Novell Developer Kit

www.novell.com

XML INTERFACES FOR C++

March 1, 2006

Novell®

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

AppNotes is a registered trademark of Novell, Inc.
AppTester is a registered trademark of Novell, Inc., in the United States.
ASM is a trademark of Novell, Inc.
Beagle is a trademark of Novell, Inc.
BorderManager is a registered trademark of Novell, Inc.
BrainShare is a registered service mark of Novell, Inc., in the United States and other countries.
C3PO is a trademark of Novell, Inc.
Certified Novell Engineer is a service mark of Novell, Inc.
Client32 is a trademark of Novell, Inc.
CNE is a registered service mark of Novell, Inc.
ConsoleOne is a registered trademark of Novell, Inc.
Controlled Access Printer is a trademark of Novell, Inc.
Custom 3rd-Party Object is a trademark of Novell, Inc.
DeveloperNet is a registered trademark of Novell, Inc., in the United States and other countries.
DirXML is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
Excelerator is a trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
exteNd Workbench is a trademark of Novell, Inc.
FAN-OUT FAILOVER is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc., in the United States and other countries.
Hardware Specific Module is a trademark of Novell, Inc.
Hot Fix is a trademark of Novell, Inc.
Hula is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
Internetwork Packet Exchange is a trademark of Novell, Inc.
IPX is a trademark of Novell, Inc.
IPX/SPX is a trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
Link Support Layer is a trademark of Novell, Inc.
LSL is a trademark of Novell, Inc.
ManageWise is a registered trademark of Novell, Inc., in the United States and other countries.
Mirrored Server Link is a trademark of Novell, Inc.
Mono is a registered trademark of Novell, Inc.
MSL is a trademark of Novell, Inc.
My World is a registered trademark of Novell, Inc., in the United States.
NCP is a trademark of Novell, Inc.
NDPS is a registered trademark of Novell, Inc.
NDS is a registered trademark of Novell, Inc., in the United States and other countries.
NDS Manager is a trademark of Novell, Inc.
NE2000 is a trademark of Novell, Inc.
NetMail is a registered trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc., in the United States and other countries.
NetWare/IP is a trademark of Novell, Inc.

Third-Party Materials

Contents

About This Guide

This guide describes the interfaces that are included in the DirXML developer's kit for manipulating XML documents in C++ (using SAX, DOM, and serialized methods). It contains functions and methods for performing the following manipulations on the documents:

- Converting data between UTF-8 and UTF-16 encoding
- Filtering data
- Encoding with Base64
- Creating new instances of the XML objects
- Writing to output streams
- Writing messages to DSTrace and a DirXML log file

This guide contains the following sections:

Feedback

We want to hear your comments and suggestions about this manual. Please use the User Comments feature at the bottom of each page of the online documentation.

Documentation Updates

For the most recent version of Novell Identity Manager (DirXML) Driver Kit, visit the Novell Identity manager (DirXML) Driver Kit NDK page (http://developer.novell.com/ndk/dirxml.htm).

Additional Information

For the related developer support postings for DirXML, see the Developer Support Forums (http://developer.novell.com/ndk/devforums.htm).

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

XmlDocument Interface

1

The XmlDocument interface provides access to the XML document through three views: a DOM tree view, a serialized stream view, and a SAX event view. The DirXML engine uses this interface to pass XML documents to and retrieve documents from the driver. Your driver should use this interface to pass XML documents to and retrieve documents from the DirXML engine.

This interface is different from the other interfaces in the NativeInterface.h file:

- You do not need to implement the methods in the XmlDocument interface. An implementation is provided.
- You do not need to use all of them. You need to select the view you are going to use to manipulate the XML documents, and use only those methods.

The following sections describe what is available for each view in the DirXML developer kit:

- Section 1.1, “DOM,” on page 15
- Section 1.2, “SAX,” on page 15
- Section 1.3, “Serialized,” on page 16
- Section 1.4, “C++ Utility Functions,” on page 16
- Section 1.5, “Java Utility Classes,” on page 17

1.1 DOM

For the DOM (Document Object Model) view, the XmlDocument interface has the following method:

- getDocument (page 18)

For C++, after retrieving the document, there is no standard interface for manipulating documents in DOM. The dom.h file mirrors the Java interface and includes documentation about these methods. For more information, see Chapter 2, “Document Object Model (DOM) Interfaces,” on page 25XML Interfaces for C++.

For Java, see the com.novell.xml.dom package (../../dirxmlbk/api/com/novell/xml/dom/package-summary.html).

1.2 SAX

For the SAX (Simple API for XML) event view, the XmlDocument interface has the following methods:

- getDocumentSAX (page 21)
- getDocumentInputSource (page 20)

For C++, after retrieving the document, there is no standard interface for manipulating documents in SAX. The sax.h file mirrors the Java interface and includes documentation about these methods. For more information, see Chapter 3, “Simple API for XML (SAX) Interfaces,” on page 111.

For Java, see the com.novell.xml.sax package (../../dirxmlbk/api/com/novell/xml/sax/package-summary.html).

1.3 Serialized

For the serialized stream view, the XmlDocument interface has the following methods:

- getDocumentBytes (page 19)
- getXMLWriter (page 22)
- releaseXmlWriter (page 23)
- writeDocument (page 24)

The serialization is written to an OutputStream interface.

- In C++, a FileOutputStream implementation and a ByteArrayOutputStream implementation are available. The serialization can be written in various character encodings: UTF-8, UTF-16, US-ASCII, and ISO-8859-1.
- In Java, the serialization can be written in various character encodings such as UTF-8, UTF-16, US-ASCII, and ISO-8859-1 through 9, 14 and 15. Any other encoding supported in the Java environment is available, but unencodable characters will not be correctly escaped in the serialized XML.

For C++, see Chapter 4, “Serialized XML Interface,” on page 169.

For Java, see com.novell.nds.dirxml.driver.XmlDocument (../../dirxmlbk/api/com/novell/nds/dirxml/driver/XmlDocument.html).

1.4 C++ Utility Functions

For C++, the DirXML developer kit also provides access to the following.

Header File	**Description**
OutputStream.h	Defines an interface modeled on java.io.OutputStream for writing to a byte sink.
UTFConverter.h	Converts between UTF-8 and UTF-16 encoding.
XMLWriter.h	Creates an XMLWriter.
Base64Code.h	Defines an interface for encoding binary data. Attributes which contain binary data (octet syntax type) must be encoded in an XML document.
DriverFilter.h	Defines an interface for using the driver filter to filter classes and attributes.
InterfaceFactory.h	Defines an interface for constructing XML document objects in all views: DOM, SAX, and serialized.
NdsDtd.h	Contains helper functions for creating input and output documents.
Trace.h	Enables a DirXML driver to write debug messages to the DSTrace console and to the DirXML log file.

For more information, see XML Interfaces for C++.

1.5 Java Utility Classes

The DirXML developer kit also provides access to the following Java utility classes.

Class	Description
ClassFilter	Allows use of the publisher or subscriber filter passed to the init method.
DelimitedText	Provides methods for representing a delimited text file as XML.
DriverFilter	Allows use of the publisher or subscriber filter passed to the init method.
ThreadBridge	Provides a method of calling methods on a different thread.
Trace	Enables a DirXML driver to write debug messages to the DSTrace console and to the DirXML log file.
XdsQueryProcessor	Provides a query processor to use within XSLT style sheets which are rules or by functions called from an XSLTstyle sheet which are used as rules.

For more information, see the com.novell.nds.dirxml.driver package (../../dirxmlbk/api/com/novell/nds/dirxml/driver/package-summary.html).

getDocument

Returns the XML document as a DOM tree.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

Document * METHOD_CALL getDocument (
   void);
```

Java

```
public Document getDocument ()
```

Remarks

The returned document belongs to the XmlDocument object. You should not try to delete it.

getDocumentBytes

Returns the XML document as a serialized byte array.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

const unsigned char * METHOD_CALL getDocumentBytes (
   const unicode   *encoding,
   int              endian,
   int             *length);
```

Java

```
public byte[] getDocumentBytes (
   java.lang.String   encoding)
```

Parameters

encoding

Points a standard encoding string such as "ASCII", "US-ASCII", "UTF-8", or "UTF-16".

endian

Specifies the byte-order: 0=little-endian, 1=big endian, if necessary.

length

Points to a variable which receives the length of the array, specified in bytes.

Remarks

This method returns zero if there is no data or the desired encoding is not supported. "US-ASCII", UTF-8", and "UTF-16" are always supported.

getDocumentInputSource

Returns a SAX input source object to use with the parser returned from the getDocmentSAX method.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

InputSource * METHOD_CALL getDocumentInputSource (
   void);
```

Java

```
public InputSource getDocumentInputSource ()
```

Remarks

The returned InputSource must only be used with the parser returned from the getDocumentSAX method.

The InputSource belongs to the XmlDocument object. You should not try to delete it.

getDocumentSAX

Returns a SAX parser interface through which the caller can get a series of events describing the document.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

Parser * METHOD_CALL getDocumentSAX (
   void);
```

Java

```
public Parser getDocumentSAX ()
```

Remarks

The caller must also call the getDocumentInputSource method to get the InputSource to use with the parser's parse method.

getXMLWriter

Returns an XmlWriter interface that can be used to serialize an XML document.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

XmlWriter * METHOD_CALL getXmlWriter (
   OutputStream   *outputStream);
```

Java

```
public XmlWriter getXmlWriter (
     java.io.OutputStream   outputStream,
     java.lang.String        encoding)


throws   java.io.UnsupportedEncodingException
```

Parameters

outputStream

Points to the stream to which to write. This parameter cannot be zero (0).

encoding

Specifies the character encoding to use in writing to the outputStream such as "ASCII", "US-ASCII", "UTF-8", or "UTF-16".

Remarks

The XmlWriter interface has methods for closely controlling how the XML is output. For example, if the document contains only text, the XML output escaping can be disabled.

After you set the various attributes, calling the XmlWriter's write method causes the serialization to occur.

releaseXmlWriter

Notifies the XmlDocument object that it can free the XmlWriter object because the XmlWriter object returned from getXMLWriter.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

void METHOD_CALL releaseXmlWriter (
   void);
```

Remarks

This method must be called before the OutputStream that is passed to the getXmlWriter method is released. This method is not needed in Java.

writeDocument

Serializes the document to the specified output stream using the default settings.

NDS Version: 8.5

Syntax

C++

```
#include "NativeInterface.h"

int METHOD_CALL writeDocument (
   OutputStream         *outputStream,
   const unicode        *encoding,
   int                   endian);
```

Java

```
public void writeDocument (
   java.io.OutputStream   document,
   java.lang.String       encoding)


throws   java.io.IOException,
         java.io.UnsupportedEncodingException
```

Parameters

outputStream

Specifies the stream to which to write. This parameter cannot be zero (0).

encoding

Specifies the character encoding to use. If encoding is set to zero (0), the default encoding, UTF-8, is used.

endian

Specifies the byte-order: 0=little-endian, 1=big endian.

Remarks

A non-zero return value indicates success.

This is an overloaded method in Java. The writeDocument method can also write the contents to an XML writer.

Document Object Model (DOM) Interfaces

2

The following C++ interfaces and methods are patterned after the Java implementation of the W3C DOM (Document Object Model) Level 1. For complete documentation of this interface, see the web site of the W3C Organization (http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html).

- Section 2.1, “Node,” on page 27
- Section 2.2, “Document,” on page 48
- Section 2.3, “Element,” on page 61
- Section 2.4, “Attr,” on page 71
- Section 2.5, “CharacterData,” on page 75
- Section 2.6, “Text,” on page 83
- Section 2.7, “ProcessingInstruction,” on page 85
- Section 2.8, “DocumentType,” on page 88
- Section 2.9, “DOMImplementation,” on page 91
- Section 2.10, “Entity,” on page 92
- Section 2.11, “Notation,” on page 96
- Section 2.12, “NamedNodeMap,” on page 98
- Section 2.13, “NodeList,” on page 103
- Section 2.14, “DOMException,” on page 106

Because this interface supports C as well as C++, both return values and exceptions are supported. To enable exceptions for C++, you must call the enableExceptions method in the Document interface with a non-zero value.

