15-34 ページ「createAttributeNS」	名前空間の情報を含む属性ノードを作成します。
15-34 ページ「createCDATASection」	CDATA ノードを作成します。
15-35 ページ「createComment」	コメント・ノードを作成します。
15-35 ページ「createDocumentFragment」	ドキュメント・フラグメントを作成します。
15-36 ページ「createElement」	要素ノードを作成します。
15-36 ページ「createElementNS」	名前空間の情報を含む要素ノードを作成します。
15-37 ページ「createEntityReference」	実体参照ノードを作成します。
15-38 ページ「createProcessingInstruction」	`ProcessingInstruction` ノードを作成します。
15-38 ページ「createTextNode」	Text ノードを作成します。
15-39 ページ「getDoctype」	ドキュメントに対応する DTD を取得します。
15-39 ページ「getDocumentElement」	このドキュメントの最上位要素を取得します。
15-40 ページ「getElementById」	指定された ID の要素を取得します。
15-40 ページ「getElementsByTagName」	ドキュメント内の任意のタグ名の要素を取得します。
15-41 ページ「getElementsByTagNameNS」	ドキュメント内の任意のタグ名の要素を取得します（名前空間認識バージョン）。
15-42 ページ「getImplementation」	ドキュメントに対応する `DOMImplementation` オブジェクトを取得します。
15-42 ページ「importNode」	別の DOM からノードをインポートします。
15-43 ページ「~DocumentRef」	デフォルトのパブリック・デストラクタです。

DocumentRef

これはコンストラクタです。

構文	説明
`DocumentRef( const NodeRef< Node>& nref, Node* nptr);`	デフォルトのコンストラクタです。
`DocumentRef( const DocumentRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
nref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(DocumentRef) Node 参照オブジェクト

createAttribute

指定された名前の属性ノードを作成します。これは名前空間を認識しないファンクションです。新しい属性は、名前空間 URI および接頭辞が NULL となり、指定された名前が修飾名である場合も、属性のローカル名は属性名と同じになります。新しいノードは、親を持たない孤立した子です。名前はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

構文

```
Node* createAttribute(
   oratext* name)
throw (DOMException);
```

パラメータ	説明
name	名前

戻り値

(Node*) 新しい属性ノード

createAttributeNS

指定された名前空間 URI および修飾名を持つ属性ノードを作成します。新しいノードは、親を持たない孤立した子です。URI および修飾名はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

構文

```
Node* createAttributeNS(
   oratext* namespaceURI,
   oratext* qualifiedName)
   throw (DOMException);
```

パラメータ	説明
namespaceURI	名前空間 URI
qualifiedName	修飾名

戻り値

(Node*) 新しい属性ノード

createCDATASection

指定された初期データ（データ・エンコーディングに含まれる必要があります）を持つ CDATA セクション・ノードを作成します。CDATA セクションは文字どおりとみなされ、解析されません。つまり、正規化処理によって隣接した Text ノードと結合されることはありません。初期データは NULL にできます（指定する場合）。検証、変換、確認は行われません。CDATA ノードの名前は、常に「#cdata-section」となります。新しいノードは、親を持たない孤立した子です。CDATA はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

構文

```
Node* createCDATASection(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	新しいノードのデータ

戻り値

(Node*) 新しい CDATA ノード

createComment

指定された初期データ（データ・エンコーディングに含まれる必要があります）を持つコメント・ノードを作成します。このデータは NULL にできます（指定する場合）。検証、変換、確認は行われません。コメント・ノードの名前は、常に「#comment」となります。新しいノードは、親を持たない孤立した子です。コメント・データはコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

構文

```
Node* createComment(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	新しいノードのデータ

戻り値

(Node*) 新しいコメント・ノード

createDocumentFragment

空のドキュメント・フラグメント・ノードを作成します。ドキュメント・フラグメントは、DOM ツリーに挿入されると特殊な方法で処理されます。フラグメント・ノード自体ではなく、フラグメントの子が順番に挿入されます。挿入された後、フラグメント・ノードはまだ存在しますが、子はなくなります。フラグメント・ノードの名前は、常に「#document-fragment」となります。

構文

```
Node* createDocumentFragment()
throw (DOMException);
```

戻り値

(Node*) 新しいドキュメント・フラグメント・ノード

createElement

指定されたタグ名（データ・エンコーディングに含まれる必要があります）を持つ要素ノードを作成します。新しいノードは、親を持たない孤立した子です。tagname はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

要素のタグ名は大 / 小文字が区別されることに注意してください。これは名前空間を認識しないファンクションです。したがって、新しいノードは、名前空間 URI および接頭辞が NULL となり、指定されたタグ名が修飾名である場合も、ノードのローカル名はノードのタグ名と同じになります。

構文

```
Node* createElement(
   oratext* tagname)
throw (DOMException);
```

パラメータ	説明
tagname	タグ名

戻り値

(Node*) 新しい要素ノード

createElementNS

指定された名前空間 URI および修飾名を持つ要素を作成します。新しいノードは、親を持たない孤立した子です。URI および修飾名はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

要素名は大 / 小文字が区別され、URI は NULL にできますが修飾名は必須であることに注意してください。修飾名は、接頭辞部分とローカル部分に分割されます。tagName は完全な修飾名になります。

構文

```
Node* createElementNS(
   oratext* namespaceURI,
   oratext* qualifiedName)
throw (DOMException);
```

パラメータ	説明
namespaceURI	名前空間 URI
qualifiedName	修飾名

戻り値

(Node*) 新しい要素ノード

createEntityReference

実体参照ノードを作成します。名前（データ・エンコーディングに含まれる必要があります）は、参照される実体の名前です。名前付きの実体が存在する必要はありません。名前の検証、変換、確認は行われません。新しいノードは、親を持たない孤立した子です。実体参照名はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

実体参照ノードはパーサーによって生成されないことに注意してください。かわりに、実体参照は検出されると拡張されます。出力の際、実体参照ノードはスタイル参照の「&name;」に変換されます。

構文

```
Node* createEntityReference(
   oratext* name)
throw (DOMException);
```

パラメータ	説明
name	名前

戻り値

(Node*) 新しい実体参照ノード

createProcessingInstruction

指定されたターゲットおよびデータ（データ・エンコーディングに含まれる必要があります）を持つ処理命令ノードを作成します。このデータは NULL にできますが、ターゲットは必須であり、変更できません。ターゲットおよびデータの検証、変換、確認は行われません。ノードの名前はターゲットと同じです。新しいノードは、親を持たない孤立した子です。ターゲットおよびデータはコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

構文

```
Node* createProcessingInstruction(
   oratext* target,
   oratext* data)
throw (DOMException);
```

パラメータ	説明
target	ターゲット
data	新しいノードのデータ

戻り値

(Node*) 新しい処理命令ノード

createTextNode

指定された初期データ（非 NULL であり、データ・エンコーディングに含まれる必要があります）を持つ Text ノードを作成します。このデータは NULL にできます（指定する場合）。検証、変換、確認、解析は行われません（実体は拡張されません）。ノードの名前は、常に「#text」となります。新しいノードは、親を持たない孤立した子です。テキスト・データはコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

構文

```
Node* createTextNode(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	新しい Text ノードのデータ

戻り値

(Node*) 新しい Text ノード

getDoctype

このドキュメントに対応する DTD ノードを戻します。このコールの後、メンバー・ファンクションをコールするために、適切なコンストラクタを使用して DocumentTypeRef オブジェクトを作成する必要があります。DTD ツリーは編集できません。

構文

```
Node* getDoctype() const;
```

戻り値

(Node*) DTD ノード

getDocumentElement

DOM ツリーのルート要素（ノード）を戻します。各ドキュメントには、ルート要素と呼ばれる最上位の要素ノードが 1 つのみ含まれます。ルート要素がない場合、NULL が戻されます。これは、ドキュメント・ツリーが作成されている場合に発生することがあります。

構文

```
Node* getDocumentElement() const;
```

戻り値

(Node*) ルート要素

getElementById

指定された ID を持つ要素ノードを戻します。要素が検出されない場合は、NOT_FOUND_ERR が発生します。指定された ID はデータ・エンコーディングに含まれる必要があります。そうでないと一致しない場合があります。

ID という属性は自動的にタイプ ID とはならないことに注意してください。ID 属性（任意の名前を指定可能）は、ドキュメントに対応する DTD または XML Schema でタイプ ID として宣言する必要があります。

構文

```
Node* getElementById(
   oratext* elementId);
```

パラメータ	説明
elementId	要素 ID

戻り値

(Node*)Element ノード

getElementsByTagName

ドキュメント内の指定されたタグ名を持つすべての要素のリストを、ドキュメント順序（ツリーの先行順走査で検出された順序）で戻します。リストが不要になった場合は、ユーザーが解放する必要があります。このリストは最新のものではなく、スナップショットです。つまり、リストが戻された後に、タグ名と一致した新しいノードが DOM に追加された場合、新しいノードを含めるために、このリストが自動的に更新されることはありません。

特殊な名前「*」はすべてのタグ名と一致します。NULL の名前に一致するものはありません。タグ名は大 / 小文字が区別されることに注意してください。また、データ・エンコーディングに含まれる必要があり、そうでないと不一致が生じる場合があります。

このファンクションは名前空間を認識しません。したがって、完全なタグ名が比較されます。2 つの異なる接頭辞を持つ修飾名がどちらも同じ URI にマップされており、これらを比較した場合、比較は失敗します。

構文

```
NodeList< Node>* getElementsByTagName(
   oratext* tagname) const;
```

パラメータ	説明
tagname	タグ名

戻り値

(NodeList< Node>*) ノードのリスト

getElementsByTagNameNS

ドキュメント内の指定された名前空間 URI およびローカル名を持つすべての要素のリストを、ドキュメント順序（ツリーの先行順走査で検出された順序）で戻します。リストが不要になった場合は、ユーザーが解放する必要があります。このリストは最新のものではなく、スナップショットです。つまり、リストが戻された後に、URI およびローカル名と一致した新しいノードが DOM に追加された場合、新しいノードを含めるために、このリストが自動的に更新されることはありません。

URI およびローカル名は、データ・エンコーディングに含まれる必要があります。特殊な名前「*」はすべてのローカル名と一致します。NULL のローカル名に一致するものはありません。名前空間 URI は常に一致する必要がありますが、ワイルド・カードは使用できません。比較は大 / 小文字が区別されることに注意してください。

構文

```
NodeList< Node>* getElementsByTagNameNS(
   oratext* namespaceURI,
   oratext* localName);
```

パラメータ	説明
namespaceURI	名前空間 URI
localName	ローカル名

戻り値

(NodeList< Node>*) ノードのリスト

getImplementation

このドキュメントの作成に使用された DOMImplementation オブジェクトを戻します。DOMImplementation オブジェクトが破棄された場合、これに対応するドキュメント・ツリーもすべて破棄されます。

構文

```
DOMImplementation< Node>* getImplementation() const;
```

戻り値

(DOMImplementation) DOMImplementation 参照オブジェクト

importNode

あるドキュメントから別のドキュメントにノードをインポートします。新しいノードは孤立した子であり、親を持ちません。元のノードは、変更されたりドキュメントから削除されることはありません。かわりに、元のノードの修飾名、接頭辞、名前空間 URI およびローカル名のすべてのコピーによって、新しいノードが作成されます。

deep は、ノードの子が再帰的にインポートされるかどうかを制御します。FALSE の場合、ノード自体のみがインポートされ、子は含まれません。TRUE の場合、ノードのすべての子が同様にインポートされ、新しいサブツリー全体が作成されます。要素には指定された属性のみがインポートされます。指定されていない（デフォルトの）属性は省略されます。その後、インポート先ドキュメントの新しいデフォルトの属性が追加されます。ドキュメント・ノードおよび DocumentType ノードはインポートできません。

構文

```
Node* importNode(
   NodeRef< Node>& importedNode,
   boolean deep) const
throw (DOMException);
```

パラメータ	説明
importedNode	インポートされるノード
deep	ノードの子を再帰的にインポートするかどうかを制御

戻り値

(Node*) インポートされた新しいノード

~DocumentRef

これはデフォルトのデストラクタです。ノードに対する参照を消去します。ドキュメント・ノードが削除用にマークされている場合、このデストラクタがノードとその下のツリーを削除します。ドキュメント・ノードの場合は、常に深いレベルの削除となります。デストラクタは、環境からコールすることも、またユーザーが直接コールすることもできます。

構文

```
~DocumentRef();
```

DocumentTraversal インタフェース

表 15-12 では、DocumentTraversal インタフェースで使用できるメソッドを示しています。

表 15-12 DocumentTraversal メソッドの概要 : DOM パッケージ

ファンクション	概要
15-44 ページ「DocumentTraversal」	コンストラクタです。
15-45 ページ「createNodeIterator」	新しい NodeIterator オブジェクトを作成します。
15-45 ページ「createTreeWalker」	新しい TreeWalker オブジェクトを作成します。
15-46 ページ「destroyNodeIterator」	NodeIterator オブジェクトを破棄します。
15-46 ページ「destroyTreeWalker」	TreeWalker オブジェクトを破棄します。
15-46 ページ「~DocumentTraversal」	デフォルトのデストラクタです。

DocumentTraversal

ファクトリを作成します。

構文

```
DocumentTraversal();
```

戻り値

(DocumentTraversal) 新しいファクトリ・オブジェクト

createNodeIterator

新しいイテレータ・オブジェクトを作成します。

構文

```
NodeIterator< Node>* createNodeIterator(
   NodeRef< Node>& root,
   WhatToShowCode whatToShow,
   boolean entityReferenceExpansion)
throw (DOMException);
```

パラメータ	説明
root	反復用のサブツリーのルート
whatToShow	ノード・タイプ・フィルタ
entityReferenceExpansion	TRUE の場合、実体参照を拡張

戻り値

(NodeIterator*) 新しいイテレータに対するポインタ

createTreeWalker

新しい TreeWalker オブジェクトを作成します。

構文

```
TreeWalker< Node>* createTreeWalker(
   NodeRef< Node>& root,
   WhatToShowCode whatToShow,
   boolean entityReferenceExpansion)
throw (DOMException);
```

パラメータ	説明
root	横断用のサブツリーのルート
whatToShow	ノード・タイプ・フィルタ
entityReferenceExpansion	TRUE の場合、実体参照を拡張

戻り値

(TreeWalker*) 新しいツリー・ウォーカに対するポインタ

destroyNodeIterator

ノード・イテレータ・オブジェクトを破棄します。

構文

```
void destroyNodeIterator(
   NodeIterator< Node>* iter)
throw (DOMException);
```

パラメータ	説明
iter	イテレータ

destroyTreeWalker

TreeWalker オブジェクトを破棄します。

構文

```
void destroyTreeWalker(
   TreeWalker< Node>* walker)
throw (DOMException);
```

パラメータ	説明
walker	TreeWalker

~DocumentTraversal

デフォルトのデストラクタです。

構文

```
~DocumentTraversal();
```

DocumentTypeRef インタフェース

表 15-13 では、DocumentTypeRef インタフェースで使用できるメソッドを示しています。

表 15-13 DocumentTypeRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-47 ページ「DocumentTypeRef」	コンストラクタです。
15-48 ページ「getEntities」	DTD のエンティティを取得します。
15-48 ページ「getInternalSubset」	DTD の内部サブセットを取得します。
15-49 ページ「getName」	DTD の名前を取得します。
15-49 ページ「getNotations」	DTD の表記法を取得します。
15-49 ページ「getPublicId」	DTD の公開識別子を取得します。
15-50 ページ「getSystemId」	DTD のシステム識別子を取得します。
15-50 ページ「~DocumentTypeRef」	デフォルトのパブリック・デストラクタです。

DocumentTypeRef

これはコンストラクタです。

構文	説明

`DocumentTypeRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	デフォルトのコンストラクタです。

`DocumentTypeRef(` `  const DocumentTypeRef< Node>& node_ref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(DocumentTypeRef) Node 参照オブジェクト

getEntities

DTD によって定義された汎用エンティティの名前付きノード・マップを戻します。

構文

```
NamedNodeMap< Node>* getEntities() const;
```

戻り値

(NamedNodeMap< Node>*) エンティティを含むマップ

getInternalSubset

要素のコンテンツ・モデルを戻します。DTD が存在しない場合、NULL を戻します。

構文

```
Node* getInternalSubset(
   oratext* name);
```

パラメータ	説明
name	要素名。

戻り値

(xmlnode*) コンテンツ・モデル・サブツリー

getName

DOCTYPE キーワードの直後に指定された DTD の名前を戻します。

構文

```
oratext* getName() const;
```

戻り値

(oratext*) DTD の名前

getNotations

DTD によって宣言された表記法の名前付きノード・マップを戻します。

構文

```
NamedNodeMap< Node>* getNotations() const;
```

戻り値

(NamedNodeMap< Node>*) 表記法を含むマップ

getPublicId

DTD の公開識別子を戻します。

構文

```
oratext* getPublicId() const;
```

戻り値

(oratext*) DTD の公開識別子

getSystemId

DTD のシステム識別子を戻します。

構文

```
oratext* getSystemId() const;
```

戻り値

(oratext*) DTD のシステム識別子

~DocumentTypeRef

これはデフォルトのデストラクタです。

構文

```
~DocumentTypeRef();
```

ElementRef インタフェース

表 15-14 では、ElementRef インタフェースで使用できるメソッドを示しています。

表 15-14 ElementRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-52 ページ「ElementRef」	コンストラクタです。
15-52 ページ「getAttribute」	指定された名前の属性の値を取得します。
15-53 ページ「getAttributeNS」	指定された URI およびローカル名の属性の値を取得します。
15-53 ページ「getAttributeNode」	指定された名前の属性ノードを取得します。
15-54 ページ「getElementsByTagName」	指定されたタグ名の要素を取得します。
15-54 ページ「getTagName」	要素のタグ名を取得します。
15-55 ページ「hasAttribute」	指定された属性が存在するかどうかを確認します。
15-55 ページ「hasAttributeNS」	指定された属性が存在するかどうかを確認します（名前空間認識バージョン）。
15-56 ページ「removeAttribute」	指定された名前の属性を削除します。
15-56 ページ「removeAttributeNS」	指定された URI およびローカル名の属性を削除します。
15-57 ページ「removeAttributeNode」	属性ノードを削除します。
15-57 ページ「setAttribute」	この要素または新しい値（あるいはその両方）に対して新しい属性を設定します。
15-58 ページ「setAttributeNS」	要素または新しい値（あるいはその両方）に対して新しい属性を設定します。
15-58 ページ「setAttributeNode」	属性ノードを設定します。
15-59 ページ「~ElementRef」	デフォルトのパブリック・デストラクタです。

ElementRef

クラス・コンストラクタです。

構文	説明
`ElementRef( const NodeRef< Node>& node_ref, Node* nptr);`	createElement をコールした後に、指定の要素ノードに対する参照を作成します。
`ElementRef( const ElementRef< Node>& node_ref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(ElementRef) Node 参照オブジェクト

getAttribute

名前で指定された要素の属性の値を戻します。属性には値として空の文字列を使用できますが、NULL にはできない点に注意してください。

構文

```
oratext* getAttribute(
   oratext* name) const;
```

パラメータ	説明
name	属性の名前（データ・エンコーディング）

戻り値

(oratext*) 指定された属性の値（データ・エンコーディング内）

getAttributeNS

URI およびローカル名で指定された要素の属性の値を戻します。属性には値として空の文字列を使用できますが、NULL にはできない点に注意してください。

構文

```
oratext* getAttributeNS(
   oratext* namespaceURI,
   oratext* localName);
```

パラメータ	説明
namespaceURI	属性の名前空間 URI（データ・エンコーディング）
localName	属性のローカル名（データ・エンコーディング）

戻り値

(oratext*) 指定された属性の値（データ・エンコーディング内）

getAttributeNode

指定された名前の属性ノードを戻します。

構文

```
Node* getAttributeNode(
   oratext* name) const;
```

パラメータ	説明
name	属性の名前（データ・エンコーディング）

戻り値

(Node*) 属性ノード

getElementsByTagName

指定されたタグ名を持つすべての要素のリストを、サブツリーの先行順走査で検出した順に戻します。タグ名はデータ・エンコーディングに含まれる必要があります。特殊な名前「*」はすべてのタグ名と一致します。NULL の名前に一致するものはありません。タグ名では大 / 小文字が区別されます。このファンクションは名前空間を認識しません。したがって、完全なタグ名が比較されます。戻されたリストはユーザーが解放する必要があります。

構文

```
NodeList< Node>* getElementsByTagName(
   oratext* name);
```

パラメータ	説明
name	一致させるタグ名（データ・エンコーディング）

戻り値

(NodeList< Node>*) 要素のリスト

getTagName

ノード・インタフェースのノード名と同じ値を持つ要素ノードのタグ名を戻します。

構文

```
oratext* getTagName() const;
```

戻り値

(oratext*) 要素名（データ・エンコーディング内）

hasAttribute

指定された名前の属性が要素に設定されているかどうかを判別します。

構文

```
boolean hasAttribute(
   oratext* name);
```

パラメータ	説明
name	属性の名前（データ・エンコーディング）

戻り値

(boolean) 指定された名前の属性が要素に設定されている場合は TRUE

hasAttributeNS

指定された URI およびローカル名の属性が要素に設定されているかどうかを判別します。

構文

```
boolean hasAttributeNS(
   oratext* namespaceURI,
   oratext* localName);
```

パラメータ	説明
namespaceURI	属性の名前空間 URI（データ・エンコーディング）
localName	属性のローカル名（データ・エンコーディング）

戻り値

(boolean) 該当する属性が要素に設定されている場合は TRUE

removeAttribute

名前で指定された属性を削除します。属性は要素の属性リストから削除されますが、属性ノード自体は破棄されません。

構文

```
void removeAttribute(
   oratext* name) throw (DOMException);
```

パラメータ	説明
name	属性の名前（データ・エンコーディング）

removeAttributeNS

URI およびローカル名で指定された属性を削除します。属性は要素の属性リストから削除されますが、属性ノード自体は破棄されません。

構文

```
void removeAttributeNS(
   oratext* namespaceURI,
   oratext* localName)
throw (DOMException);
```

パラメータ	説明
namespaceURI	属性の名前空間 URI（データ・エンコーディング）
localName	属性のローカル名（データ・エンコーディング）

removeAttributeNode

要素から属性を削除します。削除された属性に対するポインタまたは NULL を戻します。

構文

```
Node* removeAttributeNode(
   AttrRef< Node>& oldAttr)
throw (DOMException);
```

パラメータ	説明
oldAttr	古い属性ノード

戻り値

(Node*) 古い属性ノードまたは NULL

setAttribute

指定された名前および値（データ・エンコーディングに含まれる必要があります）を持つ、要素の新しい属性を作成します。指定された属性名を持つ属性がすでに存在する場合は、単純にその値が置換されます。名前および値の検証、変換、確認は行われません。値は解析されないため、実体参照は拡張されません。

構文

```
void setAttribute(
   oratext* name,
   oratext* value)
throw (DOMException);
```

パラメータ	説明
name	属性の名前（データ・エンコーディング）
value	属性の値（データ・エンコーディング）

setAttributeNS

指定された URI、ローカル名および値（データ・エンコーディングに含まれる必要があります）を持つ、要素の新しい属性を作成します。指定された属性名を持つ属性がすでに存在する場合は、単純にその値が置換されます。名前および値の検証、変換、確認は行われません。値は解析されないため、実体参照は拡張されません。

構文

```
void setAttributeNS(
   oratext* namespaceURI,
   oratext* qualifiedName,
   oratext* value)
throw (DOMException);
```

パラメータ	説明
namespaceURI	属性の名前空間 URI（データ・エンコーディング）
qualifiedName	属性の修飾名（データ・エンコーディング）
value	属性の値（データ・エンコーディング）

setAttributeNode

要素に新しい属性を追加します。指定された名前の属性がすでに存在する場合、その属性は置換され、古い属性に対するポインタが戻されます。この属性が新しい場合、要素リストに追加され、新しい属性に対するポインタが戻されます。

構文

```
Node* setAttributeNode(
   AttrRef< Node>& newAttr)
throw (DOMException);
```

パラメータ	説明
newAttr	新しいノード

戻り値

(Node*) 古い、または新しい属性ノード

~ElementRef

これはデフォルトのデストラクタです。

構文

```
~ElementRef();
```

EntityRef インタフェース

表 15-15 では、EntityRef インタフェースで使用できるメソッドを示しています。

表 15-15 EntityRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-60 ページ「EntityRef」	コンストラクタです。
15-61 ページ「getNotationName」	エンティティの表記法を取得します。
15-61 ページ「getPublicId」	エンティティの公開識別子を取得します。
15-61 ページ「getSystemId」	エンティティのシステム識別子を取得します。
15-62 ページ「getType」	エンティティの型を取得します。
15-62 ページ「~EntityRef」	デフォルトのパブリック・デストラクタです。

EntityRef

クラス・コンストラクタです。

構文	説明

`EntityRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createEntity をコールした後に、指定のエンティティ・ノードに対する参照を作成します。

`EntityRef(` `  const EntityRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(EntityRef) Node 参照オブジェクト

getNotationName

解析対象外のエンティティの場合、表記法名（データ・エンコーディング内）を戻します。解析対象エンティティおよびその他のノード・タイプの場合は、NULL を戻します。

構文

```
oratext* getNotationName() const;
```

戻り値

(oratext*) エンティティの表記法

getPublicId

エンティティの公開識別子（データ・エンコーディング内）を戻します。

構文

```
oratext* getPublicId() const;
```

戻り値

(oratext*) エンティティの公開識別子

getSystemId

エンティティのシステム識別子（データ・エンコーディング内）を戻します。

構文

```
oratext* getSystemId() const;
```

戻り値

(oratext*) エンティティのシステム識別子

getType

エンティティが汎用であるか（TRUE）パラメータであるか（FALSE）を説明するブール値を戻します。

構文

```
boolean getType() const;
```

戻り値

(boolean) 汎用エンティティの場合は TRUE、パラメータ・エンティティの場合は FALSE

~EntityRef

これはデフォルトのデストラクタです。

構文

```
~EntityRef();
```

EntityReferenceRef インタフェース

表 15-16 では、EntityReferenceRef インタフェースで使用できるメソッドを示しています。

表 15-16 EntityReferenceRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-63 ページ「EntityReferenceRef」	コンストラクタです。
15-64 ページ「~EntityReferenceRef」	デフォルトのパブリック・デストラクタです。

EntityReferenceRef

クラス・コンストラクタです。

構文	説明

`EntityReferenceRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createEntityReference をコールした後に、指定の実体参照ノードに対する参照を作成します。

`EntityReferenceRef(` `  const EntityReferenceRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(EntityReferenceRef) Node 参照オブジェクト

~EntityReferenceRef

これはデフォルトのデストラクタです。

構文

```
~EntityReferenceRef();
```

NamedNodeMapRef インタフェース

表 15-17 では、NamedNodeMapRef インタフェースで使用できるメソッドを示しています。

表 15-17 NamedNodeMapRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-65 ページ「NamedNodeMapRef」	コンストラクタ
15-66 ページ「getLength」	マップの長さを取得します。
15-66 ページ「getNamedItem」	指定された名前の項目を取得します。
15-67 ページ「getNamedItemNS」	指定された名前空間 URI およびローカル名の項目を取得します。
15-67 ページ「item」	指定されたインデックスの項目を取得します。
15-68 ページ「removeNamedItem」	指定された名前の項目を削除します。
15-68 ページ「removeNamedItemNS」	マップから項目を削除します。
15-69 ページ「setNamedItem」	マップに新しい項目を追加します。
15-69 ページ「setNamedItemNS」	指定された項目をマップに設定します。
15-70 ページ「~NamedNodeMapRef」	デフォルトのデストラクタです。

NamedNodeMapRef

クラス・コンストラクタです。

構文	説明

`NamedNodeMapRef(`
`  const NodeRef< Node>& node_ref,`
`  NamedNodeMap< Node>* mptr);`
	指定の NamedNodeMap ノードに対する参照を作成します。

`NamedNodeMapRef(` `  const NamedNodeMapRef< Node>& mref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(NamedNodeMapRef) Node 参照オブジェクト

getLength

マップの長さを取得します。

構文

```
ub4 getLength() const;
```

戻り値

(ub4) マップの長さ

getNamedItem

指定された名前の項目の名前を取得します。

構文

```
Node* getNamedItem( oratext* name) const;
```

パラメータ	説明
name	項目の名前

戻り値

(Node*) 項目に対するポインタ

getNamedItemNS

指定された名前空間 URI およびローカル名の項目の名前を取得します。

構文

```
Node* getNamedItemNS(
   oratext* namespaceURI,
   oratext* localName) const;
```

パラメータ	説明
namespaceURI	項目の名前空間 URI
localName	項目のローカル名

戻り値

(Node*) 項目に対するポインタ

item

指定されたインデックスの項目を取得します。

構文

```
Node* item(
   ub4 index) const;
```

パラメータ	説明
index	項目のインデックス

戻り値

(Node*) 項目に対するポインタ

removeNamedItem

マップから指定された名前の項目を削除します。

構文

```
Node* removeNamedItem(
   oratext* name)
throw (DOMException);
```

パラメータ	説明
name	項目の名前

戻り値

(Node*) 削除された項目に対するポインタ

removeNamedItemNS

マップから、指定された名前空間 URI およびローカル名の項目を削除します。

構文