All methods that are defined in the Java interfaces with a void return are defined in this interface to return a pointer to a DOMException interface.

- If the return pointer is null, there was no error.
- If the return pointer is not null, the DOMException interface can be queried for the error.

For all other methods, if the return pointer is null, there was an error. Call the getLastError method of the Document interface to receive a DOMException interface that you can query for the error.

The DOM methods are grouped by the type of node the method interacts with. The Node interface has methods for general manipulation of nodes. Some of the methods return structures that can be manipulated with the following interfaces.

Method	Returned Structure	Interface
getChildNodes	NodeList	Section 2.13, “NodeList,” on page 103

Method	Returned Structure	Interface
appendChild getLastError insertBefore removeChild replaceChild setNodeValue	DOMException	Section 2.14, "DOMException," on page 106
getAttributes	NamedNodeMap	Section 2.12, "NamedNodeMap," on page 98

Some node types have specialized interfaces. The table below lists these nodes and their interfaces:

Node Type	Interface
Attribute	Section 2.4, "Attr," on page 71
Document	Section 2.2, "Document," on page 48
Document Type	Section 2.8, "DocumentType," on page 88
Element	Section 2.3, "Element," on page 61
Entity	Section 2.10, "Entity," on page 92
Notation	Section 2.11, "Notation," on page 96
Processing Instruction	Section 2.7, "ProcessingInstruction," on page 85
Text	Section 2.6, "Text," on page 83

Some node types do not have a specialized interface. The following node types must be manipulated with the Node interface or the inherited interface from the parent node interface.

CDATASection Interface. The CDATASection interface inherits from the Text interface. CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections can not be nested. The primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The DOMString attribute of the Text node holds the text that is contained by the CDATA section. The text may contain characters that need to be escaped outside of CDATA sections and that, depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The CDATASection interface inherits the CharacterData interface through the Text interface. Adjacent CDATASection nodes are not merged by use of the Element.normalize() method.

Comment Interface. The Comment interface inherits from the CharacterData interface and represents the contents of a comment. In other words, a comment node represents all the characters between the starting '<!--' and ending '-->'. This is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

EntityReference Interface. The EntityReference interface inherits from the Node interface. Entity reference objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference. Character references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. Moreover, the XML processor may completely expand references to entities while building the structure model, instead of providing EntityReference objects. If it does provide such objects, then for a given EntityReference node, it may be that there is no Entity node representing the referenced entity; but if such an Entity exists, then the child list of the EntityReference node is the same as that of the Entity node. As with the Entity node, all descendants of the EntityReference are read-only.

The resolution of the children of the EntityReference (the replacement value of the referenced Entity) may be lazily evaluated; actions by the user (such as calling the childNodes method on the EntityReference node) are assumed to trigger the evaluation.

Document Fragment. The DocumentFragment interface inherits from the Node interface and is a “lightweight” or “minimal” Document object and allows you to extract a portion of a document’s tree or to create a new fragment of a document.

Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments, and it is quite natural to use a Node for this purpose. While it is true that a Document object could fulfil this role, a Document object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. DocumentFragment is such an object and is created with the createDocumentFragment method of the Document interface.

Various operations—such as inserting nodes as children of another Node—may take DocumentFragment objects as arguments; this results in all the child nodes of the DocumentFragment being moved to the child list of this node.

The children of a DocumentFragment node are zero or more nodes, representing the tops of any sub-trees defining the structure of the document. DocumentFragment nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a DocumentFragment might have only one child and that child node could be a Text node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a DocumentFragment is inserted into a Document (or indeed any other Node that may take children), the children of the DocumentFragment and not the DocumentFragment itself are inserted into the Node. This makes the DocumentFragment very useful when the user wishes to create nodes that are siblings; the DocumentFragment acts as the parent of these nodes so that the user can use the standard methods from the Node interface, such as the insertBefore method and the appendChild method.

2.1 Node

The Node interface is the primary data type for the entire Document Object Model.It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a DOMException being raised.

The attributes (nodeName, nodeValue, and attributes) are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (for example, nodeValue for an Element or attributes for a Comment), these attributes return NULL. The specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

NOTE: This C++ binding adds the following methods to the DOM Level 1 specified methods:

```
void enableExceptions (int enable);
DOMException * getLastError ();
void destroy();
```

---

appendChild

Adds the newChild node to the end of the list of children of this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, appendChild, Node *)
   METHOD_PARM1(Node * newChild)
END_METHOD
```

C++

```
Node * METHOD_CALL appendChild (
   Node   *newChild);
```

Parameters

newChild

(IN) Points to the new child node to add.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

Returns WRONG_DOCUMENT_ERR if this Node is of a type that does not allow children of the newChild type, or if newChild is one of this Node's ancestors.

Returns HIERARCHY_REQUEST_ERR if newChild is of a type that is not allowed as a child of this Node.

Remarks

If the newChild node is already in the tree, it is first removed.

cloneNode

Returns a duplicate of this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, cloneNode, Node *)
   METHOD_PARM1(int deep)
END_METHOD
```

C++

```
Node * METHOD_CALL cloneNode (
   int    deep);
```

Parameters

deep

(IN) Specifies whether only the node or all subordinate nodes are cloned. If non-zero, specifies to recursively clone the subtree under the node.

Remarks

This method serves as a generic copy constructor for nodes. The duplicate node has no parent. For example, the getParentNode method returns NULL.

Cloning an Element copies all attributes and their values, including those generated by an XML processor to represent defaulted attributes. This method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

destroy

Deletes this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, destroy, void)
END_METHOD
```

C++

```
void METHOD_CALL destroy ();
```

Remarks

If this node is currently in the tree, then this method is equivalent to calling the removeChild method on the node’s parent and then the destroy method on this node. If this node is a Document object, the entire tree is destroyed.

This method is a C++ extension to DOM Level 1.

getAttributes

Returns a NamedNodeMap with a list of the attributes of the node.

Syntax

Defining Macros for C++

```
#include "dom.h"
```

```
METHOD (Node, getAttributes, NamedNodeMap *)
END_METHOD
```

C++

```
NamedNodeMap * METHOD_CALL getAttributes ();
```

Return Values

Returns a NamedNodeMap if the node is an element; otherwise returns NULL.

Remarks

The returned NamedNodeMap belongs to the node. No attempt must be made to delete it.

Only elements have attributes. For example, if the node is a document, a text or a CDATA node, this method returns NULL.

For the methods available to manipulate the NamedNodeMap, see Section 2.12, “NamedNodeMap,” on page 98.

getChildNodes

Returns a NodeList with the children of the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getChildNodes, NodeList *)
END_METHOD
```

C++

```
NodeList * METHOD_CALL getChildNodes ();
```

Return Values

Returns a NodeList. If the node has no children, it returns an empty NodeList.

Remarks

The returned NodeList must be deleted when it is no longer needed. This must be done by calling the NodeList::destroy method.

If this Node is deleted before the NodeList is destroyed, the NodeList becomes invalid, and results are unpredictable.

For the methods available to manipulate the NodeList, see Section 2.13, “NodeList,” on page 103.

getFirstChild

Returns the first child of the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getFirstChild, Node *)
END_METHOD
```

C++

```
Node * METHOD_CALL getFirstChild ();
```

Return Values

Returns the first child of the node or NULL if the node has no children.

getLastChild

Returns the last child of this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getLastChild, Node *)
END_METHOD
```

C++

```
Node * METHOD_CALL getLastChild ();
```

Return Values

Returns the last child of the node or NULL if the node has no children.

getLastError

Returns the DOMException generated by the last operation on this node. If no exception was generated, returns NULL.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getLastError, DOMException *)
END_METHOD
```

C++

```
DOMException * METHOD_CALL getLastError ();
```

Remarks

The DOMException returned by this method belongs to the Node. No attempt should be made to delete it.

This method is a C++ extension to DOM Level 1.

getNextSibling

Returns the node immediately following this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getNextSibling, Node *)
END_METHOD
```

C++

```
Node * METHOD_CALL getNextSibling ();
```

Return Values

Returns the node following this node or NULL if no following node exists.

getNodeName

Returns the name of this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getNodeName, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getNodeName ();
```

Return Values

Returns the Unicode name of the node or 0 (zero) if the node does not have a name.

Remarks

Not all node types have a name.

getNodeType

Returns the type of this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getNodeType, short)
END_METHOD
```

C++

```
short METHOD_CALL getNodeType ();
```

Return Values

Returns a value from 1 through 12 that specifies the type of node.

Remarks

NodeType defines the types of nodes.

```
enum NodeType
{
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
};
```

getNodeValue

Returns the value of this node

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getNodeValue, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getNodeValue ();
```

Return Values

Returns the Unicode value of the node or 0 (zero) if the node has no value.

Remarks

Not all node types have values.

getOwnerDocument

Returns the Document object associated with the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getOwnerDocument, Document *)
END_METHOD
```

C++

```
Document * METHOD_CALL getOwnerDocument ();
```

Return Values

Returns the Document object to which the node belongs or NULL if the node is the Document object.

Remarks

The Document object is used to create new nodes.

getParentNode

Returns the parent of the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getParentNode, Node *)
END_METHOD
```

C++

```
Node * METHOD_CALL getParentNode ();
```

Return Values

Returns the parent node, or NULL if the node does not have a parent.

Remarks

All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has been created but not yet added to the tree, this method returns NULL for the parent.

getPreviousSibling

Returns the node immediately preceding this node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, getPreviousSibling, Node *)
END_METHOD
```

C++

```
Node * METHOD_CALL getPreviousSibling ();
```

Return Values

Returns the node preceding this node or NULL if no previous node exists.

hasChildNodes

Returns non-zero if this node has children.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, hasChildNodes, int)
END_METHOD
```

C++

```
int METHOD_CALL hasChildNodes ();
```

Remarks

This is a convenience method to allow easy determination of whether a node has any children.

insertBefore

Inserts the newChild node before the specified node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, insertBefore, Node *)
   METHOD_PARM1(Node * newChild)
   METHOD_PARM (Node * refChild)
END_METHOD
```

C++

```
Node * METHOD_CALL insertBefore (
   Node   *newChild,
   Node   *refChild);
```

Parameters

newChild

(IN) Points to the new child node to add to the document.

refChild

(IN) Points to the child node before which the new child node is inserted. If this parameter is NULL, the new node is inserted as the last child.

Return Values

DOMException registered or thrown.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read only.

Returns WRONG_DOCUMENT_ERR if this Node is of a type that does not allow children of the newChild type, or if newChild is one of this Node's ancestors.

Returns HIERARCHY_REQUEST_ERR if newChild is of a type that is not allowed as a child of this Node.

Returns NOT_FOUND_ERR if refChild is not a child of this Node.

Remarks

If the newChild node is a DocumentFragment object, all of its children are inserted, in the same order, before the refChild node. If the newChild node is already in the tree, it is first removed.

removeChild

Removes the child indicated by oldChild from the list of children and returns it.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, removeChild, Node *)
   METHOD_PARM1(Node * oldChild)
END_METHOD
```

C++

```
Node * METHOD_CALL removeChild (
   Node   *oldChild);
```

Parameters

oldChild

(IN) Points to the child node to remove.

Return Values

If successful, returns the old child node. If unsuccessful, registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

Returns NOT_FOUND_ERR if oldChild is not a child of the Node.

Remarks

The destroy() method must be called on the returned node.

replaceChild

Replaces the oldChild node with the newChild node in the list of children and returns the oldChild node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, replaceChild, Node *)
   METHOD_PARM1(Node * newChild)
   METHOD_PARM (Node * oldChild)
END_METHOD
```

C++

```
Node * METHOD_CALL replaceChild (
   Node   *newChild,
   Node   *oldChild);
```

Parameters

newChild

(IN) Points to the node that will replace an existing node.

oldChild

(IN) Points to the node that is to be replaced.