```
Node* removeNamedItemNS(
   oratext* namespaceURI,
   oratext* localName)
throw (DOMException);
```

パラメータ	説明
namespaceURI	項目の名前空間 URI
localName	項目のローカル名

戻り値

(Node*) 削除された項目に対するポインタ

setNamedItem

マップに新しい項目を追加します。

構文

```
Node* setNamedItem(
   NodeRef< Node>& newItem)
throw (DOMException);
```

パラメータ	説明
newItem	マップに設定する項目

戻り値

(Node*) 新しい項目に対するポインタ

setNamedItemNS

名前空間を認識する指定された項目をマップに設定します。

構文

```
Node* setNamedItemNS(
   NodeRef< Node>& newItem)
throw (DOMException);
```

パラメータ	説明
newItem	マップに設定する項目

戻り値

(Node*) 項目に対するポインタ

~NamedNodeMapRef

デフォルトのデストラクタです。

構文

```
~NamedNodeMapRef();
```

NodeFilter インタフェース

表 15-18 では、NodeFilter インタフェースで使用できるメソッドを示しています。

表 15-18 NodeFilter メソッドの概要 : DOM パッケージ

ファンクション	概要
15-71 ページ「acceptNode」	指定されたノードに対して実行し、その戻り値を使用します。

acceptNode

このファンクションは、NodeIterator および TreeWalker によってテストとして使用されます。

構文

```
template< typename Node> AcceptNodeCode AcceptNode(
   NodeRef< Node>& nref);
```

パラメータ	説明
nref	テストされるノードに対する参照

戻り値

(AcceptNodeCode) フィルタ・ファンクションにより戻された結果

NodeIterator インタフェース

表 15-19 では、NodeIterator インタフェースで使用できるメソッドを示しています。

表 15-19 NodeIterator メソッドの概要 : DOM パッケージ

ファンクション	概要
15-72 ページ「adjustCtx」	このイテレータを別のコンテキストに追加します。
15-72 ページ「detach」	イテレータを無効にします。
15-73 ページ「nextNode」	次のノードに移動します。
15-73 ページ「previousNode」	前のノードに移動します。

adjustCtx

指定されたノード参照に対応するコンテキストに、このイテレータを追加します。

構文

```
void adjustCtx(
   NodeRef< Node>& nref);
```

パラメータ	説明
nref	参照ノード

detach

イテレータを無効にします。

構文

```
void detach();
```

nextNode

次のノードに移動します。

構文

```
Node* nextNode() throw (DOMException);
```

戻り値

(Node*) 次のノードに対するポインタ

previousNode

前のノードに移動します。

構文

```
Node* previousNode() throw (DOMException);
```

戻り値

(Node*) 前のノードに対するポインタ

NodeListRef インタフェース

表 15-20 では、NodeListRef インタフェースで使用できるメソッドを示しています。

表 15-20 NodeListRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-74 ページ「NodeListRef」	コンストラクタです。
15-75 ページ「getLength」	リストの長さを取得します。
15-75 ページ「item」	指定されたインデックスの項目を取得します。
15-75 ページ「~NodeListRef」	デフォルトのデストラクタです。

NodeListRef

クラス・コンストラクタです。

構文	説明

`NodeListRef(`
`  const NodeRef< Node>& node_ref,`
`  NodeList< Node>* lptr);`
	指定の NodeList ノードに対する参照を作成します。

`NodeListRef(` `  const NodeListRef< Node>& lref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
lptr	参照されるリスト

戻り値

(NodeListRef) Node 参照オブジェクト

getLength

リストの長さを取得します。

構文

```
ub4 getLength() const;
```

戻り値

(ub4) リストの長さ

item

指定されたインデックスの項目を取得します。

構文

```
Node* item(
   ub4 index) const;
```

パラメータ	説明
index	項目のインデックス

戻り値

(Node*) 項目に対するポインタ

~NodeListRef

オブジェクトを破棄します。

構文

```
~NodeListRef();
```

NodeRef インタフェース

表 15-21 では、NodeRef インタフェースで使用できるメソッドを示しています。

表 15-21 NodeRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-77 ページ「NodeRef」	コンストラクタです。
15-78 ページ「appendChild」	ノードの子リストに新しい子を追加します。
15-78 ページ「cloneNode」	このノードを複製します。
15-79 ページ「getAttributes」	このノードの属性を取得します。
15-79 ページ「getChildNodes」	このノードの子を取得します。
15-80 ページ「getFirstChild」	このノードの最初の子ノードを取得します。
15-80 ページ「getLastChild」	このノードの最後の子ノードを取得します。
15-80 ページ「getLocalName」	このノードのローカル名を取得します。
15-81 ページ「getNamespaceURI」	このノードの名前空間 URI を、NULL で終了する文字列で取得します。
15-81 ページ「getNextSibling」	このノードの直後の兄弟関係ノードを取得します。
15-81 ページ「getNoMod」	このノードで変更が許可されていないかどうかをテストします。
15-82 ページ「getNodeName」	ノード名を、NULL で終了する文字列で取得します。
15-82 ページ「getNodeType」	ノードの DOMNodeType を取得します。
15-82 ページ「getNodeValue」	ノードの値を、NULL で終了する文字列で取得します。
15-83 ページ「getOwnerDocument」	このノードの所有者ドキュメントを取得します。
15-83 ページ「getParentNode」	このノードの親ノードを取得します。
15-83 ページ「getPrefix」	このノードの名前空間接頭辞を取得します。
15-84 ページ「getPreviousSibling」	このノードの直前の兄弟関係ノードを取得します。
15-84 ページ「hasAttributes」	このノードに属性が含まれるかどうかをテストします。
15-84 ページ「hasChildNodes」	このノードに子が含まれるかどうかをテストします。
15-85 ページ「insertBefore」	ノードの子リストに新しい子を挿入します。

表 15-21 NodeRef メソッドの概要 : DOM パッケージ（続き）

ファンクション	概要
15-85 ページ「isSupported」	指定された機能が実装によってサポートされているかどうかをテストします。
15-86 ページ「markToDelete」	参照されるノードを削除するためのマークを設定します。
15-86 ページ「normalize」	隣接した Text ノードをマージします。
15-86 ページ「removeChild」	既存の子ノードを削除します。
15-87 ページ「replaceChild」	ノードの既存の子を置換します。
15-87 ページ「resetNode」	別のノードを参照するように NodeRef をリセットします。
15-88 ページ「setNodeValue」	ノードの値を、NULL で終了する文字列で設定します。
15-88 ページ「setPrefix」	このノードの名前空間接頭辞を設定します。
15-89 ページ「~NodeRef」	デフォルトのパブリック・デストラクタです。

NodeRef

クラス・コンストラクタです。

構文	説明

`NodeRef(`
`  const NodeRef< Node>& nref,`
`  Node* nptr);`
	指定のノードまたは別のノードに対する少なくとも 1 つの参照がすでに使用可能である場合、この指定のノードに対する参照を作成します。ノード削除フラグはコピーされず、FALSE に設定されます。

`NodeRef(` `  const NodeRef< Node>& nref);`	コピー・コンストラクタです。少なくとも 1 つの参照がすでに使用可能である場合、ノードに対する追加参照を作成します。ノード削除フラグはコピーされず、FALSE に設定されます。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(NodeRef) Node 参照オブジェクト

appendChild

このノードの子リストの最後にノードを追加し、新しいノードを戻します。newChild が DocumentFragment である場合、その子はすべて元の順序で追加されます。DocumentFragment ノード自体は追加されません。newChild がすでに DOM ツリー内に存在する場合は、最初にその newChild が現在の位置から削除されます。

構文

```
Node* appendChild(
   NodeRef& newChild)
throw (DOMException);
```

パラメータ	説明
newChild	参照ノード

戻り値

(Node*) 追加されたノード

cloneNode

このノードの複製を作成して戻します。複製ノードは親を持ちません。要素を複製すると、XML プロセッサによって生成されたデフォルト値の属性を含む、すべての属性およびその値がコピーされます。ただし、ノードに含まれるテキストは、子である Text ノードに含まれているため、ディープ・クローンの場合以外はコピーされません。他のタイプのノードを複製すると、単純にノードのコピーが戻されます。deep が TRUE である場合、ノードの子はすべて再帰的に複製され、複製されたノードには複製された子が含まれます。非ディープ・クローンでは子は含まれません。複製されたノードが別のツリーまたはフラグメントに挿入されない場合、(ユーザーによる) 削除のために、参照を介してマークされる必要があります。

構文

```
Node* cloneNode(
   boolean deep);
```

パラメータ	説明
deep	ノードの下のノード階層全体を複製するか（TRUE）、またはノード自体のみを複製するか（FALSE）

戻り値

（Node*）複製ノード

getAttributes

このノードの属性の NamedNodeMap を戻します。属性が存在しない場合は、NULL を戻します。属性ノードを含めることができるのは、要素ノードのみです。その他の種類のノードでは、常に NULL が戻されます。現在の実装では、子ノードのノード・マップは最新のものです。元のノードのすべての変更がすぐに反映されます。このため、一部の DOM ツリー操作形式、特にマルチスレッド環境では、副次的な影響が出る場合があります。

構文

```
NamedNodeMap< Node>* getAttributes() const;
```

戻り値

（NamedNodeMap*）属性の NamedNodeMap

getChildNodes

子ノードのリストを戻します。このノードに子ノードが存在しない場合は、NULL を戻します。子が存在する可能性があるのは、要素ノード、ドキュメント・ノード、DTD ノードおよび DocumentFragment ノードのみです。他のすべての型では NULL が戻されます。現在の実装では、子ノードのリストは最新のものです。元のノードのすべての変更がすぐに反映されます。このため、一部の DOM ツリー操作形式、特にマルチスレッド環境では、副次的な影響が出る場合があります。

構文

```
NodeList< Node>* getChildNodes() const;
```

戻り値

（NodeList*）子ノードのリスト

getFirstChild

最初の子ノードを戻します。このノードに子ノードが存在しない場合は、NULL を戻します。

構文

```
Node* getFirstChild() const;
```

戻り値

(Node*) 最初の子ノードまたは NULL

getLastChild

最後の子ノードを戻します。このノードに子ノードが存在しない場合は、NULL を戻します。

構文

```
Node* getLastChild() const;
```

戻り値

(Node*) 最後の子ノードまたは NULL

getLocalName

このノード（データ・エンコーディング内）のローカル名（修飾名のローカル部分）を、NULL で終了する文字列で戻します。このノードの名前が完全修飾名でない（接頭辞がない）場合、ローカル名はこのノード名と同じです。

構文

```
oratext* getLocalName() const;
```

戻り値

(oratext*) このノードのローカル名

getNamespaceURI

このノードの名前空間 URI（データ・エンコーディング内）を、NULL で終了する文字列で戻します。ノードの名前が修飾されていない（名前空間接頭辞が含まれない）場合は、ノードの作成時に有効であったデフォルトの名前空間（NULL の場合があります）が含まれます。

構文

```
oratext* getNamespaceURI() const;
```

戻り値

(oratext*) このノードの名前空間 URI

getNextSibling

直後の兄弟関係ノードを戻します。このノードに直後の兄弟関係が存在しない場合は、NULL を戻します。

構文

```
Node* getNextSibling() const;
```

戻り値

(Node*) 直後の兄弟関係ノードまたは NULL

getNoMod

このノードおよびノードが属する DOM ツリーに変更が許可されていないかどうかをテストします。このメンバー・ファンクションは、Oracle の拡張機能です。

構文

```
boolean getNoMod() const;
```

戻り値

(boolean) 変更が許可されていない場合は TRUE

getNodeName

ノード（データ・エンコーディング内）の（完全修飾）名を、NULL で終了する文字列で戻します。「bar¥0」または「foo:bar¥0」などとなります。ノードの種類によっては、「#text」、「#cdata-section」、「#comment」、「#document」、「#document-fragment」といった固定ノード名が設定されています。ノードの名前は、1 度作成したら変更できません。

構文

```
oratext* getNodeName() const;
```

戻り値

(oratext*) データ・エンコーディング内のノードの名前

getNodeType

ノードの DOMNodeType を戻します。

構文

```
DOMNodeType getNodeType() const;
```

戻り値

ノードの (DOMNodeType)

getNodeValue

ノードの値（対応する文字データ）を、NULL で終了する文字列で戻します。文字および汎用エンティティは置換されます。値が設定されているのは、Attr、CDATA、Comment、ProcessingInstruction および Text ノードのみであり、他のすべてのノード・タイプには NULL 値が設定されています。

構文

```
oratext* getNodeValue() const;
```

戻り値

(oratext *) ノードの値

getOwnerDocument

このノードに対応するドキュメント・ノードを戻します。ドキュメント・ノードのタイプはノード・タイプから派生するものと想定されています。各ノードは 1 つのドキュメントにのみ属することができます。つまり、ユーザーのリクエストで作成された直後など、どのドキュメントにも一切関連付けることはできません。元のドキュメント（ノード）が戻されるか、または WRONG_DOCUMENT_ERR 例外が発生します。

構文

```
Node* getOwnerDocument() const throw (DOMException);
```

戻り値

(Node*) 元のドキュメント・ノード

getParentNode

親ノードを戻します。このノードに親ノードが存在しない場合は、NULL を戻します。

構文

```
Node* getParentNode() const;
```

戻り値

(Node*) 親ノードまたは NULL

getPrefix

このノードの名前空間接頭辞（データ・エンコーディング内）を、NULL で終了する文字列で戻します。このノードの名前が完全修飾名でない（接頭辞がない）場合、NULL が戻されます。

構文

```
oratext* getPrefix() const;
```

戻り値

(oratext*) このノードの名前空間接頭辞

getPreviousSibling

直前の兄弟関係ノードを戻します。このノードに直前の兄弟関係が存在しない場合は、NULL を戻します。

構文

```
Node* getPreviousSibling() const;
```

戻り値

(Node*) 直前の兄弟関係ノードまたは NULL

hasAttributes

このノードに属性が含まれ、このノードが要素である場合は、TRUE を戻します。それ以外の場合は FALSE を戻します。要素でないノードの場合は、常に FALSE を戻す点に注意してください。

構文

```
boolean hasAttributes() const;
```

戻り値

(boolean) このノードが要素であり、属性が含まれる場合は TRUE

hasChildNodes

このノードに子が含まれるかどうかをテストします。子が存在する可能性があるのは、Element、Document、DTD および DocumentFragment ノードのみです。

構文

```
boolean hasChildNodes() const;
```

戻り値

(boolean) このノードに子が存在する場合は TRUE

insertBefore

このノードの既存の子ノード refChild の前に newChild ノードを挿入します。refChild はこのノードの子である必要があります。newChild が DocumentFragment である場合、その子はすべて refChild の前に（同じ順序で）挿入されます。DocumentFragment ノード自体は挿入されません。newChild がすでに DOM ツリー内に存在する場合は、最初にその newChild が現在の位置から削除されます。

構文