Return Values

If successful, returns the oldChild node. If unsuccessful, registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

Returns WRONG_DOCUMENT_ERR if this Node is of a type that does not allow children of the newChild type, or if newChild is one of this Node’s ancestors

Returns HIERARCHY_REQUEST_ERR if newChild is of a type that is not allowed as a child of this Node.

Returns NOT_FOUND_ERR if refChild is not a child of this Node.

Remarks

If the newChild node is already in the tree, it is first removed.

The destroy() method must be called on the returned node.

setNodeValue

Sets the value of the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, setNodeValue, DOMException *)
METHOD_PARM1(const unicode * value)
END_METHOD
```

C++

```
DOMException * METHOD_CALL setNodeValue (
   const unicode   *value);
```

Parameters

value

Points to the value for the node.

Return Values

Returns or throws a DOMException.

If the Node is read-only, returns NO_MODIFICATION_ALLOWED_ERR.

Remarks

This method returns an error if the node is read-only.

2.2 Document

The Document interface inherits from the Node interface and represents the entire HTML or XML document. Conceptually, it is the root of the document tree and provides the primary access to the document's data.

Since the other nodes such as elements, text nodes, comments, and processing instructions cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

createAttribute

Creates an Attribute node of the given name.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createAttribute, Attr *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
Attr * METHOD_CALL createAttribute (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the attribute.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns INVALID_CHARACTER_ERROR if the attribute name is invalid.

Remarks

The Attr instance can be set on an Element using the setAttribute method.

createCDATASection

Creates a CDATASection node whose value is the specified string.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createCDATASection, CDATASection *)
   METHOD_PARM1(const unicode * data)
   METHOD_PARM (int offset)
   METHOD_PARM (int length)
END_METHOD
```

C++

```
CDATASection * METHOD_CALL createCDATASection (
   const unicode   *data,
   int              offset,
   int              length);
```

Parameters

data

(IN) Points to the string to place in the CDATASection node.

offset

?

length

(IN) Specifies, in bytes, the length of the string.

createComment

Creates a Comment node from the specified string.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createComment, Comment *)
   METHOD_PARM1(const unicode * data)
END_METHOD
```

C++

```
Comment * METHOD_CALL createComment (
   const unicode   *data);
```

Parameters

data

(IN) Points the string to place in the Comment node.

Remarks

Does the string need to begin and end with the XML markings for a comment or does this method add the markings?

createDocumentFragment

Creates an empty DocumentFragment object.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createDocumentFragment, DocumentFragment *)
END_METHOD
```

C++

```
DocumentFragment * METHOD_CALL createDocumentFragment ();
```

Remarks

Use any of the other create... methods of the Document interface to add nodes to the DocumentFragment.

createElement

Creates an element of the type specified.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createElement, Element *)
   METHOD_PARM1(const unicode * tagName)
END_METHOD
```

C++

```
Element * METHOD_CALL createElement (
   const unicode   *tagName);
```

Parameters

tagName

(IN) Points to the name of the element to create.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns INVALID_CHARACTER_ERROR if tagName is invalid.

Remarks

The instance returned implements the Element interface, so attributes can be specified directly on the returned object.

createEntityReference

Creates an EntityReference object.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createEntityReference, EntityReference *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
EntityReference * METHOD_CALL createEntityReference (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the Entity Reference.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns INVALID_CHARACTER_ERROR if the name is invalid.

createProcessingInstruction

Creates a ProcessingInstruction node from the specified target and data strings.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createProcessingInstruction, ProcessingInstruction
*)
   METHOD_PARM1(const unicode * target)
   METHOD_PARM (const unicode * data)
END_METHOD
```

C++

```
ProcessingInstruction * METHOD_CALL createProcessingInstruction (
   const unicode   *target,
   const unicode   *data);
```

Parameters

target

(IN) Points to the processing instruction target.

data

(IN) Points to the data string.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns INVALID_CHARACTER_ERROR if the target or data is invalid.

createTextNode

Creates a Text node from the specified string.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, createTextNode, Text *)
   METHOD_PARM1(const unicode * data)
   METHOD_PARM (int offset)
   METHOD_PARM (int length)
END_METHOD
```

C++

```
Text * METHOD_CALL createTextNode (
   const unicode   *data,
   int              offset,
   int              length);
```

Parameters

data

(IN) Points to the text string to place in the Text node.

offset

?

length

(IN) Specifies, in bytes, the length of the string?

enableExceptions

Enables the throwing of C++ exceptions for all nodes created by this Document and for all nodes in this Document object's tree.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, enableExceptions, int)
   METHOD_PARM1(int enable)
END_METHOD
```

C++

```
int METHOD_CALL enableExceptions (
   int   enable);
```

Parameters

enable

(IN) Specifies whether to enable the throwing of C++ exceptions: 0=disable and 1=enable.

Remarks

This method is a C++ extension to DOM Level 1.

getDoctype

Returns the DocumentType node associated with this document.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, getDoctype, DocumentType *)
END_METHOD
```

C++

```
DocumentType * METHOD_CALL getDoctype ();
```

Remarks

The DOM level does not support editing the Document Type Declaration; therefore, the DocumentType cannot be altered in any way.

For documents without a Document Type Declaration, this method returns NULL.

getDocumentElement

Returns the first child element of the Document.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, getDocumentElement, Element *)
END_METHOD
```

C++

```
Element * METHOD_CALL getDocumentElement ();
```

Remarks

This is a convenience method that allows direct access to the child node that is the root element of the document. If there is more than one element as a child of the Document object then the first element child is returned. If there is no document element, null is returned.

getElementsByTagName

Returns a NodeList of all the Elements with a given tag name in the order in which they would be encountered in a preordered traversal of the Document tree.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, getElementsByTagName, NodeList *)
   METHOD_PARM1(const unicode * tagName)
END_METHOD
```

C++

```
NodeList * METHOD_CALL getElementsByTagName (
   const unicode   *tagName);
```

Parameters

tagName

(IN) Points to the name of the element type to include in the NodeList?

Remarks

The returned NodeList must be deleted when it is no longer needed. This must be done by calling the NodeList::destroy method.

If this Node is deleted before the NodeList is destroyed, the NodeList becomes invalid and results are unpredictable.

getImplementation

Returns the DOMImplementation object that handles this document.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, getImplementation, DOMImplementation *)
END_METHOD
```

C++

```
DOMImplementation * MEHTOD_CALL getImplementation ():
```

Remarks

A DOM application may use objects from multiple implementations.

2.3 Element

The Element interface inherits from the Node interface.

By far the vast majority of objects (apart from text) that authors encounter when traversing a document are Element nodes. Assume the following XML document:

```
<elementExample id="demo">
   <subelement1/>
   <subelement2>
      <subsubelement/>
   </subelement2>
</elementExample>
```

When represented using DOM, the top node is an Element node for “elementExample,” which contains two child Element nodes, one for “subelement1” and one for “subelement2”. The “subelement1” node contains no child nodes.

Elements may have attributes associated with them. Since the Element interface inherits from Node, the generic Node interface method getAttributes may be used to retrieve the set of all attributes for an element. The Element interface contains methods to retrieve either an Attr object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an Attr object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience.

getAttribute

Returns the value of the specified attribute name.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, getAttribute, const unicode *)
   METHOD_PARM1(const unicode * name)//the name of the attribute
END_METHOD
```

C++

```
const unicode * METHOD_CALL getAttribute (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the attribute whose value is to be returned.

Remarks

If the element does not have an attribute with the passed name, null is returned.

getAttributeNode

Retrieves an Attribute node by name.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, getAttributeNode, Attr *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
Attr * METHOD_CALL getAttributeNode (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the attribute.

Remarks

This method returns null if the element does not contain an attribute by the passed name.

getElementsByTagName

Returns a NodeList of all descendant elements with a given tag name, in the order in which they would be encountered in a preordered traversal of the Element tree.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Document, getElementsByTagName, NodeList *)
   METHOD_PARM1(const unicode * tagName)
END_METHOD
```

C++

```
NodeList * METHOD_CALL getElementsByTagName (
   const unicode   *tagName);
```

Parameters

tagName

(IN) Points to the name of the element to find. The special tag name value “*” matches all tags.

Remarks

The returned NodeList must be deleted when it is no longer needed. This must be done by calling the NodeList::destroy method.

If this Node is deleted before the NodeList is destroyed, the NodeList becomes invalid and results are unpredictable.

getTagName

Returns the name of the element.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, getTagName, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getTagName ();
```

Return Values

Returns the tag name of the element.

normalize

Puts all Text nodes in the full depth of the subtree underneath this Element into a “normal” form where only markup (for example tags, comments, processing instructions, CDATA sections, and entity references) separates Text nodes. When a document is normalized, there are no adjacent Text nodes.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, normalize, DOMException *)
END_METHOD
```

C++

```
DOMException * METHOD_CALL normalize ();
```

Remarks

This method can be used to ensure that the DOM view of a document is the same if it were saved and re-loaded. It is useful when operations (such as XPointer lookups) that depend on a particular document tree structure are to be used.

removeAttribute

Removes an attribute by name.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, removeAttribute, DOMException *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
DOMException * METHOD_CALL removeAttribute (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the attribute to remove from the element.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Remarks

If the removed attribute has a default value, the attribute is immediately replaced.

removeAttributeNode

Removes the specified attribute.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, removeAttributeNode, Attr *)
   METHOD_PARM1(Attr * oldAttr)
END_METHOD
```

C++

```
Attr * METHOD_CALL removeAttributeNode (
   Attr   *oldAttr);
```

Parameters

oldAttr

(IN) Points to the Attr node to remove.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

Returns NOT_FOUND_ERR if oldAttr is not an attribute of this Node.

setAttribute

Adds an attribute with the specified value to the element.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, setAttribute, DOMException *)
   METHOD_PARM1(const unicode * name)
   METHOD_PARM (const unicode * value)
END_METHOD
```

C++

```
DOMException * METHOD_CALL setAttribute (
   const unicode   *name,
   const unicode   *value);
```

Parameters

name

(IN) Points to the name of the attribute.

value

(IN) Points to the value for the attribute.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Returns INVALID_CHARACTER_ERR if name is invalid.

Remarks

If an attribute with the specified name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string. It is not parsed as it is being set. Any markup (such as syntax to be recognized as an entity reference) is treated as literal text and needs to be appropriately escaped by the implementation when it is written out.

In order to assign an attribute value that contains entity references, the user must create an Attr node plus any Text and EntityReference nodes, build the appropriate subtree, and use the setAttributeNode method to assign it as the value of an attribute.

setAttributeNode

Adds a new attribute to an element.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Element, setAttributeNode, Attr *)
   METHOD_PARM1(Attr * newAttr)
END_METHOD
```

C++

```
Attr * METHOD_CALL setAttributeNode (
   Attr   *newAttr);
```

Parameters

newAttr

(IN) Points to the Attr node to add.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

Returns WRONG_DOCUMENT_ERR if this Node is of a type that does not allow children of the newChild type or if newChild is one of this Node's ancestors.

Returns INUSE_ATTRIBUTE_ERR if newAttr is an Attr that already belongs to another Element.

Remarks

If an attribute with that name is already present in the element, this attribute is replaced by the new one.

If the new Attr replaces an existing Attr node, the previously existing Attr node is returned (and must be destroyed); otherwise null is returned.

To create an attribute node, use Document::createAttribute method.

To set the attribute's value, use Attr::setValue method.

To add the attribute to an element, use Element::setAttributeNode method.

2.4 Attr

The Attr interface inherits from the Node interface and represents an attribute in an Element object.Typically the allowable values for the attribute are defined in a document type definition. See the nds.dtd for the DirXML document type definition.

Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the Node attributes parentNode, previousSibling, and nextSibling have a NULL value for Attr objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with. This makes it more efficient to implement such features as default attributes associated with all elements of a given type.

Attr nodes may not be immediate children of a DocumentFragment. However, they can be associated with Element nodes contained within a DocumentFragment. In short, users and implementors of the DOM need to be aware that Attr nodes have some things in common with other objects inheriting the Node interface, but they also are quite distinct.