```
Node* insertBefore(
   NodeRef& newChild,
   NodeRef& refChild)
throw (DOMException);
```

パラメータ	説明
newChild	新しいノード
refChild	参照ノード

戻り値

(Node*) 挿入されるノード

isSupported

引数によって指定された機能が、このノードの DOM インプリメンテーションによりサポートされているかどうかをテストします。

構文

```
boolean isSupported(
   oratext* feature,
   oratext* version) const;
```

パラメータ	説明
feature	機能のパッケージ名
version	パッケージのバージョン

戻り値

(boolean) 指定された機能がサポートされている場合は TRUE

markToDelete

参照されるノードが、この参照のデストラクタがコールされた時点で削除される必要があることを示すマークを設定します。このノードに対する他のすべての参照は無効になります。この動作は、他のすべての参照クラスによって継承されます。このメンバー・ファンクションは、Oracle の拡張機能です。

構文

```
void markToDelete();
```

normalize

要素をルートとしたサブツリーを正規化し、要素の隣接した Text 子ノードをマージします。隣接した Text ノードは通常の解析中には作成されず、DOM コールによるドキュメントの操作後にのみ作成されることに注意してください。

構文

```
void normalize();
```

removeChild

このノードの子リストからノードを削除し、削除されたノードを戻します。このノードは孤立した子です。削除後、親ノードは NULL となります。

構文

```
Node* removeChild(
   NodeRef& oldChild)
throw (DOMException);
```

パラメータ	説明
oldChild	古いノード

戻り値

(Node*) 削除されたノード

replaceChild

このノードの子リストで、子ノード oldChild を新しいノード newChild で置換し、oldChild（これは孤立した子となり、親は NULL です）を戻します。newChild が DocumentFragment である場合、その子はすべて oldChild のかわりに挿入されます。DocumentFragment ノード自体は挿入されません。newChild がすでに DOM ツリー内に存在する場合は、最初にその newChild が現在の位置から削除されます。

構文

```
Node* replaceChild(
   NodeRef& newChild,
   NodeRef& oldChild)
throw (DOMException);
```

パラメータ	説明
newChild	新しいノード
oldChild	古いノード

戻り値

(Node*) 置換されたノード

resetNode

このファンクションは、引数として指定された Node を参照するように、NodeRef をリセットします。

構文

```
void resetNode(
   Node* nptr);
```

パラメータ	説明
nptr	参照ノード

setNodeValue

ノードの値（文字データ）を NULL で終了する文字列で設定します。値を NULL に設定することはできません。値が設定されているのは、Attr、CDATA、Comment、ProcessingInstruction および Text ノードのみです。別の種類のノードの値を設定しようとしても動作しません。新しい値はデータ・エンコーディングに含まれる必要があります。検証、変換、確認は行われません。値はコピーされず、ポインタが格納されるのみです。ノードのデータの持続および解放は、ユーザーが行います。

変更が許可されていない場合は NO_MODIFICATION_ALLOWED_ERR 例外が発生し、実装で定義されたエラーの場合は、UNDEFINED_ERR および適切な Oracle XML エラー・コード（xml.h を参照）が発生します。

構文

```
void setNodeValue(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	ノードの新しい値

setPrefix

このノードの名前空間接頭辞を、NULL で終了する文字列で設定します。接頭辞が定義済みかどうかは検証されません。また、接頭辞が現在のデータ・エンコーディングに含まれるかどうかも検証されません。新しい接頭辞および古いローカル名から新しい修飾名が作成されるのみです。

変更が許可されていない場合は、NO_MODIFICATION_ALLOWED_ERR 例外が発生します。このノードの namespaceURI が NULL である場合、または指定された接頭辞が「xml」であり、このノードの namespaceURI が「http://www.w3.org/XML/1998/namespace」と異なる場合、またはこのノードが属性であり、指定された接頭辞が「xmlns」であり、このノードの namespaceURI が「http://www.w3.org/2000/xmlns/」と異なる場合、NAMESPACE_ERR が発生します。接頭辞の作成方法は確認されないため、INVALID_CHARACTER_ERR 例外は発生しないことに注意してください。

構文

```
void setPrefix(
   oratext* prefix)
throw (DOMException);
```

パラメータ	説明
prefix	新しい名前空間接頭辞

~NodeRef

これはデフォルトのデストラクタです。ノードに対する参照を消去し、ノードが削除用にマークされている場合、はノードを削除します。ノードが深いレベルの削除用にマークされている場合は、ノードの下のツリーも削除（割当て解除）されます。通常は環境によってコールされます。ただし、必要に応じてユーザーが直接コールできます。

構文

```
~NodeRef();
```

NotationRef インタフェース

表 15-22 では、NotationRef インタフェースで使用できるメソッドを示しています。

表 15-22 NotationRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-90 ページ「NotationRef」	コンストラクタです。
15-91 ページ「getPublicId」	公開識別子を取得します。
15-91 ページ「getSystemId」	システム識別子を取得します。
15-91 ページ「~NotationRef」	デフォルトのパブリック・デストラクタです。

NotationRef

クラス・コンストラクタです。

構文	説明

`NotationRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createNotation をコールした後に、指定の表記法ノードに対する参照を作成します。

`NotationRef(` `  const NotationRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(NotationRef) Node 参照オブジェクト

getPublicId

公開識別子を取得します。

構文

```
oratext* getPublicId() const;
```

戻り値

(oratext*) 公開識別子

getSystemId

システム識別子を取得します。

構文

```
oratext* getSystemId() const;
```

戻り値

(oratext*) システム識別子

~NotationRef

これはデフォルトのデストラクタです。

構文

```
~NotationRef();
```

ProcessingInstructionRef インタフェース

表 15-23 では、ProcessingInstructionRef インタフェースで使用できるメソッドを示しています。

表 15-23 ProcessingInstructionRef メソッドの概要 : DOM パッケージ

ファンクション	概要
15-92 ページ「ProcessingInstructionRef」	コンストラクタです。
15-93 ページ「getData」	処理命令のデータを取得します。
15-93 ページ「getTarget」	処理命令のターゲットを取得します。
15-94 ページ「setData」	処理命令のデータを設定します。
15-94 ページ「~ProcessingInstructionRef」	デフォルトのパブリック・デストラクタです。

ProcessingInstructionRef

クラス・コンストラクタです。

構文	説明

`ProcessingInstructionRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	ProcessingInstruction の作成をコールした後に、指定の ProcessingInstruction ノードに対する参照を作成します。

`ProcessingInstructionRef(` `  const ProcessingInstructionRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(ProcessingInstructionRef) Node 参照オブジェクト

getData

処理命令（データ・エンコーディング内）のコンテンツ（データ）を戻します。コンテンツは、ターゲットの後の最初の空白以外の文字から、最後の「?>」までの部分です。

構文

```
oratext* getData() const;
```

戻り値

(oratext*) 処理命令のデータ

getTarget

処理命令のターゲット文字列を戻します。ターゲットは、ProcessingInstruction を開始するマークアップの後に続く最初のトークンです。すべての ProcessingInstruction にはターゲットが含まれる必要がありますが、データ部分はオプションです。

構文

```
oratext* getTarget() const;
```

戻り値

(oratext*) 処理命令のターゲット

setData

処理命令のデータ（コンテンツ）を設定します。これは、データ・エンコーディングに含まれる必要があります。データを NULL に設定することはできません。新しいデータの検証、変換、確認は行われません。

構文

```
void setData(
   oratext* data)
throw (DOMException);
```

パラメータ	説明
data	新しいデータ。

~ProcessingInstructionRef

これはデフォルトのデストラクタです。

構文

```
~ProcessingInstructionRef();
```

Range インタフェース

表 15-24 では、Range インタフェースで使用できるメソッドを示しています。

表 15-24 Range メソッドの概要 : DOM パッケージ

ファンクション	概要
15-96 ページ「CompareBoundaryPoints」	
15-96 ページ「cloneContent」	
15-97 ページ「cloneRange」	
15-97 ページ「deleteContents」	
15-97 ページ「detach」	範囲を無効にします。
15-97 ページ「extractContent」	
15-98 ページ「getCollapsed」	範囲が縮小されたかどうかを確認します。
15-98 ページ「getCommonAncestorContainer」	最も深い共通の祖先ノードを取得します。
15-98 ページ「getEndContainer」	終了コンテナ・ノードを取得します。
15-99 ページ「getEndOffset」	エンド・ポイントのオフセットを取得します。
15-99 ページ「getStartContainer」	開始コンテナ・ノードを取得します。
15-100 ページ「getStartOffset」	開始ポイントのオフセットを取得します。
15-100 ページ「insertNode」	
15-100 ページ「selectNodeContent」	
15-100 ページ「selectNode」	
15-101 ページ「setEnd」	エンド・ポイントを設定します。
15-101 ページ「setEndAfter」	
15-102 ページ「setEndBefore」	
15-102 ページ「setStart」	開始ポイントを設定します。
15-103 ページ「setStartAfter」	
15-103 ページ「setStartBefore」	

表 15-24 Range メソッドの概要 : DOM パッケージ（続き）

ファンクション	概要
15-104 ページ「surroundContents」	
15-104 ページ「toString」	

CompareBoundaryPoints

境界点を比較します。

構文

```
CompareHowCode compareBoundaryPoints(
   unsigned short how,
   Range< Node>* sourceRange)
throw (DOMException);
```

パラメータ	説明
how	比較方法
sourceRange	比較の範囲

戻り値

(CompareHowCode) 比較の結果

cloneContent

子を含むノードの複製を作成します。

構文

```
Node* cloneContents() throw (DOMException);
```

戻り値

(Node*) 複製されたサブツリー

cloneRange

一定範囲のノードを複製します。

構文

```
Range< Node>* cloneRange();
```

戻り値

(Range< Node>*) 複製された新しい範囲

deleteContents

ノードのコンテンツを削除します。

構文

```
void deleteContents() throw (DOMException);
```

detach

範囲を無効にします。オブジェクトが無効な状態になるため、このメソッドの使用はお薦めしません。デストラクタをコールする方法をお薦めします。

構文

```
void detach();
```

extractContent

ノードを抽出します。

構文

```
Node* extractContents() throw (DOMException);
```

戻り値

(Node*) 抽出されたサブツリー

getCollapsed

範囲が縮小されたかどうかを確認します。

構文

```
boolean getCollapsed() const;
```

戻り値

(boolean) 範囲が縮小されている場合は TRUE、縮小されていない場合は FALSE

getCommonAncestorContainer

ノードの最も深い共通の祖先を取得します。

構文

```
Node* getCommonAncestorContainer() const;
```

戻り値

(Node*) 共通の祖先ノード

getEndContainer

コンテナ・ノードを取得します。

構文

```
Node* getEndContainer() const;
```

戻り値

(Node*) 終了コンテナ・ノード

getEndOffset

エンド・ポイントのオフセットを取得します。

構文

```
long getEndOffset() const;
```

戻り値

(long) オフセット

getStartContainer

開始コンテナ・ノードを取得します。

構文

```
Node* getStartContainer() const;
```

戻り値

(Node*) 開始コンテナ・ノード

getStartOffset

開始ポイントのオフセットを取得します。

構文

```
long getStartOffset() const;
```

戻り値

(long) オフセット

insertNode

ノードを挿入します。

構文

```
void insertNode(
   NodeRef< Node>& newNode)
throw (RangeException, DOMException);
```

パラメータ	説明
newNode	挿入するノード

selectNodeContent

参照によりノード・コンテンツを選択します。

構文

```
void selectNodeContent(
   NodeRef< Node>& refNode)
throw (RangeException);
```

パラメータ	説明
refNode	参照ノード

selectNode

ノードを選択します。

構文

```
void selectNode(
   NodeRef< Node>& refNode)
throw (RangeException);
```

パラメータ	説明
refNode	参照ノード

setEnd

エンド・ポイントを設定します。

構文

```
void setEnd(
   NodeRef< Node>& refNode,
   long offset)
throw (RangeException, DOMException);
```

パラメータ	説明
refNode	参照ノード
offset	offset

setEndAfter

指定されたノードの後に終了ポインタを設定します。

構文

```
void setEndAfter(
   NodeRef< Node>& refNode)
throw (RangeException);
```

パラメータ	説明
refNode	参照ノード

setEndBefore

指定されたノードの前に終了を設定します。

構文

```
void setEndBefore(
   NodeRef< Node>& refNode)
throw (RangeException);
```

パラメータ	説明
refNode	参照ノード

setStart

開始ポイントを設定します。

構文

```
void setStart(
   NodeRef< Node>& refNode,
   long offset)
throw (RangeException, DOMException);
```

パラメータ	説明
refNode	参照ノード
offset	offset

setStartAfter

指定されたノードの後に開始ポインタを設定します。

構文

```
void setStartAfter(
   NodeRef< Node>& refNode)
throw (RangeException);
```

パラメータ	説明
refNode	参照ノード

setStartBefore

指定されたノードの前に開始ポインタを設定します。

構文

```
void setStartBefore(
   NodeRef< Node>& refNode)
throw (RangeException);
```

パラメータ	説明
refNode	参照ノード

surroundContents

あるノードを指定されたノードの子にします。

構文

```
void surroundContents(
   NodeRef< Node>& newParent)
throw (RangeException, DOMException);
```

パラメータ	説明
newParent	親ノード

toString

項目を文字列に変換します。

構文

```
oratext* toString();
```

戻り値

(oratext*) 範囲の文字列表現

RangeException インタフェース

表 15-25 では、RangeException インタフェースで使用できるメソッドを示しています。

表 15-25 RangeException メソッドの概要 : DOM パッケージ

ファンクション	概要
15-105 ページ「getCode」	例外に埋め込まれた Oracle XML エラー・コードを取得します。
15-105 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
15-106 ページ「getMessage」	Oracle XML エラー・メッセージを取得します。
15-106 ページ「getRangeCode」	例外に埋め込まれた範囲例外コードを取得します。

getCode

例外に埋め込まれた Oracle XML エラー・コードを取得します。XmlException から継承された仮想メンバー・ファンクションです。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード（正常に終了した場合は 0）

getMesLang

エラー・メッセージの現在の言語エンコーディングを取得します。XmlException から継承された仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext*) エラー・メッセージの現在の言語（エンコーディング）

getMessage

XML エラー・メッセージを取得します。XmlException から継承された仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext *) エラー・メッセージ

getRangeCode

これは、実行時の例外状況の範囲例外コード（RangeExceptionCode で定義）を戻す実装で定義済みのメンバー・ファンクションのプロトタイプを定義した仮想メンバー・ファンクションです。

構文

```
virtual RangeExceptionCode getRangeCode() const = 0;
```

戻り値

(RangeExceptionCode) 例外コード

TextRef インタフェース

表 15-26 では、TextRef インタフェースで使用できるメソッドを示しています。

表 15-26 NodeIterator メソッドの概要 : DOM パッケージ

ファンクション	概要
15-107 ページ「TextRef」	コンストラクタです。
15-108 ページ「splitText」	Text ノードを 2 つに分割します。
15-108 ページ「~TextRef」	デフォルトのパブリック・デストラクタです。

TextRef

クラス・コンストラクタです。

構文	説明

`TextRef(`
`  const NodeRef< Node>& node_ref,`
`  Node* nptr);`
	createtext をコールした後に、指定の Text ノードに対する参照を作成します。

`TextRef(` `  const TextRef< Node>& nref);`	コピー・コンストラクタです。

パラメータ	説明
node_ref	コンテキストを提供するための参照
nptr	参照されるノード

戻り値

(TextRef) Node 参照オブジェクト

splitText

1 つの Text ノードを 2 つの Text ノードに分割します。元のデータはこの 2 つに分割されます。オフセットは 0（ゼロ）から始まり、バイト数ではなく文字数で表されます。元のノードは保持され、ノードのデータは切り捨てられます。元のデータの残りの部分を含む新しい Text ノードが作成され、元のデータの直後の兄弟関係として挿入されます。新しい Text ノードが戻されます。

構文

```
Node* splitText(
   ub4 offset)
throw (DOMException);
```

パラメータ	説明
offset	テキストの分割点となる文字オフセット

戻り値

(Node*) 新しいノード

~TextRef

これはデフォルトのデストラクタです。

構文

```
~TextRef();
```

TreeWalker インタフェース

表 15-27 では、TreeWalker インタフェースで使用できるメソッドを示しています。

表 15-27 TreeWalker メソッドの概要 : DOM パッケージ

ファンクション	概要
15-109 ページ「adjustCtx」	このツリー・ウォーカを別のコンテキストに追加します。
15-110 ページ「firstChild」	現在のノードの最初の子を取得します。
15-110 ページ「lastChild」	現在のノードの最後の子を取得します。
15-110 ページ「nextNode」	次のノードを取得します。
15-111 ページ「nextSibling」	直後の兄弟関係ノードを取得します。
15-111 ページ「parentNode」	現在のノードの親を取得します。
15-111 ページ「previousNode」	前のノードを取得します。
15-112 ページ「previousSibling」	直前の兄弟関係ノードを取得します。

adjustCtx

指定されたノード参照に対応するコンテキストに、このツリー・ウォーカを追加します。

構文

```
void adjustCtx(
   NodeRef< Node>& nref);
```

パラメータ	説明
nref	コンテキストを提供するための参照

firstChild

現在のノードの最初の子を取得します。

構文

```
Node* firstChild();
```

戻り値

(Node*) 最初の子ノードに対するポインタ

lastChild

現在のノードの最後の子を取得します。

構文

```
Node* lastChild();
```

戻り値

(Node*) 最後の子ノードに対するポインタ

nextNode

次のノードを取得します。

構文

```
Node* nextNode();
```

戻り値

(Node*) 次のノードに対するポインタ

nextSibling

直後の兄弟関係ノードを取得します。

構文

```
Node* nextSibling();
```

戻り値

(Node*) 直後の兄弟関係ノードに対するポインタ

parentNode

現在のノードの親を取得します。

構文

```
Node* parentNode();
```

戻り値

(Node*) 親ノードに対するポインタ

previousNode

前のノードを取得します。

構文

```
Node* previousNode();
```

戻り値

(Node*) 前のノードに対するポインタ

previousSibling

直前の兄弟関係ノードを取得します。

構文

```
Node* previousSibling();
```

戻り値

(Node*) 直前の兄弟関係ノードに対するポインタ

16

C++ 用の IO API パッケージ

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

IO のデータ型

表 16-1 に IO パッケージのデータ型の概要を示します。

表 16-1 データ型の概要 : IO パッケージ

データ型	説明
16-2 ページ「InputSourceType」	入力ソース・タイプを定義します。

InputSourceType

入力ソース・タイプを定義します。

定義

```
typedef enum InputSourceType {
   ISRC_URI = 1,
   ISRC_FILE = 2,
   ISRC_BUFFER = 3,
   ISRC_DOM = 4,
   ISRC_CSTREAM = 5 }
InputSourceType;
```

InputSource インタフェース

表 16-2 では、IO インタフェースで使用できるメソッドを示しています。

表 16-2 IO パッケージ・インタフェースの概要

ファンクション	概要
16-3 ページ「getBaseURI」	ベース URI を取得します。
16-4 ページ「setBaseURI」	ベース URI を設定します。

getBaseURI

ベース URI を取得します。ファイルや URI などの一部の入力ソースにより使用されます。

構文

```
oratext* getBaseURI() { return baseURI; }
```

戻り値

(oratext*) ベース URI。

getISrcType

入力ソース・タイプを取得します。

構文

```
InputSourceType getISrcType() const { return isrctype; }
```

戻り値

(InputSourceType) 入力ソース・タイプ。

setBaseURI

ベース URI を設定します。ファイルや URI などの一部の入力ソースにより使用されます。

構文

```
void setBaseURI( oratext* base_URI)  baseURI = base_URI; }
```

17

C++ 用の OracleXml API パッケージ

OracleXml は XML C++ のすべてのインタフェースの名前空間です。XML のすべての例外のルート、クラス `XmlException` とこれらの名前空間が含まれます。

- Ctx: TCtx 関連の宣言のための名前空間。第 14 章「C++ 用の Ctx API パッケージ」で説明しています。
- Dom: DOM 関連の宣言のための名前空間。第 15 章「C++ 用の DOM API パッケージ」で説明しています。
- IO: 入力ソースと出力ソース宣言のための名前空間。第 16 章「C++ 用の IO API パッケージ」で説明しています。
- パーサー : パーサーとスキーマのバリデータ宣言のための名前空間。第 18 章「C++ 用のパーサー API パッケージ」で説明しています。
- ツール : Tools::Factory 関連の宣言のための名前空間。第 19 章「C++ 用のツール API パッケージ」で説明しています。
- XPath: XPath 関連の宣言のための名前空間。第 20 章「C++ 用の XPath API パッケージ」で説明しています。
- XPointer: XPointer 関連の宣言のための名前空間。第 21 章「C++ 用の XPointer API パッケージ」で説明しています。
- Xsl: XSLT 関連の宣言のための名前空間。第 22 章「C++ 用の Xsl API パッケージ」で説明しています。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XmlException インタフェース

XMLException はすべての XML 例外のルート・インタフェースです。表 17-1 に OracleXml パッケージで使用できるメソッドの概要を示します。

表 17-1 OracleXml パッケージ・インタフェースの概要

ファンクション	概要
17-2 ページ「getCode」	例外に埋め込まれた Oracle XML エラー・コードを取得します。
17-3 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
17-3 ページ「getMessage」	Oracle XML エラー・メッセージを取得します。

getCode

これは実行中に発生した例外状況の Oracle XML エラー・コード（xml.h で定義されるエラー・コードなど）を戻す、実装で定義されたファンクションのプロトタイプを定義する仮想メンバー・ファンクションです。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード（成功時は 0）。

getMesLang

これは実行中に発生した例外状況にあるエラー・メッセージの現在の言語（エンコーディング）を戻す、ユーザー定義のファンクションのプロトタイプを定義する仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext *) エラー・メッセージの現在の言語（エンコーディング）。

getMessage

これは実行中に発生した例外状況の Oracle XML エラー・メッセージを戻す、実装で定義されたファンクションのプロトタイプを定義する仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext*) エラー・メッセージ。

18

C++ 用のパーサー API パッケージ

パーサー・インタフェースには、パーサー例外、バリデータ、パーサー、DOMParser、SAXParser が含まれます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

パーサーのデータ型

表 18-1 に Parser パッケージのデータ型の概要を示します。

表 18-1 データ型の概要 : パーサー・パッケージ

データ型	説明
18-2 ページ「ParserExceptionCode」	例外のパーサー実装。
18-3 ページ「DOMParserIdType」	パーサーの識別子を定義します。
18-3 ページ「SAXParserIdType」	ノードのタイプを定義します。
18-3 ページ「SchValidatorIdType」	バリデータの識別子を定義します。

ParserExceptionCode

例外のパーサー実装。

定義

```
typedef enum ParserExceptionCode {
   PARSER_UNDEFINED_ERR = 0,
   PARSER_VALIDATION_ERR = 1,
   PARSER_VALIDATOR_ERR = 2,
   PARSER_BAD_ISOURCE_ERR = 3,
   PARSER_CONTEXT_ERR = 4,
   PARSER_PARAMETER_ERR = 5,
   PARSER_PARSE_ERR = 6,
   PARSER_SAXHANDLER_SET_ERR = 7,
   PARSER_VALIDATOR_SET_ERR = 8 }
ParserExceptionCode;
```

DOMParserIdType

パーサーの識別子を定義します。

定義

```
typedef enum DOMParserIdType {
      DOMParCXml         = 1
    } DOMParserIdType;
ypedef enum CompareHowCode {
   START_TO_START = 0,
   START_TO_END = 1,
   END_TO_END = 2,
   END_TO_START = 3 }
CompareHowCode;
```

SAXParserIdType

パーサーの識別子を定義します。

定義

```
typedef enum SAXParserIdType {
   SAXParCXml = 1 }
SAXParserIdType;
```

SchValidatorIdType

バリデータの識別子を定義します。これらの識別子は、特定のバリデータ・オブジェクトを作成する必要がある場合に、XML ツールのファクトリのパラメータとして使用されます。

定義

```
typedef enum SchValidatorIdType {
   SchValCXml        = 1
} SchValidatorIdType;
```

DOMParser インタフェース

表 18-2 では、DOMParser インタフェースで使用できるメソッドを示しています。

表 18-2 DOMParser メソッドの概要 : パーサー・パッケージ

ファンクション	概要
18-4 ページ「getContext」	パーサーの XML コンテキスト（割当てとエンコーディング）を戻します。
18-4 ページ「getParserId」	パーサー ID を取得します。
18-5 ページ「parse」	ドキュメントを解析します。
18-6 ページ「parseDTD」	DTD ドキュメントを解析します。
18-6 ページ「parseSchVal」	ドキュメントを解析し、検証します。
18-7 ページ「setValidator」	このパーサーのバリデータを設定します。

getContext

各パーサー・オブジェクトは特定の Oracle XML コンテキスト内で割り当てられ、実行されます。このメンバー・ファンクションは、このコンテキストのポインタを戻します。

構文

```
virtual Context* getContext() const = 0;
```

戻り値

(Context*) パーサーのコンテキストのポインタ。

getParserId

構文

```
virtual DOMParserIdType getParserId() const = 0;
```

戻り値

(DOMParserIdType) パーサー ID。

parse

ドキュメントを解析し、ツリーのルート・ノードを戻します。

構文

```
virtual DocumentRef< Node>* parse(
   InputSource* isrc_ptr,
   boolean DTDvalidate = FALSE,
   DocumentTypeRef< Node>* dtd_ptr = NULL,
   boolean no_mod = FALSE,
   DOMImplementation< Node>* impl_ptr = NULL)
throw (ParserException) = 0;
```

パラメータ	説明
isrc_ptr	入力ソース
DTDvalidate	DTD で検証された場合は TRUE
dtd_ptr	DTD 参照
no_mod	変更が許可されていない場合は TRUE
impl_ptr	オプションの DomImplementation ポインタ

戻り値

(DocumentRef) ドキュメント・ツリー。

parseDTD

DTD ドキュメントを解析します。

構文

```
virtual DocumentRef< Node>* parseDTD(
   InputSource* isrc_ptr,
   boolean no_mod = FALSE,
   DOMImplementation< Node>* impl_ptr = NULL)
throw (ParserException) = 0;
```

パラメータ	説明
isrc_ptr	入力ソース
no_mod	変更が許可されていない場合は TRUE
impl_ptr	オプションの DomImplementation ポインタ

戻り値

(DocumentRef) DTD ドキュメント・ツリー。

parseSchVal

ドキュメントを解析し、検証します。対応するパラメータが NULL 以外の場合、バリデータを設定します。

構文

```
virtual DocumentRef< Node>* parseSchVal(
   InputSource* src_par,
   boolean no_mod = FALSE,
   DOMImplementation< Node>* impl_ptr = NULL,
   SchemaValidator< Node>* tor_ptr = NULL)
throw (ParserException) = 0;
```

パラメータ	説明
isrc_ptr	入力ソース
no_mod	変更が許可されていない場合は TRUE
impl_ptr	オプションの DomImplementation ポインタ
tor_ptr	スキーマ・バリデータ

戻り値

(DocumentRef) ドキュメント・ツリー。

setValidator

parseSchVal で別のバリデータが指定されている場合を除き、すべての検証についてバリデータを設定します。

構文

```
virtual void setValidator(
SchemaValidator< Node>* tor_ptr) = 0;
```

パラメータ	説明
tor_ptr	スキーマ・バリデータ

Gparser インタフェース

表 18-3 では、GParser インタフェースで使用できるメソッドを示しています。

表 18-3 GParser メソッドの概要 : パーサー・パッケージ

ファンクション	概要
18-9 ページ「SetWarnDuplicateEntity」	複数のエンティティ宣言に対し警告を発生させるかどうかを指定します。
18-9 ページ「getBaseURI」	ドキュメントのベース URI を戻します。
18-10 ページ「getDiscardWhitespaces」	要素間の空白が破棄されているかどうかを確認します。
18-10 ページ「getExpandCharRefs」	文字参照が拡張されているかどうかを確認します。
18-10 ページ「getSchemaLocation」	ドキュメントのスキーマの位置を取得します。
18-11 ページ「getStopOnWarning」	警告の発生時にドキュメント処理が停止されるかどうかを取得します。
18-11 ページ「getWarnDuplicateEntity」	複数のエンティティ宣言に対し警告が発生するかどうかを取得します。
18-11 ページ「setBaseURI」	ドキュメントのベース URI を設定します。
18-12 ページ「setDiscardWhitespaces」	書式の空白を破棄するかどうかを設定します。
18-12 ページ「setExpandCharRefs」	文字参照を拡張するかどうかを設定します。
18-13 ページ「setSchemaLocation」	ドキュメントのスキーマの位置を設定します。
18-13 ページ「setStopOnWarning」	警告の発生時にドキュメント処理を停止させるかどうかを設定します。

SetWarnDuplicateEntity

エンティティが 2 回以上宣言された場合に警告を発生させるかどうかを指定します。

構文

```
void setWarnDuplicateEntity(
   boolean par_bool);
```

パラメータ	説明
par_bool	複数のエンティティ宣言により警告を発生させる場合は TRUE

getBaseURI

ドキュメントのベース URI を戻します。通常は、URI からロードされたドキュメントにのみ、ベース URI が自動的に設定されます。他のソース（stdin、バッファなど）からロードされたドキュメントには通常、ベース URI は設定されていませんが、相対 URI を解決するために、setBaseURI を使用してベース URI が設定されている場合もあります。