The attribute's effective value is determined as follows:

- If this attribute has been explicitly assigned any value, that value is the attribute's effective value.
- if there is a declaration for this attribute, and that declaration includes a default value, then that default value is the attribute's effective value;
- If these is no default value, the attribute does not exist on this element in the structure model until it has been explicitly added.

The nodeValue attribute on the Attr instance can also be used to retrieve the string version of the attribute's value(s).

In XML, where the value of an attribute can contain entity references, the child nodes of the Attr node provide a representation in which entity references are not expanded. These child nodes may be either Text or EntityReference nodes. Because the attribute type may be unknown, there are no tokenized attribute values.

getName

Returns the name of the attribute.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Attr, getName, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getName ();
```

getSpecified

Indicates whether the attribute was explicitly given a value in the original document.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Attr, getSpecified, int)
END_METHOD
```

C++

```
int METHOD_CALL getSpecified ();
```

Remarks

If this attribute was explicitly given a value in the original document, this is TRUE; otherwise it is FALSE.

The implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value), then the specified flag is automatically flipped to TRUE. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with the specified flag set to FALSE and the default value (if one exists).

getValue

Returns the value of the attribute as a string. Character and general entity references are replaced with their values.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Attr, getValue, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getValue ();
```

setValue

Creates a Text node with the unparsed contents of the string.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Attr, setValue, DOMException *)
   METHOD_PARM1(const unicode * value)
END_METHOD
```

C++

```
DOMException * METHOD_CALL setValue (
   const unicode   *value);
```

Parameters

value

(IN) Points to the new attribute value.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

2.5 CharacterData

The CharacterData interface inherits from the Node interface and extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to CharacterData, though Text and others do inherit the interface from it. All offsets in this interface start from 0.

appendData

Appends the string to the end of the character data of the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, appendData, DOMException *)
   METHOD_PARM1(const unicode * arg)
END_METHOD
```

C++

```
DOMException * METHOD_CALL appendData (
   const unicode   *arg);
```

Parameters

arg

(IN) Points to the string to append to the character data.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

deleteData

Removes a range of characters from the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, deleteData, DOMException *)
   METHOD_PARM1(int offset)
   METHOD_PARM (int count)
END_METHOD
```

C++

```
DOMException * METHOD_CALL deleteData (
   int   offset,
   int   count);
```

Parameters

offset

(IN) Specifies the offset where characters should be removed.

count

(IN) Specifies the number of characters to delete.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Returns INDEX_SIZE_ERR if the specified count is negative or if the specified offset is negative or greater than the number of characters in data.

getData

Returns the character data of the node that implements this interface.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, getData, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getData ();
```

Remarks

The DOM implementation cannot put arbitrary limits on the amount of data that can be stored in a CharacterData node. However, implementation limits may mean that the entirety of a node's data may not fit into a single string. In such cases the user may call substringData to retrieve the data in appropriately sized pieces.

getLength

Returns the number of characters that are available through getData() and the substringData() method below.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, getLength, int)
END_METHOD
```

C++

```
int METHOD_CALL getLength ();
```

Remarks

This method can have a value of zero, for example, when the CharacterData nodes are empty.

insertData

Inserts a string at the specified character offset.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, insertData, DOMException *)
   METHOD_PARM1(int offset)
   METHOD_PARM (const unicode * arg)
END_METHOD
```

C++

```
DOMException * METHOD_CALL insertData (
   int              offset,
   const unicode   *arg);
```

Parameters

offset

(IN) Points to the character offset where the data is to be inserted.

arg

(IN) Points to the string to insert.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Returns INDEX_SIZE_ERR if the specified offset is negative or greater than the number of characters in data

replaceData

Replaces the characters starting at the specified character offset within the specified string.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, replaceData, DOMException *)
   METHOD_PARM1(int offset)
   METHOD_PARM (int count)
   METHOD_PARM (const unicode * arg)
END_METHOD
```

C++

```
DOMException * METHOD_CALL replaceData (
   int              offset,
   int              count,
   const unicode   *arg);
```

Parameters

offset

(IN) Specifies the character offset where the replacement should begin.

count

(IN) Specifies the number of characters to replace.

arg

(IN) Points to the string to use for the replacement characters.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Returns INDEX_SIZE_ERR if the specified count is negative or if the specified offset is negative or greater than the number of characters in data.

setData

Sets the character data of the node that implements this interface.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, setData, DOMException *)
   METHOD_PARM1(const unicode * data)
END_METHOD
```

C++

```
DOMException * METHOD_CALL setData (
   const unicode   *data);
```

Parameters

data

(IN) Points to the character data for the node.

Return Values

If unsuccessful, returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if Node is read-only.

substringData

Extracts a range of data from the node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (CharacterData, substringData, const unicode *)
   METHOD_PARM1(int offset)
   METHOD_PARM (int count)
END_METHOD
```

C++

```
const unicode * METHOD_CALL substringData (
   int   offset,
   int   count);
```

Parameters

offset

(IN) Specifies the start offset of the substring to extract.

count

(IN) Specifies the number of characters to extract.

Return Values

If unsuccessful, registers or throws a DOMException.

Returns INDEX_SIZE_ERR if

- The specified offset is negative or greater than the number of characters in data
- The specified count is negative.

2.6 Text

The Text interface inherits from the CharacterData interface and represents the textual content (termed character data in XML) of an Element or Attr. If there is no markup inside an element's content, the text is contained in a single object implementing the Text interface that is the only child of the element. If there is markup, it is parsed into a list of elements and Text nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one Text node for each block of text. Users may create adjacent Text nodes that represent the contents of a given element without any intervening markup, but they should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not (in general) persist between DOM editing sessions. The normalize method on Element merges any such adjacent Text objects into a single node for each block of text; this is recommended before employing operations that depend on a particular document structure, such as navigation with XPointers.

splitText

Breaks this Text node into two Text nodes at the specified offset, keeping both in the tree as siblings.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Text, splitText, Text *)
   METHOD_PARM1(int offset)
END_METHOD
```

C++

```
Text * METHOD_CALL splitText (
   int   offset);
```

Parameters

offset

(IN) Specifies at which offset to split node, starting from 0.

Return Values

Registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Returns INDEX_SIZE_ERR if the specified offset is negative or greater than the number of characters in data.

Remarks

A split text node contains all the content up to the offset point. A new Text node, which is inserted as the next sibling of this node, contains all the content at and after the offset point.

2.7 ProcessingInstruction

The ProcessingInstruction interface inherits from the Node interface and represents a “processing instruction,” used in XML as a way to keep processor-specific information in the text of the document.

getData

Returns the content of this ProcessingInstruction node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (ProcessingInstruction, getData, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getData ();
```

Remarks

Returns the data from the first non-white space character after the target to the character immediately preceding the ?>.

getTarget

Returns the target of this ProcessingInstruction node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (ProcessingInstruction, getTarget, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getTarget ();
```

Remarks

XML defines the target of a ProcessingInstruction node as the first token following the markup that begins the processing instruction.

setData

Sets the content of this ProcessingInstruction node.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (ProcessingInstruction, setData, DOMException *)
   METHOD_PARM1(const unicode * data)
END_METHOD
```

C++

```
DOMException * METHOD_CALL setData (
   const unicode   *data);
```

Parameters

data

Points to the processing instructions.

Return Values

Returns or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the Node is read-only.

Remarks

Sets the contents from the first non-white space character after the target to the character immediately preceding the ?>.

2.8 DocumentType

The DocumentType interface inherits from the Node interface and provides an interface to the list of entities that are defined for the document. Each Document has a doctype attribute whose value is either null or a DocumentType object. The DocumentType interface in the DOM Level 1 Core specifies an interface to the list of entities that are defined for the document, and little else because the effect of name spaces and the various XML scheme efforts on DTD representation are not yet clearly defined.

The DOM Level 1 specification doesn't support editing DocumentType nodes.

getEntities

Returns a NamedNodeMap containing the general entities, both external and internal, declared in the DTD.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DocumentType, getEntities, NamedNodeMap *)
END_METHOD
```

C++

```
NamedNodeMap * METHOD_CALL getEntities ();
```

Remarks

Duplicate entities are discarded. Every Node in this map also implements the Entity interface.

The returned NamedNodeMap belongs to this node. No attempt must be made to delete it.

getName

Returns the name of the DTD, which is the name immediately following the DOCTYPE keyword.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DocumentType, getName, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getName ();
```

getNotations

Returns a NamedNodeMap containing the notations declared in the DTD.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DocumentType, getNotations, NamedNodeMap *)
END_METHOD
```

C++

```
NamedNodeMap * MEHTOD_CALL getNotations ();
```

Remarks

Duplicate notations are discarded. Every Node in this map also implements the Notation interface.

The returned NamedNodeMap belongs to this node. No attempt must be made to delete it.

2.9 DOMImplementation

The DOMImplementation interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

The DOM Level 1 does not specify a way of creating a document instance, and hence document creation is an operation specific to an implementation. Future levels of the DOM specification are expected to provide methods for creating documents directly.

hasFeature

Discovers if the DOM implementation implements a specific feature.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DOMImplementation, hasFeature, int)
   METHOD_PARM1(const unicode * feature)
   METHOD_PARM (const unicode * version)
END_METHOD
```

C++

```
int METHOD_CALL hasFeature (
   const unicode   *feature,
   const unicode   *version);
```

Parameters

feature

(IN) Points to the package name of the feature to test. In Level 1, the legal values for feature are “HTML” and “XML” (case insensitive).

version

(IN) Points to the version number of the package name to test. In Level 1, the legal value is the string “1.0”. If the version is not specified, supporting any version of the feature will cause the method to return non-zero (TRUE).

Return Values

Returns non-zero if the feature is supported.

Returns zero (0) if the feature is not supported.

2.10 Entity

The Entity interface inherits from the Node interface and represents an entity, either parsed or unparsed, in an XML document.This interface models the entity itself not the entity declaration. Entity declaration modeling has been left for a later level of the DOM specification.

The nodeName attribute that is inherited from Node contains the name of the entity.

An XML processor may choose to completely expand entities before the structure model is passed to the DOM; in this case there will be no EntityReference nodes in the document tree.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available. When the replacement value is available, the corresponding Entity node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

The resolution of the children of the Entity (the replacement value) may be lazily evaluated; actions by the user (such as calling the childNodes method on the Entity Node) are assumed to trigger the evaluation.

The DOM Level 1 does not support editing Entity nodes; if a user wants to make changes to the contents of an Entity, every related EntityReference node has to be replaced in the structure model by a clone of the Entity's contents, and then the desired changes must be made to each of those clones instead. All the descendants of an Entity node are read-only.

An Entity node does not have a parent.

getNotationName

For unparsed entities, returns the name of the notation for the entity.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Entity, getNotationName, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getNotationName ();
```

Remarks

For parsed entities, this method returns null.

getPublicId

Returns the public identifier associated with the entity, if specified.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Entity, getPublicId, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getPublicId ();
```

Remarks

If the public identifier was not specified, this method returns null.

getSystemId

Returns the system identifier associated with the entity, if specified.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Entity, getSystemId, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getSystemId ();
```

Remarks

If the system identifier was not specified, this method returns null.

2.11 Notation

The Notation interface inherits from the Node interface and represents a notation declared in the DTD. A notation does one of the following:

- Declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification).
- Makes a formal declaration of Processing Instruction targets (see section 2.6 of the XML 1.0 specification).

The nodeName attribute inherited from Node is set to the declared name of the notation.

Notation nodes are read-only.

A Notation node does not have a parent.

getPublicId

Returns the public identifier of this notation.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Entity, getPublicId, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getPublicId ();
```

Remarks

If a public identifier was not specified, this method returns null.

getSystemId

Returns the system identifier of this notation.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Entity, getSystemId, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getSystemId ();
```

Remarks

If a system identifier was not specified, this method returns null.

2.12 NamedNodeMap

Objects implementing the NamedNodeMap interface represent collections of nodes that can be accessed by name. NamedNodeMap does not inherit from NodeList.

NamedNodeMaps are not maintained in any particular order. Objects contained in an object implementing NamedNodeMap may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a NamedNodeMap and does not imply that the DOM specifies an order to these Nodes.

getLength

Returns the number of nodes in the map.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NamedNodeMap, getLength, int)
END_METHOD
```

C++

```
int METHOD_CALL getLength ();
```

Remarks

The range of valid child node indices is 0 to length-1, inclusive.

getNamedItem

Retrieves a node specified by name.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NamedNodeMap, getNamedItem, Node *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
Node * METHOD_CALL getNamedItem (
   const unicode   *name);
```

Parameters

name

Points to the name of the node to retrieve.

item

Returns the “index”th item in the map.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NamedNodeMap, item, Node *)
   METHOD_PARM1(int index)
END_METHOD
```

C++

```
Node * MEHTOD_CALL item (
   int   index);
```

Parameters

index

(IN) Specifies the index number of a node in the map.

Remarks

If the index is greater than or equal to the number of nodes in the map, this method returns null.

removedNamedItem

Removes a node from the NamedNodeMap using the node's specified by name.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NamedNodeMap, removeNamedItem, Node *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
Node * METHOD_CALL removeNamedItem (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the node to remove.

Return Values

Registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if the NamedNodeMap is read-only.

Returns NOT_FOUND_ERR if no Node with name is in the map.

Remarks

If the removed node is an Attr with a default value, the node is immediately replaced.

setNamedItem

Adds a node to a NamedNodeMap using the node's nodeName attribute.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NamedNodeMap, setNamedItem, Node *)
   METHOD_PARM1(Node * arg)
END_METHOD
```

C++

```
Node * MEHTOD_CALL setNamedItem (
   Node   *arg);
```

Parameters

arg

(IN) Points to the nodeName of the node to add.

Return Values

Registers or throws a DOMException.

Returns NO_MODIFICATION_ALLOWED_ERR if NamedNodeMap is read-only.

Returns WRONG_DOCUMENT_ERR if arg was created with a different Document than the NamedNodeMap belongs to.

Returns INUSE_ATTRIBUTE_ERR if arg is an Attr that already belongs to another Element.

Remarks

As the node name attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a “special” string value) cannot be stored in a NamedNodeMap because the names would clash.

2.13 NodeList

The NodeList interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.

The items in the NodeList are accessible with an integral index, starting from 0.

This interface contains one extension to DOM Level 1, the destroy method.

destroy

Deletes the NodeList.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (Node, destroy, void)
END_METHOD
```

C++

```
void METHOD_CALL destroy ();
```

Remarks

This method must be called when the NodeList is no longer needed.

This method is a C++ extension to DOM Level 1.

getLength

Returns the number of nodes in the list.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NodeList, getLength, int)
END_METHOD
```

C++

```
int METHOD_CALL getLength ();
```

Remarks

The range of valid child node indices is 0 to length-1, inclusive.

item

Returns the “index”th item in the list.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (NodeList, item, Node *)
   METHOD_PARM1(int index)
END_METHOD
```

C++

```
Node * METHOD_CALL item (
   int   index);
```

Parameters

int

(IN) Specifies the index of the node to return.

Remarks

If the index is greater than or equal to the number of nodes in the list, this method returns null.

2.14 DOMException

DOM operations only raise exceptions in “exceptional” circumstances, for example when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). In general, DOM methods return specific error values in ordinary processing situation, such as out-of-bound errors when using NodeList.

Implementations may raise other exceptions under other circumstances. For example, implementations may raise an implementation-dependent exception if a null argument is passed.

Some languages and object systems do not support the concept of exceptions. For such systems, error conditions may be indicated using native error reporting mechanisms. For some bindings, for example, methods may return error codes similar to those listed in the corresponding method descriptions.

destroy

Deletes this exception.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DOMException, destroy, void)
END_METHOD
```

C++

```
void METHOD_CALL destroy ();
```

Remarks

This is a C++ extension to DOM Level 1.

getCode

Returns the DOM-defined exception code associated with this exception.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DOMException, getCode, int)
END_METHOD
```

C++

```
int METHOD_CALL getCode ();
```

getMessage

Returns the text message associated with the exception.

Syntax

Defining Macros for C++

```
#include "dom.h"

METHOD (DOMException, getMessage, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getMessage ();
```

Remarks

This method returns null if no message was associated with the exception.

This method is a C++ extension to DOM Level 1.

Simple API for XML (SAX) Interfaces

3

The following C++ interfaces and methods for receiving information about XML documents are patterned after the Java implementation of version 1 of the Simple API for XML (SAX) event interface. For additional documentation about this interface, see Megginson Technologies (http://www.megginson.com/SAX/index.html).

Exceptions are not enabled by default. Instead, the methods return a pointer to a SAXException interface. If the method returns null, there was no error.

The Parser, DocumentHandler and DTDHandler interfaces have a method to enable exceptions. To enable exceptions for C++, you must call the enableExceptions method with a non-zero value.

- If exceptions have not been implemented in C++, these methods return 0.
- If they have been implemented in C++, these methods return the exception-enabled state.

Since C++ allows overloading, C++ methods use the same name as the Java methods.

The SAX parser implements the following interfaces:

- Section 3.1, “AttributeList,” on page 111
- Section 3.7, “Locator,” on page 150
- Section 3.8, “Parser,” on page 156
- Section 3.9, “SAXException,” on page 163
- Section 3.10, “SAXParseException,” on page 165

The SAX application implements the following interfaces:

- Section 3.2, “DocumentHandler,” on page 118
- Section 3.3, “DTDHandler,” on page 129
- Section 3.4, “EntityResolver,” on page 134
- Section 3.5, “ErrorHandler,” on page 135
- Section 3.6, “InputSource,” on page 139
- Section 3.9, “SAXException,” on page 163
- Section 3.10, “SAXParseException,” on page 165

3.1 AttributeList

The AttributeList interface allows access to an element’s attribute specifications.

The SAX parser implements this interface and passes an instance to the SAX application as the second argument of each startElement event.

The instance provided returns valid results only during the scope of the startElement invocation. To save it for future use, the application must make a copy; the AttributeListImpl helper class provides a convenient constructor for making a copy.

An AttributeList includes only attributes that have been specified or have default values; #IMPLIED attributes will not be included.

There are two ways for the SAX application to obtain information from the AttributeList. First, it can iterate through the entire list. The following code sample illustrates this:

```
SAXException * startElement (const unicode * name, AttributeList *
atts) {
for (int i = 0; i < atts->getLength(); i++) {
const unicode * name = atts->getName(i);
const unicode * type = atts->getType(i);
const unicode * value = atts->getValue(i);
[...]
}
}
```

The result of the getLength method will be zero if there are no attributes.

As an alternative, the application can request the value or type of specific attributes. The following code illustrates this method.

```
SAXException * startElement (const unicode * name, AttributeList *
atts) {
const unicode * identifier = atts->getValue(L"id");
const unicode * label = atts->getValue(L"label");
[...]
}
```

DirXML provides an implementation of AttributeList via the C++ class AttributeListImpl.

getLength

Returns the number of attributes in the list.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (AttributeList, getLength, int)
END_METHOD
```

C++

```
int METHOD_CALL getLength ();
```

Remarks

If the element has no attributes, this method returns zero.

getName

Returns the name of an attribute in this list by specifying its position.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (AttributeList, getName, const unicode *)
   METHOD_PARM1(int index)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getName (
   int   index);
```

Parameters

index

(IN) Specifies the attribute's index in the 0-based index of attributes.

Remarks

This method returns null if the specified index is out of range.

getType

Returns the type of an attribute in the list by specifying its position.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (AttributeList, getType, const unicode *)
   METHOD_PARM1(int index)
END_METHOD
```

C++

```
const unicode * MEHTOD_CALL getType (
   int   index);
```

Parameters

index

(IN) Specifies the attribute's index in the 0-based index of attributes.

Remarks

This method returns null if the specified index is out of range.

The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES" or "NOTATION" (always in upper case).

getType

Returns the type of an attribute in the list by specifying its name.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (AttributeList, getType, const unicode *)
   METHOD_PARM1(const unicode * name)//name of attribute
END_METHOD
```

C++

```
const unicode * METHOD_CALL getType (
   const unicode *name);
```

Parameters

name

(IN) Points to the name of the attribute.

Remarks

This method returns null if the passed name doesn’t match an attribute in the list.

The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES" or "NOTATION" (always in upper case).

getValue

Returns the value of an attribute in the list by specifying its position.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (AttributeList, getValue, const unicode *)
   METHOD_PARM1(int index)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getValue (
   int   index);
```

Parameters

index

(IN) Specifies the attribute's index in the 0-based index of attributes.

Remarks

This method returns null if the specified index is out of range.

If the attribute value is a list of tokens (IDREFS, ENTITIES, or NMTOKENS), the tokens are concatenated into a single string separated by whitespace.

getValue

Returns the value of an attribute in the list by specifying its name.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (AttributeList, getValue, const unicode *)
   METHOD_PARM1(const unicode * name)//name of attribute
END_METHOD
```

C++

```
const unicode * MEHTOD_CALL getValue (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the attribute.

Remarks

This method returns null if the passed name doesn't match an attribute in the list.

If the attribute value is a list of tokens (IDREFS, ENTITIES, or NMTOKENS), the tokens will be concatenated into a single string separated by whitespace.

3.2 DocumentHandler

This interface receives notification of general document events.

This is the main interface that most SAX applications implement. If the application needs to be informed of basic parsing events, it implements this interface and registers an instance with the SAX parser using the setDocumentHandler method. The parser uses the instance to report basic document-related events like the start and end of elements and character data.

The order of events in this interface is very important, and mirrors the order of information in the document itself. For example, all of an element's content (character data, processing instructions, and/or subelements) will appear, in order, between the startElement event and the corresponding endElement event.

Application writers who do not want to implement the entire interface can derive a class from HandlerBase, which implements the default functionality; parser writers can instantiate HandlerBase to obtain a default handler. The application can find the location of any document event using the Locator interface supplied by the Parser through the setDocumentLocator method.

This specification extends the Java org.xml.sax.DocumentHandler with the following methods:

```
SAXException * comment (const unicode * data);
int enableExceptions (int enable);
```

characters

Receives notification of character data.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, characters, SAXException *)
   METHOD_PARM1(const unicode ch[])
   METHOD_PARM (int start)
   METHOD_PARM (int length)
END_METHOD
```

C++

```
SAXException * METHOD_CALL characters (
  const unicode   ch[],
  int             start,
  int             length);
```

Parameters

ch

Specifies the character array.

start

Specifies the first character.

length

Specifies the number of characters.

Remarks

The Parser calls this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks. However, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Some parsers report whitespace using the ignorableWhitespace method rather than this one (validating parsers must do so).

comment

Receives notification of a comment.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, comment, SAXException *)
   METHOD_PARM1(const unicode * data)
END_METHOD
```

C++

```
SAXException * METHOD_CALL comment (
   const unicode   *data);
```

Parameters

data

Points to the comment string.

Remarks

The parser invokes this method once for each comment found. This method does not exist in the Java interface.

enableExceptions

Enables the throwing of C++ exceptions for all methods in this interface.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, enableExceptions, int)
   METHOD_PARM1(int enable)
END_METHOD
```

C++

```
int METHOD_CALL enableExceptions (
  int   enable);
```

Parameters

enable

(IN) Specifies whether C++ exceptions are enabled: 0= disabled, 1=enabled.

Remarks

This is a C++ extension to the interface.

endDocument

Receives notification of the end of a document.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, endDocument, SAXException *)
END_METHOD
```

C++

```
SAXException * METHOD_CALL endDocument ();
```

Remarks

The SAX parser invokes this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

endElement

Receives notification of the end of an element.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, endElement, SAXException *)
   METHOD_PARM1(const unicode * name)
END_METHOD
```

C++

```
SAXException * METHOD_CALL endElement (
   const unicode   *name);
```

Parameters

name

(IN) Points to the name of the element.