構文

```
oratext* getBaseURI() const;
```

戻り値

(oratext *) 現在のドキュメントのベース URI（または NULL）。

getDiscardWhitespaces

入力ドキュメントの改行やインデントなど、要素間の書式の空白が破棄されているかどうかを確認します。デフォルトでは、すべての入力文字が維持されます。

構文

```
boolean getDiscardWhitespaces() const;
```

戻り値

(boolean) 要素間の空白が破棄されている場合は TRUE。

getExpandCharRefs

DOM データで文字参照が拡張されているかどうかを確認します。デフォルトでは、文字参照は参照が表す文字に置き換えられます。ただしドキュメントが保存されると、それらの文字エンティティは再度表示されません。ロードと保存を通じて維持するためには、文字参照を拡張しません。

構文

```
boolean getExpandCharRefs() const;
```

戻り値

(boolean) 文字参照が拡張されている場合は TRUE。

getSchemaLocation

ドキュメントのスキーマの位置を取得します。データベースにドキュメントをロードするときに、最適なレイアウトを判断するために使用します。

構文

```
oratext* getSchemaLocation() const;
```

戻り値

(oratext*) スキーマの位置。

getStopOnWarning

TRUE が戻される場合、警告はエラーと同様に処理され、解析、検証などは即時停止します。デフォルトでは、警告が発行されても処理は続けられます。

構文

```
boolean getStopOnWarning() const;
```

戻り値

(boolean) 警告の発生時にドキュメント処理を停止させる場合は TRUE。

getWarnDuplicateEntity

エンティティが 2 回以上宣言された場合に警告が発生するかどうかを取得します。

構文

```
boolean getWarnDuplicateEntity() const;
```

戻り値

(boolean) 複数のエンティティ宣言により警告が発生する場合は TRUE。

setBaseURI

ドキュメントのベース URI を設定します。通常は、URI からロードされたドキュメントにのみ、ベース URI が自動的に設定されます。他のソース（stdin、バッファなど）からロードされたドキュメントには通常、ベース URI は設定されませんが、相対 URI を解決するために、setBaseURI を使用してベース URI を設定する場合があります。

構文

```
void setBaseURI( oratext* par);
```

パラメータ	説明
par	ベース URI

setDiscardWhitespaces

入力ドキュメントの要素（改行とインデント）間の書式の空白を破棄するかどうかを設定します。デフォルトでは、すべての入力文字が維持されます。

構文

```
void setDiscardWhitespaces(
   boolean par_bool);
```

パラメータ	説明
par_bool	空白を破棄する場合は TRUE

setExpandCharRefs

DOM データで文字参照を拡張するかどうかを設定します。通常は、文字参照は参照が表す文字に置き換えられます。ただしドキュメントが保存されると、それらの文字エンティティは再度表示されません。ロードと保存を通じて維持するためには、文字参照を拡張しません。

構文

```
void setExpandCharRefs(
   boolean par_bool);
```

パラメータ	説明
par_bool	文字参照を破棄する場合は TRUE

setSchemaLocation

ドキュメントのスキーマの位置を設定します。データベースにドキュメントをロードするときに、最適なレイアウトを判断するために使用します。

構文

```
void setSchemaLocation(
  oratext* par);
```

パラメータ	説明
par	スキーマの位置

setStopOnWarning

TRUE が設定されると、警告はエラーと同様に処理され、解析、検証などは即時停止します。デフォルトでは、警告が発行されても処理は続けられます。

構文

```
void setStopOnWarning(
   boolean par_bool);
```

パラメータ	説明
par_bool	警告によりドキュメント処理を停止させる場合は TRUE

ParserException インタフェース

表 18-4 では、ParserException インタフェースで使用できるメソッドを示しています。

表 18-4 ParserException メソッドの概要 : パーサー・パッケージ

ファンクション	概要
18-14 ページ「getCode」	例外に埋め込まれた Oracle XML エラー・コードを取得します。
18-14 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
18-15 ページ「getMessage」	Oracle XML エラー・メッセージを取得します。
18-15 ページ「getParserCode」	例外に埋め込まれたパーサー例外コードを取得します。

getCode

XmlException から継承された仮想メンバー・ファンクション。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード（成功時は 0）。

getMesLang

XmlException から継承された仮想メンバー・ファンクション。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext *) エラー・メッセージの現在の言語（エンコーディング）。

getMessage

XmlException から継承された仮想メンバー・ファンクション。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext*) エラー・メッセージ。

getParserCode

実行中に発生した例外状況の、ParserExceptionCode で定義されたパーサーおよびバリデータの例外コードを戻す、実装で定義されたメンバー・ファンクションのプロトタイプを定義する仮想メンバー・ファンクション。

構文

```
virtual ParserExceptionCode getParserCode() const = 0;
```

戻り値

(ParserExceptionCode) 例外コード。

SAXHandler インタフェース

表 18-5 では、SAXHandler インタフェースで使用できるメソッドを示しています。

表 18-5 SAXHandler メソッドの概要 : パーサー・パッケージ

ファンクション	概要
18-17 ページ「CDATA」	CDATA の通知を受け取ります。
18-17 ページ「XMLDecl」	XML 宣言の通知を受け取ります。
18-18 ページ「attributeDecl」	属性の宣言の通知を受け取ります。
18-18 ページ「characters」	文字データの通知を受け取ります。
18-19 ページ「comment」	コメントの通知を受け取ります。
18-19 ページ「elementDecl」	要素の宣言の通知を受け取ります。
18-19 ページ「endDocument」	ドキュメントの終わりの通知を受け取ります。
18-20 ページ「endElement」	要素の終わりの通知を受け取ります。
18-20 ページ「notationDecl」	表記法宣言の通知を受け取ります。
18-21 ページ「parsedEntityDecl」	解析対象エンティティ宣言の通知を受け取ります。
18-22 ページ「processingInstruction」	処理命令の通知を受け取ります。
18-22 ページ「startDocument」	ドキュメントの始まりの通知を受け取ります。
18-22 ページ「startElement」	要素の始まりの通知を受け取ります。
18-23 ページ「startElementNS」	要素の始まりに対する、名前空間を認識する通知を受け取ります。
18-23 ページ「unparsedEntityDecl」	解析対象外のエンティティ宣言の通知を受け取ります。
18-24 ページ「whitespace」	空白文字の通知を受け取ります。

CDATA

このイベントは CDATA を Text と異なるものとして処理します。データはデータ・エンコーディングされ、戻される長さはバイトではなく文字数です。これは Oracle 拡張機能です。

構文

```
virtual void CDATA(
   oratext* data,
   ub4 size) = 0;
```

パラメータ	説明
data	CDATA のポインタ
size	CDATA のサイズ

XMLDecl

このイベントは XML 宣言（XMLDecl）をマークします。startDocument イベントが常に最初に処理され、このイベントは 2 番目のイベントになります。エンコーディング・フラグにより、エンコードが指定されているかどうかが示されます。スタンドアロン・フラグの場合、フラグが指定されていない場合は -1 が戻されます。それ以外の場合、FALSE は 0、TRUE は 1 が戻されます。このメンバー・ファンクションは Oracle 拡張機能です。

構文

```
virtual void XMLDecl(
   oratext* version,
   boolean is_encoding,
   sword standalone) = 0;
```

パラメータ	説明
version	XMLDecl のバージョン文字列
is_encoding	エンコーディングが指定されているか否かを指定
standalone	スタンドアロン値フラグの値

attributeDecl

このイベントは DTD の属性宣言をマークします。Oracle 拡張機能であり、SAX 標準ではありません。

構文

```
virtual void attributeDecl(
   oratext* attr_name,
   oratext *name,
   oratext *content) = 0;
```

パラメータ	説明
attr_name	属性名
name	名前
content	宣言される属性の本体

characters

このイベントは文字データをマークします。

構文

```
virtual void characters(
   oratext* ch,
   ub4 size) = 0;
```

パラメータ	説明
ch	データのポインタ
size	データの長さ

comment

このイベントは XML 文書のコメントをマークします。コメントのデータはデータ・エンコーディングされています。Oracle 拡張機能であり、SAX 標準ではありません。

構文

```
virtual void comment(
   oratext* data) = 0;
```

パラメータ	説明
data	コメントのデータ

elementDecl

このイベントは DTD の要素宣言をマークします。Oracle 拡張機能であり、SAX 標準ではありません。

構文

```
virtual void elementDecl(
   oratext *name,
   oratext *content) = 0;
```

パラメータ	説明
name	要素名
content	要素の内容

endDocument

ドキュメントの終わりの通知を受け取ります。

構文

```
virtual void endDocument() = 0;
```

endElement

このイベントは要素の終わりをマークします。名前は要素の tagName（名前空間を認識した要素の場合は修飾名になる場合があります）であり、データ・コーディングされます。

構文

```
virtual void endElement( oratext* name) = 0;
```

notationDecl

このイベントは DTD の表記法の宣言をマークします。表記法名、公開識別子、およびシステム識別子はすべてデータ・エンコーディングに含まれます。いずれの ID もオプションで、NULL でもかまいません。

構文

```
virtual void notationDecl(
   oratext* name,
   oratext* public_id,
   oratext* system_id) = 0;
```

パラメータ	説明
name	表記法名
public_id	表記法の公開識別子
sysem_id	表記法のシステム識別子

parsedEntityDecl

DTD の解析対象のエンティティ宣言をマークします。解析対象のエンティティ名、公開識別子、システム識別子および表記法名はすべてデータ・エンコーディングに含まれます。これは Oracle 拡張機能です。

構文

```
virtual void parsedEntityDecl(
   oratext* name,
   oratext* value,
   oratext* public_id,
   oratext* system_id,
   boolean general) = 0;
```

パラメータ	説明
name	エンティティ名
value	内部エンティティの場合はエンティティの値
public_id	エンティティの公開識別子
sysem_id	エンティティのシステム識別子
general	汎用エンティティか否かの指定（パラメータ・エンティティの場合は FALSE）

processingInstruction

このイベントは処理命令をマークします。PI のターゲットとデータはデータ・エンコーディングに含まれます。ターゲットは必須ですが、データは NULL でもかまいません。

構文

```
virtual void processingInstruction(
   oratext* target,
   oratext* data) = 0;
```

パラメータ	説明
target	PI のターゲット
data	PI のデータ

startDocument

ドキュメントの始まりの通知を受け取ります。

構文

```
virtual void startDocument() = 0;
```

startElement

このイベントは要素の開始をマークします。

構文

```
virtual void startElement(
   oratext* name,
   NodeListRef< Node>* attrs_ptr) = 0;
```

パラメータ	説明
name	要素名
attrs_ptr	要素の属性のリスト

startElementNS

このイベントは要素の開始をマークします。これは名前空間を認識する新しいバージョンの SAX 2 であることに注意してください。要素の修飾名、ローカル名、名前空間 URI はすべての属性パートと同様に、データ・エンコーディングに含まれます。

構文

```
virtual void startElementNS(
   oratext* qname,
   oratext* local,
   oratext* ns_URI,
   NodeListRef< Node>* attrs_ptr) = 0;
```

パラメータ	説明
qname	要素の修飾名
local	要素の名前空間のローカル名
ns_URI	要素の名前空間 URI
attrs_ref	要素の属性の NodeList

unparsedEntityDecl

DTD の解析対象外エンティティ宣言をマークします。解析対象外のエンティティ名、公開識別子、システム識別子および表記法名はすべてデータ・エンコーディングに含まれます。

構文

```
virtual void unparsedEntityDecl(
   oratext* name,
   oratext* public_id,
   oratext* system_id,
   oratext* notation_name) = 0;
};
```

パラメータ	説明
name	エンティティ名
public_id	エンティティの公開識別子
sysem_id	エンティティのシステム識別子
notation_name	エンティティの表記法名

whitespace

このイベントは改行、行間のインデントなどの無視できる空白データをマークします。

構文

```
virtual void whitespace(
   oratext* data,
   ub4 size) = 0;
```

パラメータ	説明
data	データのポインタ
size	データの長さ

SAXParser インタフェース

表 18-6 では、SAXParser インタフェースで使用できるメソッドを示しています。

表 18-6 SAXParser メソッドの概要 : パーサー・パッケージ

ファンクション	概要
18-25 ページ「getContext」	パーサーの XML コンテキスト（割当てとエンコーディング）を戻します。
18-26 ページ「getParserId」	パーサー ID を戻します。
18-26 ページ「parse」	ドキュメントを解析します。
18-27 ページ「parseDTD」	DTD を解析します。
18-27 ページ「setSAXHandler」	SAX ハンドラを設定します。

getContext

各パーサー・オブジェクトは特定の Oracle XML コンテキスト内で割り当てられ、実行されます。このメンバー・ファンクションは、このコンテキストのポインタを戻します。

構文

```
virtual Context* getContext() const = 0;
```

戻り値

(Context*) パーサーのコンテキストのポインタ。

getParserId

パーサー ID を戻します。

構文

```
virtual SAXParserIdType getParserId() const = 0;
```

戻り値

(SAXParserIdType) パーサー ID。

parse

ドキュメントを解析します。

構文

```
virtual void parse(
   InputSource* src_ptr,
   boolean DTDvalidate = FALSE,
   SAXHandlerRoot* hdlr_ptr = NULL)
throw (ParserException) = 0;
```

パラメータ	説明
src_ptr	入力ソース
DTDValidate	DTD で検証された場合は TRUE
hdlr_ptr	SAX ハンドラ・ポインタ

parseDTD

DTD を解析します。

構文

```
virtual void parseDTD(
   InputSource* src_ptr,
   SAXHandlerRoot* hdlr_ptr = NULL)
throw (ParserException) = 0;
```

パラメータ	説明
src_ptr	入力ソース
hdlr_ptr	SAX ハンドラ・ポインタ

setSAXHandler

パーサー・コールで別の SAX ハンドラが指定されている場合を除き、すべてのパーサー起動の SAX ハンドラを設定します。

構文

```
virtual void setSAXHandler(
   SAXHandlerRoot* hdlr_ptr) = 0;
```

パラメータ	説明
hdlr_ptr	SAX ハンドラ・ポインタ

SchemaValidator インタフェース

表 18-7 では、SchemaValidator インタフェースで使用できるメソッドを示しています。

表 18-7 SchemaValidator メソッドの概要 : パーサー・パッケージ

ファンクション	概要
18-28 ページ「getSchemaList」	スキーマ・リストを戻します。
18-29 ページ「getValidatorId」	バリデータ識別子を取得します。
18-29 ページ「loadSchema」	スキーマ・ドキュメントをロードします。
18-30 ページ「unloadSchema」	スキーマ・ドキュメントをアンロードします。

getSchemaList

「リスト」が NULL の場合、ロードされたスキーマ・リスト・ドキュメントのサイズのみを戻します。「リスト」が NULL 以外の場合、ユーザー側のポインタ・バッファで URL ポインタのリストが戻されます。十分な大きさのバッファを用意するのは、ユーザー側の責任であることに注意してください。