Remarks

The SAX parser invokes this method at the end of every element in the XML document; there will be a corresponding startElement() event for every endElement() event (even when the element is empty).

If the element name has a name space prefix, the prefix will still be attached to the name.

ignorableWhitespace

Receives notification of ignorable whitespace in element content.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, ignorableWhitespace, SAXException *)
   METHOD_PARM1(const unicode ch[])
   METHOD_PARM (int start)
   METHOD_PARM (int length)
END_METHOD
```

C++

```
SAXException * METHOD_CALL ignorableWhitespace (
   const unicode   ch[],
   int             start,
   int             length);
```

Parameters

ch

Specifies the character array.

start

Specifies the first character.

length

Specifies the number of characters.

Remarks

Validating parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 Recommendation, Section 2.10). Non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks. However, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

processingInstruction

Receives notification of a processing instruction.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD(DocumentHandler,processingInstruction,SAXException *)
   METHOD_PARM1(const unicode * target)
   METHOD_PARM (const unicode * data)
END_METHOD
```

C++

```
SAXException * METHOD_CALL processingInstruction (
   const unicode   *target,
   const unicode   *data);
```

Parameters

target

Points to the processing instruction target.

data

Points to the data string.

Remarks

The parser invokes this method once for each processing instruction found. Processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, Section 2.8) or a text declaration (XML 1.0, Section 4.3.1) using this method.

setDocumentLocator

Receives an object for locating the origin of SAX document events.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, setDocumentLocator, void)
   METHOD_PARM1(Locator * locator)
END_METHOD
```

C++

```
void METHOD_CALL setDocumentLocator (
   Locator   *locator);
```

Parameters

locator

(IN) Points to a locator object.

Remarks

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator. If it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

The locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

startDocument

Receives notification of the beginning of a document.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, startDocument, SAXException *)
END_METHOD
```

C++

```
SAXException * METHOD_CALL startDocument ();
```

Remarks

The SAX parser invokes this method only once, before any other methods in this interface or in DTDHandler (except for the setDocumentLocator method).

startElement

Receives notification of the beginning of an element.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DocumentHandler, startElement, SAXException *)
   METHOD_PARM1(const unicode * name)
   METHOD_PARM (AttributeList * atts)
END_METHOD
```

C++

```
SAXException * METHOD_CALL startElement (
   const unicode   *name,
   AttributeList   *atts);
```

Parameters

name

(IN) Points to the name of the element.

atts

(IN) Points to the attribute list for the element.

Remarks

The Parser invokes this method at the beginning of every element in the XML document; there will be a corresponding endElement event for every startElement event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding endElement() event.

If the element name has a name space prefix, the prefix will still be attached. The attribute list provided will contain only attributes with explicit values (specified or defaulted); #IMPLIED attributes will be omitted.

3.3 DTDHandler

This interface receives notification of basic DTD-related events.

If a SAX application needs information about notations and unparsed entities, then the application implements this interface and registers an instance with the SAX parser using the parser's setDTDHandler method. The parser uses the instance to report notation and unparsed entity declarations to the application.

The SAX parser may report these events in any order, regardless of the order in which the notations and unparsed entities were declared; however, all DTD events must be reported after the document handler’s startDocument event, and before the first startElement event.

It is up to the application to store the information for future use (perhaps in a hash table or object tree). If the application encounters attributes of type “NOTATION”, “ENTITY”, or “ENTITIES”, it can use the information that it obtained through this interface to find the entity and/or notation corresponding with the attribute value.

The specification extends the Java org.xml.sax.DTDHandler with the following methods:

```
int enableExceptions (int enable);
```

enableExceptions

Enables the throwing of C++ exceptions for all methods in this interface.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DTDHandler, enableExceptions, int)
   METHOD_PARM1(int enable)
END_METHOD
```

C++

```
int METHOD_CALL enableExceptions (
   int   enable);
```

Parameters

enable

(IN) Specifies whether C++ exceptions are enabled: 0= disabled, 1=enabled.

Remarks

This is a C++ extension to the SAX specification.

notationDecl

Receives notification of a notation declaration event.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DTDHandler, notationDecl, SAXException *)
   METHOD_PARM1(const unicode * name)
   METHOD_PARM (const unicode * publicId)
   METHOD_PARM (const unicode * systemId)
END_METHOD
```

C++

```
SAXException * METHOD_CALL notationDecl (
   const unicode   *name,
   const unicode   *publicId,
   const unicode   *systemId);
```

Parameters

name

Points to the name for this notation.

publicId

Points to the public identifier for this notation.

systemId

Points to the system identifier for this notation.

Remarks

The application is responsible to record the notation for later reference, if necessary.

If a system identifier is present, and it is a URL, the SAX parser must resolve it fully before passing it to the application.

unparsedEntityDecl

Receives notification of an unparsed entity declaration event.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (DTDHandler, unparsedEntityDecl, SAXException *)
   METHOD_PARM1(const unicode * name)
   METHOD_PARM (const unicode * publicId)
   METHOD_PARM (const unicode * systemId)
   METHOD_PARM (const unicode * notationName)
END_METHOD
```

C++

```
SAXException * METHOD_CALL unparsedEntityDec1 (
   const unicode   *name,
   const unicode   *publicId,
   const unicode   *systemId,
   const unicode   *notationName);
```

Parameters

name

Points to the name of the entity declaration.

publicId

Points to the public identifier for this entity.

systemId

Points to the system identifier for this entity.

notationName

Points to entity name.

Remarks

The notation name corresponds to a notation reported by the notationDecl() event. The application is responsible to record the entity for later reference, if necessary.

If the system identifier is a URL, the parser must resolve it fully before passing it to the application.

3.4 EntityResolver

If a SAX application needs to implement customized handling for external entities, it must implement this interface and register an instance with the SAX parser using the parser's setEntityResolver method.

The parser then allows the application to intercept any external entities (including the external DTD subset and external parameter entities, if any) before including them.

Many SAX applications will not need to implement this interface, but it will be especially useful for applications that build XML documents from databases or other specialized input sources, or for applications that use URI types other than URLs.

resolveEntity

Allows the application to resolve external entities.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (EntityResolver, resolveEntity, InputSource *)
   METHOD_PARM1(const unicode * publicId)
   METHOD_PARM (const unicode * systemId)
END_METHOD
```

C++

```
InputSource * METHOD_CALL resolveEntity (
   const unicode   *publicId,
   const unicode   *systemId);
```

Parameters

publicId

Points to the entity's public identifier.

systemId

Points to the entity's system identifier.

Remarks

The parser calls this method before opening any external entity except the top-level document entity (including the external DTD subset, external entities referenced within the DTD, and external entities referenced within the document element). The application may request that the parser resolve the entity itself, that it use an alternative URI, or that it use an entirely different input source.

Application writers can use this method to redirect external system identifiers

- To use secure or local URIs
- To look up public identifiers in a catalogue
- To read an entity from a database or other input source (including, for example, a dialog box).

If the system identifier is a URL, the SAX parser must resolve it fully before reporting it to the application.

3.5 ErrorHandler

This interface is the basic interface for SAX error handlers.

If a SAX application needs to implement customized error handling, it must implement this interface and then register an instance with the SAX parser using the parser's setErrorHandler method. The parser will then report all errors and warnings through this interface.

The parser shall use this interface instead of throwing an exception: it is up to the application whether to throw an exception for different types of errors and warnings. There is no requirement that the parser continue to provide useful information after a call to fatalError; in other words, a SAX driver class could catch an exception and report a fatalError.

error

Receives notification of a recoverable error.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (ErrorHandler, error, void)
   METHOD_PARM1(SAXParseException * exception)
END_METHOD
```

C++

```
void METHOD_CALL error (
   SAXParseException   *exception);
```

Parameters

exception

Points to the SAX exception.

Remarks

This corresponds to the definition of “error” in Section 1.2 of the W3C XML 1.0 Recommendation. For example, a validating parser would use this callback to report the violation of a validity constraint. The default behavior is to take no action.

The SAX parser must continue to provide normal parsing events after invoking this method. The application should be able to process the document through to the end. If the application cannot do so, then the parser should report a fatal error even if the XML 1.0 Recommendation does not require it to do so.

fatalError

Receives notification of a non-recoverable error.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (ErrorHandler, fatalError, void)
   METHOD_PARM1(SAXParseException * exception)
END_METHOD
```

C++

```
void METHOD_CALL fatalError (
   SAXParseException   *exception);
```

Parameters

exception

Points to the SAX exception.

Remarks

This corresponds to the definition of “fatal error” in Section 1.2 of the W3C XML 1.0 Recommendation. For example, a parser would use this callback to report the violation of a well-formedness constraint.

The application must assume that the document is unusable after the parser has invoked this method, and should continue (if at all) only for the sake of collecting addition error messages. SAX parsers are free to stop reporting any other events once this method has been invoked.

warning

Receives notification of a warning.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (ErrorHandler, warning, void)
   METHOD_PARM1 (SAXParseException * exception)
END_METHOD
```

C++

```
void METHOD_CALL warning (
   SAXParseException   *exception);
```

Parameters

exception

Points to the SAX exception.

Remarks

SAX parsers use this method to report conditions that are not errors or fatal errors as defined by the XML 1.0 Recommendation. The default behavior is to take no action.

The SAX parser must continue to provide normal parsing of events after invoking this method. The application should still be able to process the document through to the end.

3.6 InputSource

The InputSource interface allows a SAX application to encapsulate information about an input source in a single object, which may include a public identifier, a system identifier, a byte stream (possibly with a specified encoding), and/or a character stream.

There are two places that the application will deliver this input source to the parser: as the argument to the Parser.parse method, or as the return value of the EntityResolver.resolveEntity method.

An InputSource object belongs to the application: the SAX parser shall never modify it in any way. It can only modify a copy, if necessary.

This interface diverges from the Java interface because of the lack of a common stream functionality in C++.

The primary divergence is the readByteChunk method which the SAX parser can call to read a chunk of the data stream at a time.

DirXML provides an implementation with the C++ InputSourceImpl class accessed through the interface.h file.

getByteStream

Gets the byte stream for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, getByteStream, const unsigned char *)
   METHOD_PARM1(int * length)
END_METHOD
```

C++

```
const unsigned char * METHOD_CALL getByteStream (
   int   *length);
```

Parameters

length

(OUT) Points to a variable that receives the length of the array.

Remarks

This method returns null if no byte stream was set for the input source.

The getEncoding method will return the character encoding for this byte stream, or null if unknown.

getCharacterStream

Gets the character stream for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, getCharacterStream, const unicode *)//array or
characters or 0
   METHOD_PARM1(int * length)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getCharacterStream (
   int   *length);
```

Parameters

length

(OUT) Points to the length of the array in characters.

Remarks

This method returns null if no character stream was set.

getEncoding

Gets the character encoding for a byte stream or URI.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, getEncoding, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL ();
```

Remarks

This method returns null if no encoding was set.

getPublicId

Gets the public identifier for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, getPublicId, const char *)
END_METHOD
```

C++

```
const char * METHOD_CALL getPublicId ();
```

Remarks

This method returns null if no public identifier was supplied.

getSystemId

Gets the system identifier for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, getSystemId, const char *)
END_METHOD
```

C++

```
const char * METHOD_CALL getSystemId ();
```

Remarks

This method returns null if no system identifier was supplied.

If the system identifier is a URL, it will be fully resolved.

readByteChunk

Returns the number of bytes passed from the input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, readByteChunk, int)
   METHOD_PARM1 (int count)
   METHOD_PARM (unsigned char * buffer)
END_METHOD
```

C++

```
int METHOD_CALL readByteChunk (
   int              count,
   unsigned char   *buffer);
```

Parameters

count

(IN) Specifies the number of bytes to read.

buffer

(OUT) Points to the buffer for the returned bytes.

Remarks

The passed buffer must be at least as large as the count parameter.

When the return value is less than count, the source is exhausted.

setByteStream

Sets the byte stream for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, setByteStream, void)
   METHOD_PARM1 (const unsigned char * byteStream)
   METHOD_PARM (int length)
END_METHOD
```

C++

```
void METHOD_CALL setByteStream (
   const unsigned char   *byteStream,
   int                    length);
```

Parameters

byteStream

(IN) Points to an array of bytes.

length

(IN) Specifies the length of the array in bytes.

Remarks

The SAX parser ignores the byte stream if there is also a character stream specified. However, the parser will use a byte stream in preference to opening a URI connection itself.

If the application knows the character encoding of the byte stream, it should set it with the setEncoding method.

setCharacterStream

Sets the character stream for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, setCharacterStream, void)
   METHOD_PARM1 (const unicode * charStream)
   METHOD_PARM (int length)
END_METHOD
```

C++

```
void METHOD_CALL setCharacterStream (
   const unicode   *charStream,
   int              length);
```

Parameters

charStream

(IN) Points to an array of characters.

length

(IN) Specifies the length of the array in characters.

Remarks

If there is a character stream specified, the SAX parser will ignore any byte stream and will not attempt to open a URI connection to the system identifier.

setEncoding

Sets the character encoding.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, setEncoding, void)
   METHOD_PARM1(const unicode * encoding)
END_METHOD
```

C++

```
void METHOD_CALL setEncoding (
   const unicode   *encoding);
```

Parameters

encoding

(IN) Points to the encoding string to use for this input source.

Remarks

The encoding must be a string acceptable for an XML encoding declaration (see Section 4.3.3 of the XML 1.0 Recommendation).

This method has no effect when the application provides a character stream.

setPublicId

Sets the public identifier for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, setPublicId, void)
   METHOD_PARM1(const char * publicId)
END_METHOD
```

C++

```
void METHOD_CALL setPublicId (
   const char   *publicId);
```

Parameters

publicId

(IN) Points to the public identifier to use for the input source.

Remarks

The public identifier is always optional. If the application writer includes one, it will be provided as part of the location information.

setSystemId

Sets the system identifier for this input source.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (InputSource, setSystemId, void)
   METHOD_PARM1(const char * systemId)
END_METHOD
```

C++

```
void METHOD_CALL setSystemId (
   const char   *systemId);
```

Parameters

systemId

(IN) Points to the system identifier to use for the input source.

Remarks

The system identifier is optional if there is a byte stream or a character stream. However, it is still useful to provide one, since the application can use it to resolve relative URIs and can include it in error messages and warnings. The parser will attempt to open a connection to the URI only if there is no byte stream or character stream specified.

If the application knows the character encoding of the object pointed to by the system identifier, it can register the encoding using the setEncoding method.

If the system identifier is a URL, it must be fully resolved.

3.7 Locator

The Locator interface associates a SAX event with a document location.

If a SAX parser provides location information to the SAX application, it does so by implementing this interface and then passing an instance to the application using the document handler’s setDocumentLocator method. The application can use the object to obtain the location of any other document handler event in the XML source document.

The results returned by the object are valid only during the scope of each document handler method; the application will receive unpredictable results if it attempts to use the locator at any other time.

SAX parsers are not required to supply a locator, but they are very strong encouraged to do so. If the parser supplies a locator, it must do so before reporting any other document events. If no locator has been set by the time the application receives the startDocument event, the application should assume that a locator is not available.

This specification extends the Java org.xml.sax.Locator with the following methods:

```
void destroy ();
```

The destroy method is required so that SAXParseException implementations can destroy the Locator object that they may have.

destroy

Destroys this locator object.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Locator, destroy, void)
END_METHOD
```

C++

```
void METHOD_CALL destroy ();
```

Remarks

This method must not be called by an application. This method must only be called by an implementation of SAXParseException.

getColumnNumber

Returns the column number where the current document event ends.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Locator, getColumnNumber, int)
END_METHOD
```

C++

```
int METHOD_CALL getColumnNumber ();
```

Remarks

This method returns -1 if the column number is not available.

The column number is the first character after the text associated with the document event. The first column in a line is position 1.

getLineNumber

Returns the line number where the current document event ends.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Locator, getLineNumber, int)
END_METHOD
```

C++

```
int METHOD_CALL getLineNumber ();
```

Remarks

This method returns -1 if the line number is not available.

The line position is the first character after the text associated with the document event.

getPublicId

Returns the public identifier for the current document event.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Locator, getPublicId, const unicode *)
END_METHOD
```

C++

```
const unicode * MEHTOD_CALL getPublicId ();
```

Remarks

This method returns null if no public identifier is available.

getSystemId

Returns the system identifier for the current document event.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Locator, getSystemId, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getSystemId ();
```

Remarks

This method returns null if no system identifier is available.

If the system identifier is a URL, the parser must resolve it fully before passing it to the application.

3.8 Parser

The Parser interface is the basic interface for SAX (Simple API for XML) parsers.

All SAX parsers must implement this basic interface; it allows applications to register handlers for different types of events and to initiate a parse from a URI, or a character stream.

All SAX parsers must also implement a zero-argument constructor (though other constructors are also allowed).

SAX parsers are reusable but not re-entrant; the application may reuse a parser object (possibly with a different input source) once the first parse has completed successfully, but it may not invoke the parse() methods recursively within a parse.

The SAX Parser interface is useful for any reporting of XML structure events. For example, this interface can convert a DOM tree to a series of SAX events.

This specification extends the Java org.xml.sax.DocumentHandler with the following methods:

```
int enableExceptions (int enable);
```

enableExceptions

Enables the throwing of C++ exceptions for all methods in this interface.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, enableExceptions, int)
   METHOD_PARM1(int enable)
END_METHOD
```

C++

```
int METHOD_CALL enableExceptions (
   int   enable);
```

Parameters

enable

(IN) Specifies whether C++ exceptions are enabled: 0= disabled, 1=enabled.

Remarks

This method is a C++ extension.

parse

Parses an XML document.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, parse, SAXException *)
   METHOD_PARM1(InputSource * inputSource)
END_METHOD
```

C++

```
SAXException * METHOD_CALL parse (
   InputSource   *inputSource);
```

Parameters

inputSource

(IN) Points to the input source for the top level of the XML document.

Remarks

The application can use this method to instruct the SAX parser to begin parsing an XML document from any valid input source (a character stream, a byte stream, or a URI).

Applications may not invoke this method while a parse is in progress. They should create a new Parser instead for each additional XML document. Once a parse is complete, an application may reuse the same Parser object, possibly with a different input source.

Objects other than “parsers” may implement this interface. For example, to convert a DOM tree to SAX events a converter might implement this interface and ignore the InputSource argument.

setDocumentHandler

Allows an application to register a document event handler.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, setDocumentHandler, void)
   METHOD_PARM1(DocumentHandler * handler)
END_METHOD
```

C++

```
void METHOD_CALL setDocumentHandler (
   DocumentHandler   *handler);
```

Parameters

handler

(IN) Points to the application's document handler.

Remarks

If the application does not register a document handler, all document events reported by the SAX parser will be silently ignored. This is the default behavior implemented by HandlerBase.

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

setDTDHandler

Allows an application to register a DTD event handler.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, setDTDHandler, void)
   METHOD_PARM1(DTDHandler * handler)
END_METHOD
```

C++

```
void METHOD_CALL setDTDHandler (
   DTDHandler   *handler);
```

Parameters

handler

(IN) Points to the application's DTD handler.

Remarks

If the application does not register a DTD handler, all DTD events reported by the SAX parser will be silently ignored. This is the default behavior implemented by HandlerBase.

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

setEntityResolver

Allows an application to register a custom entity resolver.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, setEntityResolver, void)
   METHOD_PARM1(EntityResolver * resolver)
END_METHOD
```

C++

```
void METHOD_CALL setEntityResolver (
   EntityResolver   *resolver);
```

Parameters

resolver

(IN) Points to the application's entity resolver.

Remarks

If the application does not register an entity resolver, the SAX parser will resolve system identifiers and open connections to entities itself. This is the default behavior implemented in HandlerBase.

Applications may register a new or different entity resolver in the middle of a parse, and the SAX parser must begin using the new resolver immediately.

setErrorHandler

Allows an application to register an error event handler.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, setErrorHandler, void)
   METHOD_PARM1(ErrorHandler * handler)
END_METHOD
```

C++

```
void METHOD_CALL setErrorHandler (
   ErrorHandler   *handler);
```

Parameters

handler

(IN) Points to the application's error event handler.

Remarks

If the application does not register an error event handler, all error events reported by the SAX parser will be silently ignored, except for fatalError, which will throw a SAXException. This is the default behavior implemented by HandlerBase.

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

setLocale

Allows an application to request a locale for errors and warnings.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (Parser, setLocale, SAXException *)
   METHOD_PARM1(const unicode * locale)
END_METHOD
```

C++

```
SAXException * METHOD_CALL setLocale (
   const unicode   *locale);
```

Parameters

locale

Points to a standard locale string.

Remarks

SAX parsers are not required to provide localization for errors and warnings; if they cannot support the requested locale, however, they must throw a SAX exception. Applications may not request a locale change in the middle of a parse.

3.9 SAXException

The SAXException interface encapsulates a general SAX error or warning.

This interface can contain basic error or warning information from either the XML parser or the application. A parser writer or application writer can subclass it to provide additional functionality. SAX handlers may throw this exception or any exception subclassed from it.

If the application needs to pass through other types of exceptions, it must wrap those exceptions in a SAXException or an exception derived from a SAXException.

If the parser or application needs to include information about a specific location in an XML document, it should use the SAXParseException subclass.

destroy

Destroys this SAXException.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (SAXException, destroy, void)
END_METHOD
```

C++

```
void METHOD_CALL destroy ();
```

getMessage

Returns the text message associated with this exception.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (SAXException, getMessage, const unicode *)
END_METHOD
```

C++

```
const unicode * METHOD_CALL getMessage ();
```

Remarks

This method returns null if no text is associated with the exception.

3.10 SAXParseException

The SAXParseException interface encapsulates an XML parse error or warning.

The methods include information for locating the error in the original XML document. Although the application will receive a SAXParseException as the argument to the handlers in the ErrorHandler interface, the application is not actually required to throw the exception; instead, it can simply read the information in it and take a different action.

getLocator

Gets the Locator set by the parser.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (SAXParseException, getLocator, Locator *)
END_METHOD
```

C++

```
Locator * METHOD_CALL getLocator ();
```

Remarks

This method returns null if the parser has not set a Locator object.

setLocator

Allows a parser to set a Locator object with information on the exception.

Syntax

Defining Macros for C++

```
#include "sax.h"

METHOD (SAXParseException, setLocator, void)
   METHOD_PARM1(Locator * locator)
END_METHOD
```

C++

```
void METHOD_CALL setLocator (
   Locator    *locator);
```

Parameters

locator

(IN) Points to the locator object.

Serialized XML Interface

4

This interface allows control over the serialization of an XML document. Of the three interfaces for handling XML documents, the serialized interface is the slowest. You should use this interface only if your application cannot use the SAX or DOM interface.

The prototypes for the following functions and methods are defined in the XMLWriter.h file:

getDisableTextEscaping

Returns whether XML text escaping is disabled for this XmlWriter.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, getDisableTextEscaping, int)
      END_METHOD
```

C++

```
int METHOD_CALL getDisableTextEscaping ();
```

Return Values

Returns zero if text escaping is enabled.

Returns non-zero if text escaping is disabled.

Remarks

Disabling text escaping means that no character references or entity references will be output for text nodes.

getDoctypePublic

Returns the public id to use for the DOCTYPE.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, getDoctypePublic, const unicode *)
      END_METHOD
```

C++

```
const unicode * METHOD_CALL getDoctypePublic ();
```

Return Values

Returns 0 if no public identifier has been set.