構文

```
virtual ub4 getSchemaList(
   oratext **list) const = 0;
```

パラメータ	説明
list	ポインタ・バッファのアドレス

戻り値

(ub4) リスト・サイズとロードされたスキーマのリスト (I/O パラメータ)。

getValidatorId

このバリデータ・オブジェクトの実装に対応するバリデータ識別子を取得します。

構文

```
virtual SchValidatorIdType getValidatorId() const = 0;
```

戻り値

(SchValidatorIdType) バリデータ識別子。

loadSchema

次の検証セッションで使用されるスキーマ・ドキュメントをロードします。エラーの場合は、例外を発生させます。

構文

```
virtual void loadSchema(
   oratext* schema_URI)
throw (ParserException) = 0;
```

パラメータ	説明
schema_URI	スキーマ・ドキュメントの URL。コンパイラ・エンコーディング

unloadSchema

スキーマ・ドキュメントとその（バリデータからネスト方式で追加された、またはインポートされた）すべての子孫をアンロードします。以前にロードされたすべてのスキーマ・ドキュメントは、アンロードされるまでロードされた状態です。ロードされたすべてのスキーマ・ドキュメントをアンロードするためには、schema_URI が NULL になるように設定します。エラーの場合は、例外を発生させます。

構文

```
virtual void unloadSchema(
   oratext* schema_URI)
throw (ParserException) = 0;
```

パラメータ	説明
schema_URI	スキーマ・ドキュメントの URL。コンパイラ・エンコーディング

19

C++ 用のツール API パッケージ

Tools パッケージには、Oracle XML ツールの作成とインスタンス化に関連した型とインタフェースが含まれます。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

ツールのデータ型

表 19-1 に Tools パッケージのデータ型の概要を示します。

表 19-1 データ型の概要 : ツール・パッケージ

データ型	説明
19-2 ページ「FactoryExceptionCode」	Factory ツールの例外。

FactoryExceptionCode

Factory ツールの例外。

定義

```
typedef enum FactoryExceptionCode {
   FACTORY_UNDEFINED_ERR = 0,
   FACTORY_OTHER_ERR = 1
} FactoryExceptionCode;
```

Factory インタフェース

表 19-2 に Factory インタフェースで使用できるメソッドの概要を示します。

表 19-2 Factory メソッドの概要 : ツール・パッケージ

ファンクション	概要
19-4 ページ「Factory」	コンストラクタです。
19-4 ページ「createDOMParser」	DOM パーサーを作成します。
19-5 ページ「createSAXParser」	SAX パーサーを作成します。
19-5 ページ「createSchemaValidator」	スキーマ・バリデータを作成します。
19-6 ページ「createXPathCompProcessor」	拡張 XPath プロセッサを作成します。
19-6 ページ「createXPathCompiler」	XPath コンパイラを作成します。
19-7 ページ「createXPathProcessor」	XPath プロセッサを作成します。
19-7 ページ「createXPointerProcessor」	XPointer プロセッサを作成します。
19-8 ページ「createXslCompiler」	Xsl コンパイラを作成します。
19-8 ページ「createXslExtendedTransformer」	XSL 拡張トランスフォーマを作成します。
19-9 ページ「createXslTransformer」	XSL トランスフォーマを作成します。
19-9 ページ「getContext」	ファクトリのコンテキストを取得します。
19-9 ページ「~Factory」	デフォルト・デストラクタ

Factory

クラス・コンストラクタです。

構文	説明
`Factory()` `throw (FactoryException);`	デフォルトのコンストラクタ

`Factory(`
`  Context* ctx_ptr)`
`throw (FactoryException);`
	コンテキスト・オブジェクトに指定されたファクトリ・オブジェクトを作成します。

パラメータ	説明
`ctx_ptr`	コンテキスト・オブジェクトのポインタ

戻り値

`(Factory)` オブジェクト。

createDOMParser

DOM パーサーを作成します。

構文

```
DOMParser< Context, Node>* createDOMParser (
   DOMParserIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
`id_type`	パーサー ID の型
`ctx_ptr`	コンテキスト・オブジェクトのポインタ

戻り値

`(DOMParser*)` パーサー・オブジェクトのポインタ

createSAXParser

SAX パーサーを作成します。

構文

```
SAXParser< Context>* createSAXParser (
   SAXParserIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	パーサー ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(SAXParser*) パーサー・オブジェクトのポインタ。

createSchemaValidator

スキーマ・バリデータを作成します。

構文

```
SchemaValidator< Node>* createSchemaValidator (
   SchValidatorIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	バリデータ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(SchemaValidator*) バリデータ・オブジェクトのポインタ。

createXPathCompProcessor

拡張 XPath プロセッサを作成します。値 XvmPrCXml のみをとります。

構文

```
CompProcessor< Context, Node>* createXPathCompProcessor (
   XPathPrIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	プロセッサ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(CompProcessor*) プロセッサ・オブジェクトのポインタ。

createXPathCompiler

XPath コンパイラを作成します。

構文

```
XPath::Compiler< Context, Node>* createXPathCompiler (
   XPathCompIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	コンパイラ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(XPathCompiler*) コンパイラ・オブジェクトのポインタ。

createXPathProcessor

XPath プロセッサを作成します。

構文

```
XPath::Processor< Context, Node>* createXPathProcessor (
   XPathPrIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	プロセッサ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(Processor*) プロセッサ・オブジェクトのポインタ。

createXPointerProcessor

XPointer プロセッサを作成します。

構文

```
XPointer::Processor< Context, Node>* createXPointerProcessor (
   XppPrIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	プロセッサ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(Processor*) プロセッサ・オブジェクトのポインタ。

createXslCompiler

Xsl コンパイラを作成します。

構文

```
Xsl::Compiler< Context, Node>* createXslCompiler (
   XslCompIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	コンパイラ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(Compiler*) コンパイラ・オブジェクトのポインタ。

createXslExtendedTransformer

XSL 拡張トランスフォーマを作成します。値 XvmTrCXml のみをとります。

構文

```
CompTransformer< Context, Node>* createXslExtendedTransformer (
   XslTrIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	トランスフォーマ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(CompTrasformer*) トランスフォーマ・オブジェクトのポインタ。

createXslTransformer

XSL トランスフォーマを作成します。

構文

```
Transformer< Context, Node>* createXslTransformer (
   XslTrIdType id_type,
   Context* ctx_ptr = NULL)
throw (FactoryException);
```

パラメータ	説明
id_type	トランスフォーマ ID の型
ctx_ptr	コンテキスト・オブジェクトのポインタ

戻り値

(Trasformer*) トランスフォーマ・オブジェクトのポインタ。

getContext

ファクトリのコンテキストを戻します。

構文

```
Context* getContext() const;
```

戻り値

(Context*) コンテキスト・オブジェクトのポインタ。

~Factory

デフォルト・デストラクタ。

構文

```
~Factory();
```

FactoryException インタフェース

表 19-3 に FactoryException インタフェースで使用できるメソッドの概要を示します。

表 19-3 FactoryException メソッドの概要 : ツール・パッケージ

ファンクション	概要
19-10 ページ「getCode」	例外に埋め込まれた Oracle XML エラー・コードを取得します。
19-11 ページ「getFactoryCode」	例外に埋め込まれた FactoryException コードを取得します。
19-11 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
19-11 ページ「getMessage」	Oracle XML エラー・メッセージを取得します。

getCode

例外に埋め込まれた Oracle XML エラー・コードを取得します。XmlException から継承された仮想メンバー・ファンクション。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード（成功時は 0）。

getFactoryCode

これは実行中に発生した例外状況の、FactoryExceptionCode で定義されたツール名前空間に固有の例外コードを戻す、実装で定義されたメンバー・ファンクションのプロトタイプを定義する仮想メンバー・ファンクションです。

構文

```
virtual FactoryExceptionCode getFactoryCode() const = 0;
```

戻り値

(FactoryExceptionCode) 例外コード。

getMesLang

XmlException から継承された仮想メンバー・ファンクション。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext *) エラー・メッセージの現在の言語（エンコーディング）。

getMessage

XmlException から継承された仮想メンバー・ファンクション。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext*) エラー・メッセージ。

20

C++ 用の XPath API パッケージ

XPath パッケージには、XPath の処理に関連のある型およびインタフェースが含まれています。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XPath データ型

表 20-1 に、XPath パッケージのデータ型の概要を示します。

表 20-1 データ型の概要 : XPath パッケージ

データ型	説明
20-2 ページ「XPathCompIdType」	XPath コンパイラの識別子を定義します。
20-2 ページ「XPathObjType」	XPath 1.0 に基づいた実装のオブジェクト型を定義します。
20-3 ページ「XPathExceptionCode」	XPath に関連する例外コードです。
20-3 ページ「XPathPrIdType」	XPath プロセッサの識別子を定義します。

XPathCompIdType

XPath コンパイラの識別子を定義します。

定義

```
typedef enum XPathCompIdType {
   XvmXPathCompCXml = 1
} XPathCompIdType;
```

XPathObjType

XPath 1.0 に基づいた実装のオブジェクト型を定義します。

定義

```
typedef enum XPathObjType {
   XPOBJ_TYPE_UNKNOWN = 0,
   XPOBJ_TYPE_NDSET   = 1,
   XPOBJ_TYPE_BOOL    = 2,
   XPOBJ_TYPE_NUM     = 3,
   XPOBJ_TYPE_STR     = 4
} XPathObjType;
```

XPathExceptionCode

XPath に関連する例外コードです。

定義

```
typedef enum XPathExceptionCode {
   XPATH_UNDEFINED_ERR = 0,
   XPATH_OTHER_ERR = 1
} XPathExceptionCode;
```

XPathPrIdType

XPath プロセッサの識別子を定義します。

定義

```
typedef enum XPathPrIdType {
      XPathPrCXml       = 1,
      XvmPrCXml         = 2
    } XPathPrIdType;
```

CompProcessor インタフェース

表 20-2 に、CompProcessor インタフェースで使用できるメソッドの概要を示します。

表 20-2 CompProcessor メソッドの概要 : XPath パッケージ

ファンクション	概要
20-4 ページ「getProcessorId」	プロセッサの ID を取得します。
20-4 ページ「process」	指定したドキュメントに対して XPath 式を評価します。
20-5 ページ「processWithBinXPath」	指定したドキュメントに対してコンパイルした XPath 式を評価します。

getProcessorId

プロセッサの ID を取得します。

構文

```
virtual XPathPrIdType getProcessorId() const = 0;
```

戻り値

(XPathPrIdType) プロセッサの ID。

process

Processor から継承されます。

構文

```
virtual XPathObject< Node>* process (
   InputSource* isrc_ptr,
   oratext* xpath_exp)
throw (XPathException) = 0;
```

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。
xpath_exp	XPATH 式。

戻り値

(XPathGenObject*) XPath オブジェクト。

processWithBinXPath

指定したドキュメントに対してコンパイルした XPath 式を評価します。

構文

```
virtual XPathObject< Node>* processWithBinXPath (
   InputSource* isrc_ptr,
   ub2* bin_xpath)
throw (XPathException) = 0;
```

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。
bin_xpath	コンパイルした XPATH 式。

戻り値

(XPathGenObject*) XPath オブジェクト。

Compiler インタフェース

表 20-3 に、Compiler インタフェースで使用できるメソッドの概要を示します。

表 20-3 Compiler メソッドの概要 : XPath パッケージ

ファンクション	概要
20-6 ページ「compile」	XPath をコンパイルし、コンパイルしたバイナリ表現を戻します。
20-7 ページ「getCompilerId」	コンパイラの ID を取得します。

compile

XPath をコンパイルし、コンパイルしたバイナリ表現を戻します。

構文

```
virtual ub2* compile (
   oratext* xpath_exp)
throw (XPathException) = 0;
```

パラメータ	説明
xpath_exp	XPATH 式。

戻り値

(ub2) コンパイルしたバイナリ表現の XPath 式。

getCompilerId

コンパイラの ID を取得します。

構文

```
virtual XPathCompIdType getCompilerId() const = 0;
```

戻り値

(XPathCompIdType) コンパイラの ID。

NodeSet インタフェース

表 20-4 に、NodeSet インタフェースで使用できるメソッドの概要を示します。

表 20-4 NodeSet メソッドの概要 : XPath パッケージ

ファンクション	概要
20-8 ページ「getNode」	指定されたインデックスのノードを取得します。
20-8 ページ「getSize」	NodeSet のサイズを取得します。

getNode

ノードへの参照を戻します。

構文

```
NodeRef< Node>* getNode(
   ub4 idx) const;
```

パラメータ	説明
idx	セット内のノードのインデックス。

戻り値

(NodeRef) ノードへの参照。

getSize

ノード・セットのサイズ。

構文

```
ub4 getSize() const;
```

戻り値

(ub4) ノード・セットのサイズ。

Processor インタフェース

表 20-5 に、Processor インタフェースで使用できるメソッドの概要を示します。

表 20-5 Processor メソッドの概要 : XPath パッケージ

ファンクション	概要
20-9 ページ「getProcessorId」	プロセッサの ID を取得します。
20-9 ページ「process」	指定したドキュメントに対して XPath 式を評価します。

getProcessorId

プロセッサの ID を取得します。

構文

```
virtual XPathPrIdType getProcessorId() const = 0;
```

戻り値

(XPathPrIdType) プロセッサの ID。

process

指定したドキュメントに対して XPath 式を評価し、結果の XPath オブジェクトを戻します。

構文

```
virtual XPathObject< Node>* process (
   InputSource* isrc_ptr,
   oratext* xpath_exp)
throw (XPathException) = 0;
```

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。
xpath_exp	XPath 式。

戻り値

(XPathGenObject*) XPath オブジェクト。

XPathException インタフェース

表 20-6 に、XPathException インタフェースで使用できるメソッドの概要を示します。

表 20-6 XPathException メソッドの概要 : XPath パッケージ

ファンクション	概要
20-11 ページ「getCode」	例外に埋め込まれた Oracle XML のエラー・コードを取得します。
20-11 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
20-12 ページ「getMessage」	Oracle XML のエラー・メッセージを取得します。
20-12 ページ「getXPathCode」	例外に埋め込まれた XPath 例外のコードを取得します。

getCode

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード。正常に終了した場合は 0（ゼロ）が戻されます。

getMesLang

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext*) エラー・メッセージの現在の言語（エンコーディング）。

getMessage

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext *) エラー・メッセージ。

getXPathCode

プロトタイプを定義する仮想メンバー・ファンクションです。これは、実行中の例外発生時に、XPathExceptionCode で定義された XPath プロセッサおよびコンパイラの例外コードを戻す、実装によって定義されるメンバー・ファンクションのプロトタイプです。

構文

```
virtual XPathExceptionCode getXPathCode() const = 0;
```

戻り値

(XPathExceptionCode) 例外コード。

XPathObject インタフェース

表 20-7 に、XPathObject インタフェースで使用できるメソッドの概要を示します。

表 20-7 XPathObject メソッドの概要 : XPath パッケージ

ファンクション	概要
20-13 ページ「XPathObject」	コンストラクタをコピーします。
20-14 ページ「getNodeSet」	ノード・セットを取得します。
20-14 ページ「getObjBoolean」	オブジェクトからブール値を取得します。
20-14 ページ「getObjNumber」	オブジェクトから数値を取得します。
20-14 ページ「getObjString」	オブジェクトから文字列を取得します。
20-14 ページ「getObjType」	オブジェクトから型を取得します。

XPathObject

コンストラクタをコピーします。

構文

```
XPathObject(
   XPathObject< Node>& src);
```

パラメータ	説明
src	コピーするオブジェクトへの参照。

戻り値

(XPathObject) 新しいオブジェクト。

getNodeSet

ノード・セットを取得します。

構文

```
NodeSet< Node>* getNodeSet() const;
```

getObjBoolean

オブジェクトからブール値を取得します。

構文

```
boolean getObjBoolean() const;
```

getObjNumber

オブジェクトから数値を取得します。

構文

```
double getObjNumber() const;
```

getObjString

オブジェクトから文字列を取得します。

構文

```
oratext* getObjString() const;
```

getObjType

オブジェクトから型を取得します。

構文

```
XPathObjType getObjType() const;
```

21

C++ 用の XPointer API パッケージ

XPointer パッケージには、XPointer の処理に関連のある型およびインタフェースが含まれています。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

XPointer データ型

表 21-1 に、XPointer パッケージのデータ型の概要を示します。

表 21-1 データ型の概要 : XPointer パッケージ

データ型	説明
21-2 ページ「XppExceptionCode」	XPath コンパイラの識別子を定義します。
21-2 ページ「XppPrIdType」	XPointer プロセッサの識別子を定義します。
21-3 ページ「XppLocType」	XPointer の位置の型を定義します。

XppExceptionCode

XPointer に関連する例外コードです。

定義

```
typedef enum XPathCompIdType {
   XvmXPathCompCXml = 1
} XPathCompIdType;
```

XppPrIdType

XPointer プロセッサの識別子を定義します。

定義

```
typedef enum XppPrIdType {
   XPtrPrCXml          = 1
} XppPrIdType;
```

XppLocType

XPointer の位置の型を定義します。

定義

```
typedef enum XppLocType {
      XPPLOC_TYPE_UNKNOWN = 0,
      XPPLOC_TYPE_NODE    = 1,
      XPPLOC_TYPE_POINT   = 2,
      XPPLOC_TYPE_RANGE   = 3,
      XPPLOC_TYPE_BOOL    = 4,
      XPPLOC_TYPE_NUM     = 5,
      XPPLOC_TYPE_STR     = 6
    } XppLocType;
```

Processor インタフェース

表 21-2 に、Processor インタフェースで使用できるメソッドの概要を示します。

表 21-2 Processor メソッドの概要 : XPointer パッケージ

ファンクション	概要
21-4 ページ「getProcessorId」	プロセッサの ID を取得します。
21-4 ページ「process」	指定したドキュメントに対して XPointer 式を評価します。

getProcessorId

プロセッサの ID を取得します。

構文

```
virtual XppPrIdType getProcessorId() const = 0;
```

戻り値

(XppPrIdType) プロセッサの ID。

process

指定したドキュメントに対して XPointer 式を評価し、結果の XPointer の位置セット・オブジェクトを戻します。

構文

```
virtual XppLocSet< Node>* process (
   InputSource* isrc_ptr,
   oratext* xpp_exp)
throw (XppException) = 0;
```

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。
xpp_exp	XPointer 式。

戻り値

(XppLocSet*) XPath オブジェクト。

XppException インタフェース

表 21-3 に、XPPException インタフェースで使用できるメソッドの概要を示します。

表 21-3 XppException メソッドの概要 : XPointer パッケージ

ファンクション	概要
21-6 ページ「getCode」	例外に埋め込まれた Oracle XML のエラー・コードを取得します。
21-6 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
21-6 ページ「getMesLang」	Oracle XML のエラー・メッセージを取得します。
21-7 ページ「getXppCode」	例外に埋め込まれた XPointer の例外コードを取得します。

getCode

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード。正常に終了した場合は 0（ゼロ）が戻されます。

getMesLang

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext*) エラー・メッセージの現在の言語（エンコーディング）。

getMessage

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext *) エラー・メッセージ。

getXppCode

プロトタイプを定義する仮想メンバー・ファンクションです。これは、実行中の例外発生時に、XppExceptionCode で定義された XPointer プロセッサおよびコンパイラの例外コードを戻す、実装によって定義されるメンバー・ファンクションのプロトタイプです。

構文

```
virtual XppExceptionCode getXppCode() const = 0;
```

戻り値

(XppExceptionCode) 例外コード。

XppLocation インタフェース

表 21-4 に、XppLocation インタフェースで使用できるメソッドの概要を示します。

表 21-4 XppLocation メソッドの概要 : XPointer パッケージ

ファンクション	概要
21-8 ページ「getLocType」	位置の型を取得します。
21-8 ページ「getNode」	ノードを取得します。
21-8 ページ「getRange」	範囲を取得します。

getLocType

位置の型を取得します。

構文

```
XppLocType getLocType() const;
```

getNode

ノードを取得します。

構文

```
Node* getNode() const;
```

getRange

範囲を取得します。

構文

```
Range< Node>* getRange() const;
```

XppLocSet インタフェース

表 21-5 に、XppLocSet インタフェースで使用できるメソッドの概要を示します。

表 21-5 XppLocSet メソッドの概要 : XPointer パッケージ

ファンクション	概要
21-9 ページ「getItem」	指定されたインデックスの項目を取得します。
21-9 ページ「getSize」	位置セットのサイズを取得します。

getItem

項目への参照を戻します。

構文

```
XppLocation< Node>* getItem(
  ub4 index) const;
```

パラメータ	説明
index	項目のインデックス。

戻り値

(XppLocation*) 項目への参照。

getSize

ノード・セットのサイズ。

構文

```
ub4 getSize() const;
```

戻り値

(ub4) ノード・セットのサイズ。

22

C++ 用の Xsl API パッケージ

`Xsl` パッケージには、XSLT に関連のある型およびインタフェースが含まれています。

この章の内容は次のとおりです。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』

Xsl データ型

表 22-1 に、Xsl パッケージのデータ型の概要を示します。

表 22-1 データ型の概要 : Xsl パッケージ

データ型	説明
22-2 ページ「XslCompIdType」	XSL コンパイラの ID を定義します。
22-2 ページ「XslExceptionCode」	XSLT に関連のある例外を定義します。
22-3 ページ「XslTrIdType」	XSL トランスフォーマの ID を定義します。

XslCompIdType

XSL コンパイラの ID を定義します。

定義

```
typedef enum XslCompIdType {
   XvmCompCXml = 1
} XslCompIdType;
```

XslExceptionCode

XSLT に関連のある例外を定義します。

定義

```
typedef typedef enum XslExceptionCode {
   XSL_UNDEFINED_ERR = 0,
   XSL_OTHER_ERR = 1
} XslExceptionCode;
```

XslTrIdType

XSL トランスフォーマの ID を定義します。

定義

```
typedef enum XslTrIdType {
   XslTrCXml          = 1,
   XvmTrCXml          = 2
} XslTrIdType;
```

Compiler インタフェース

表 22-2 に、Compiler インタフェースで使用できるメソッドの概要を示します。

表 22-2 Compiler メソッドの概要 : Xsl パッケージ

ファンクション	概要
22-4 ページ「compile」	Xsl をコンパイルし、コンパイルしたバイナリ表現を戻します。
22-5 ページ「getCompilerId」	コンパイラの ID を取得します。
22-5 ページ「getLength」	コンパイルした XSL 文書の長さを取得します。

compile

Xsl をコンパイルし、コンパイルしたバイナリ表現を戻します。

構文

```
virtual ub2* compile(
   InputSource* isrc_ptr)
throw (XslException) = 0;
```

パラメータ	説明
isrc_ptr	Xsl 文書。

戻り値

(InputSource) コンパイルしたバイナリ表現の Xsl 文書。

getCompilerId

コンパイラの ID を取得します。

構文

```
virtual XslCompIdType getCompilerId() const = 0;
```

戻り値

(XslCompIdType) コンパイラの ID。

getLength

コンパイルした XSL 文書の長さを戻します。

構文

```
virtual ub4 getLength(
   ub2* binxsl_ptr)
throw (XslException) = 0;
```

パラメータ	説明
binxsl_ptr	コンパイルした Xsl 文書。

戻り値

(ub4) 文書の長さ。

CompTransformer インタフェース

表 22-3 に、CompTransformer インタフェースで使用できるメソッドの概要を示します。

表 22-3 CompTransformer メソッドの概要 : Xsl パッケージ

ファンクション	概要
22-6 ページ「getTransformerId」	トランスフォーマの ID を取得します。
22-7 ページ「setBinXsl」	コンパイルした Xsl を設定します。
22-7 ページ「setSAXHandler」	SAX ハンドラを設定します。
22-8 ページ「setXSL」	トランスフォーマに XSLT 文書を設定します。
22-8 ページ「transform」	ドキュメントを変換します。

getTransformerId

トランスフォーマの ID を取得します。

構文

```
virtual XslTrIdType getTransformerId() const = 0;
```

戻り値

(XslTrIdType) トランスフォーマの ID。

setBinXsl

コンパイルした Xsl を設定します。

構文

```
virtual void setBinXsl (
   ub2* binxsl_ptr)
throw (XslException) = 0;
```

パラメータ	説明
binxsl_ptr	コンパイルした Xsl 文書。

setSAXHandler

トランスフォーマから継承されます。

構文

```
virtual void setSAXHandler(
   SAXHandlerRoot* hdlr_ptr) = 0;
```

パラメータ	説明
hdlr_ptr	SAX ハンドラのポインタ。

setXSL

トランスフォーマに XSLT 文書を設定します。transform メンバー・ファンクションがコールされる前にコールする必要があります。これは、Transform から継承されます。

構文

```
virtual void setXSL (
   InputSource* isrc_ptr)
throw (XslException) = 0;
```

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。

transform

ドキュメントを変換します。事前の setXSL コールによって XSLT 文書が設定されていない場合、例外が発生します。これは、Transform から継承されます。

構文	説明

`virtual NodeRef< Node>* transform(`
`  nputSource* isrc_ptr)`
`throw (XslException) = 0;`
	ドキュメントを変換し、DOM を戻します。

`virtual void transform(`
`  InputSource* isrc_ptr,`
`  SAXHandlerRoot* hdlr_ptr)`
`throw (XslException) = 0;`
	ドキュメントを変換し、SAX イベントを戻します。

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。
hdlr_ptr	SAX ハンドラのポインタ。

戻り値

(DocumentRef) 新しいドキュメントのドキュメント・ツリー。

Transformer インタフェース

表 22-4 に、Transformer インタフェースで使用できるメソッドの概要を示します。

表 22-4 Transformer メソッドの概要 : Xsl パッケージ

ファンクション	概要
22-9 ページ「getTransformerId」	トランスフォーマの ID を取得します。
22-9 ページ「setSAXHandler」	SAX ハンドラを設定します。
22-10 ページ「setXSL」	トランスフォーマに XSLT 文書を設定します。
22-10 ページ「transform」	ドキュメントを変換し、SAX イベントを戻します。

getTransformerId

トランスフォーマの ID を取得します。

構文

```
virtual XslTrIdType getTransformerId() const = 0;
```

戻り値

(XslTrIdType) トランスフォーマの ID。

setSAXHandler

SAX ハンドラを設定します。

構文

```
virtual void setSAXHandler(
   SAXHandlerRoot* hdlr_ptr) = 0;
```

パラメータ	説明
hdlr_ptr	SAX ハンドラのポインタ。

setXSL

トランスフォーマに XSLT 文書を設定します。transform メンバー・ファンクションがコールされる前にコールする必要があります。

構文

```
virtual void setXSL (
   InputSource* isrc_ptr)
throw (XslException) = 0;
```

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。

transform

ドキュメントを変換します。事前の setXSL コールによって XSLT 文書が設定されていない場合、例外が発生します。

構文	説明

`virtual NodeRef< Node>* transform(`
`  nputSource* isrc_ptr)`
`throw (XslException) = 0;`
	ドキュメントを変換し、DOM を戻します。

`virtual void transform(`
`  InputSource* isrc_ptr,`
`  SAXHandlerRoot* hdlr_ptr)`
`throw (XslException) = 0;`
	ドキュメントを変換し、SAX イベントを戻します。

パラメータ	説明
isrc_ptr	処理するインスタンス・ドキュメント。
hdlr_ptr	SAX ハンドラのポインタ。

戻り値

(DocumentRef) 新しいドキュメントのドキュメント・ツリー。

XSLException インタフェース

表 22-5 に、XSLException インタフェースで使用できるメソッドの概要を示します。

表 22-5 XSLException メソッドの概要 : Xsl パッケージ

ファンクション	概要
22-11 ページ「getCode」	例外に埋め込まれた Oracle XML のエラー・コードを取得します。
22-11 ページ「getMesLang」	エラー・メッセージの現在の言語（エンコーディング）を取得します。
22-12 ページ「getMessage」	Oracle XML のエラー・メッセージを取得します。
22-12 ページ「getXslCode」	実装するプロトタイプを定義します。

getCode

例外に埋め込まれた Oracle XML のエラー・コードを取得します。XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual unsigned getCode() const = 0;
```

戻り値

(unsigned) 数値のエラー・コード。正常に終了した場合は 0（ゼロ）が戻されます。

getMesLang

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMesLang() const = 0;
```

戻り値

(oratext*) エラー・メッセージの現在の言語（エンコーディング）。

getMessage

XmlException から継承される仮想メンバー・ファンクションです。

構文

```
virtual oratext* getMessage() const = 0;
```

戻り値

(oratext *) エラー・メッセージ。

getXslCode

プロトタイプを定義する仮想メンバー・ファンクションです。これは、実行中の例外発生時に、XslExceptionCode で定義された XSL トランスフォーマおよびコンパイラの例外コードを戻す、実装によって定義されるメンバー・ファンクションのプロトタイプです。

構文

```
virtual XslExceptionCode getXslCode() const = 0;
```

戻り値

(XslExceptionCode) 例外コード。