Returns non-zero if the DOCTYPE public identifier has been set.

getDoctypeSystem

Returns the system identifier to use for the DOCTYPE.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, getDoctypeSystem, const unicode *)
      END_METHOD
```

C++

```
const unicode * METHOD_CALL getDoctypeSystem ();
```

Return Values

Returns zero if a system identifier has not been set.

Returns non-zero if a system identifier has been set.

Remarks

If the system identifier is non-null and non-empty, then a DOCTYPE declaration will be output.

getEncoding

Returns the string with name of the character encoding to use.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, getEncoding, const unicode *)
      END_METHOD
```

C++

```
const unicode * METHOD_CALL getEncoding ();
```

Return Values

Returns 0 if the XML Writer has not set character encoding.

getEndian

Returns the type of byte-ordering.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, getEndian, int)
      END_METHOD
```

C++

```
int METHOD_CALL getEndian ();
```

Return Values

Returns non-zero if big-endian (Motorola) byte-ordering will be used with encodings for which it matters (for example, UTF-16).

Returns zero if little-endian (Intel) ordering will be used.

getIndent

Returns whether extra white space will be output for readability.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, getIndent, int)
      END_METHOD
```

C++

```
int METHOD_CALL getIndent ();
```

Return Values

Returns zero if extra white space will not be output.

Returns non-zero if extra white space will be output.

getStandalone

Returns whether a “standalone=yes” declaration will be output in the XML declaration.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, getStandalone, int)
      END_METHOD
```

C++

```
int METHOD_CALL getStandalone ();
```

Remarks

XML declaration must be enable for output before a standalone declaration can be output.

getWriteDeclaration

Returns whether the XML Writer will output an XML declaration.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, getWriteDeclaration, int)
      END_METHOD
```

C++

```
int METHOD_CALL getWriteDeclaration ();
```

Return Values

Returns zero if the writer will not output an XML declaration.

Returns non-zero if the writer will output an XML declaration.

setDisableTextEscaping

Sets whether XML text escaping should be disabled for this XmlWriter.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, setDisableTextEscaping, void)
            METHOD_PARM1(int disableTextEscaping)
      END_METHOD
```

C++

```
void METHOD_CALL setDisableTextEscapting (
   int   disableTextExcaping);
```

Parameters

disableTextEscaping

(IN) Specifies whether to disable XML text escaping: zero = enable and non-zero = disable.

Remarks

Disabling text escaping means that no character references or entity references will be output for text nodes.

setDoctypePublic

Sets the public identifier to use for the DOCTYPE.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, setDoctypePublic, void)
               METHOD_PARM1(const unicode * doctypePublic)
      END_METHOD
```

C++

```
void METHOD_CALL setDoctypePublic (
   const unicode   *doctypePublic);
```

Parameters

doctypePublic

(IN) Points to the public identifier for the DOCTYPE. If one has not been defined, the public identifier can be set to 0.

Remarks

The public identifier will only be used if there is also a system identifier.

setDoctypeSystem

Sets the system identifier to use for the DOCTYPE.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, setDoctypeSystem, void)
               METHOD_PARM1(const unicode * doctypeSystem)
      END_METHOD
```

C++

```
void METHOD_CALL setDoctypeSystem (
   const unicode   *doctypeSystem);
```

Parameters

doctypeSystem

(IN) Points to the system identifier for the DOCTYPE. If one has not been defined, the system identifier can be set to 0.

Remarks

If the system identifier is non-null and non-empty, then a DOCTYPE declaration will be output.

setEncoding

Sets the name of the character encoding to use.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, setEncoding, void)
               METHOD_PARM1(const unicode * encoding)
      END_METHOD
```

C++

```
void METHOD_CALL setEncoding (
   const unicode   *encoding);
```

Parameters

encoding

(IN) Points to the encoding string to use.

Remarks

The XML Writer is not required to set an encoding string.

setEndian

Sets the type of byte-ordering

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, setEndian, void)
            METHOD_PARM1(int endian)
      END_METHOD
```

C++

```
void METHOD_CALL setEndian (
   int   endian);
```

Parameters

endian

(IN) Specifies the byte-order: zero = little-endian and non-zero = big-endian.

setIndent

Sets whether extra white space may be output for readability.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, setIndent, void)
               METHOD_PARM1(int indent)
      END_METHOD
```

C++

```
void METHOD_CALL setIndent (
   int indent);
```

Parameters

indent

(IN) Specifies whether extra white space is output: zero = disable extra and non-zero = enable extra.

setStandalone

Sets whether a “standalone=yes” declaration should be output in any XML declaration that is output.

Syntax

Defining Macros for C++

```
      #include "XMLWriter.h"

METHOD (XmlWriter, setStandalone, void)
            METHOD_PARM1(int standalone)
      END_METHOD
```

C++

```
void METHOD_CALL setStandalone (
   int   standalone);
```

Parameters

standalone

(IN) Specifies whether a standalone declaration should be output: zero = disable output and non-zero = enable output.

Remarks

XML declaration must be enable for output before a standalone declaration can be output.

setWriteDeclaration

Sets whether the XML Writer should output an XML declaration.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

      METHOD (XmlWriter, setWriteDeclaration, void)
            METHOD_PARM1(int writeDeclaration)
      END_METHOD
```

C++

```
void METHOD_CALL setWriteDeclaration (
   int   writeDeclaration);
```

Parameters

writeDeclaration

(IN) Specifies whether the XML Writer outputs an XML declaration: zero = disable output and non-zero = enable output.

write

Serializes the XML data associated with the writer.

Syntax

Defining Macros for C++

```
#include "XMLWriter.h"

METHOD (XmlWriter, write, int)
      END_METHOD
```

C++

```
int METHOD_CALL write ();
```

Return Values

Zero indicates failure.

Non-zero indicates success.

UTF Converter Interface

5

The following interfaces allows an application to convert strings between UTF 8 and UTF 16:

- "UTFConverter_8to16" on page 188
- "UTFConverter_16to8" on page 189
- "UTFConverter_free" on page 190

UTFConverter_8to16

Converts a NULL-terminated UTF-8 string to a NULL-terminated UTF-16 string

Syntax

C++

```
#include "UTFConverter.h"

DIRXML_EXPORT
unicode * IFAPI UTFConverter_8to16 (
   const char   *utf8);
```

Parameters

utf8

(IN) Points to the NULL-terminated UTF-8 string to convert.

Return Values

If successful, returns a pointer to the converted string.

If an error occurs, returns NULL.

Remarks

The returned string must be freed with a call to the UTFConverter_free method.

UTFConverter_16to8

Converts a NULL-terminated UTF-16 string to a NULL-terminated UTF-8 string.

Syntax

C++

```
#include "UTFConverter.h"

DIRXML_EXPORT
char * IFAPI                        UTFConverter_16to8 (
   const unicode   *utf16);
```

Parameters

utf16

(IN) Points to the NULL-terminated UTF-16 to convert.

Return Values

If successful, returns a pointer to the converted string.

If an error occurs, returns NULL.

Remarks

The returned string must be freed with a call to the UTFConverter_free method.

UTFConverter_free

Frees a string returned from the UTFConverter_16to8 or the UTFConverter_8to16 method.

Syntax

C++

```
#include "UTFConverter.h"

DIRXML_EXPORT
void IFAPI UTFConverter_free(
   void   *encodedData);
```

Parameters

encodedData

(IN) Points to the NULL-terminated string returned by the UTFConverter_16to8 or the UTFConverter_8to16 method.

Base64 Encoding Interface

6

The Base64 Encoding interface contains the following collection of functions to encode and decode to and from base64 encoding:

- "Base64Codec_decode" on page 192
- "Base64Codec_decodeFree" on page 193
- "Base64Codec_encode" on page 194
- "Base64Codec_encodeFree" on page 195

Base64Codec_decode

Decodes binary data from a UTF-16 string containing the data encoded in base64.

Syntax

C++

```
#include "Base64Codec.h"

DIRXML_EXPORT
unsigned char * IFAPI Base64Codec_decode(
   const unicode   *encodedData,
   int             *decodedLength);
```

Parameters

encodedData

(IN) Points to a UTF-16 string containing base64-encoded data.

decodedLength

(OUT) Points to a variable that receives the length of the decoded data.

Return Values

If successful, returns a pointer to an array of bytes containing decoded data.

If the input data is malfromed, returns zero.

Remarks

The returned array must be freed with a call to the base64DecodeFree function.

Base64Codec_decodeFree

Frees a string returned from the base64Decode function.

Syntax

C++

```
#include "Base64Codec.h"

DIRXML_EXPORT
void IFAPI Base64Codec_decodeFree(
   unsigned char   *decodedData);
```

Parameters

decodeData

(IN) Points to an array which was returned from the base64Decode function.

Base64Codec_encode

Encodes an array of bytes as a NULL-terminated string of UTF-16 characters using base64 encoding.

Syntax

C++

```
#include "Base64Codec.h"

DIRXML_EXPORT
unicode * IFAPI Base64Codec_encode (
   unsigned char   *data,
   int              off,
   int              len);
```

Parameters

data

(IN) Points to the array of bytes to encode.

off

(IN) Specifies the starting offset in the array.

len

(IN) Specifies the number of bytes to encode.

Return Values

If successful, returns a pointer to a UTF-16 string.

Remarks

The returned string must be freed with a call to the base64EncodeFree method.

Base64Codec_encodeFree

Frees a string returned from the base64Encode function.

Syntax

C++

```
#include "Base64Codec.h"

DIRXML_EXPORT
void IFAPI Base64Codec_encodeFree(
   unicode   *encodedData);
```

Parameters

encodedData

(IN) Points to the UTF-16 string returned from the base64Encode function.

Driver Filter Interface

7

The following DriverFilter and ClassFilter interfaces allow an application to obtain information about the Subscriber or Publisher filter. The application can then use that information to filter the data before sending information to DirXML.

- "getClassFilter" on page 198
- "passAttribute" on page 199
- "passClass" on page 200

See also the driver filter methods in InterfaceFactory.h (see Chapter 8, "Factory Interface," on page 201).

getClassFilter

Returns the class filter for the specified class.

Syntax

C++

```
#include "DriverFilter.h"

ClassFilter * DriverFilter::getClassFilter (
   const unicode   *className);
```

Parameters

className

(IN) Points to the name of the class.

Return Values

Returns zero if the specified class is not in the filter.

passAttribute

Returns whether the specified attribute is in the filter.

Syntax

C++

```
#include "DriverFilter.h"

int ClassFilter::passAttribute(
   const unicode   *attrName);
```

Parameters

attrName

(IN) Points to the name of the attribute to test.

Return Values

Returns zero if the specified attribute is not in the filter.

Returns non-zero if the specified attribute is in the filter.

passClass

Returns whether the specified class is in the filter.

Syntax

C++

```
#include "DriverFilter.h"

int DriverFilter::passClass (
   const unicode   *className);
```

Parameters

className

(IN) Points to the name of the class.

Return Values

Returns zero if the specified class is not in the filter.

Returns non-zero if the specified class is in the filter.

Factory Interface

8

The factory interface creates implementations of the following XML interfaces:

- Section 8.1, “DOM,” on page 201
- Section 8.2, “SAX,” on page 201
- Section 8.3, “Serialized,” on page 201

8.1 DOM

The following functions or methods are used with DOM:

- Document_destroyInstance (page 207)
- Document_new (page 208)
- DriverFilter_new (page 210)
- DriverFilter_destroy (page 209)
- XmlDocument_newFromDOM (page 224)
- XmlDocument_destroy (page 222)

8.2 SAX

The following functions or methods are used with SAX:

- InputSource_destroy (page 214)
- InputSource_new (page 215)
- Parser_destroy (page 216)
- Parser_new (page 217)
- SAXException_new (page 218)
- SAXParseException_new (page 219)
- XmlDocument_destroy (page 222)
- XmlDocument_newFromSAX (page 225)

8.3 Serialized

The following functions or methods are used with serialized:

- ByteArrayOutputStream_destroy (page 203)
- ByteArrayOutputStream_getBytes() (page 204)
- ByteArrayOutputStream_getDataSize() (page 205)
- ByteArrayOutputStream_new (page 206)
- FileOutputStream_destroy (page 211)
- FileOutputStream_newFromFILE (page 212)

- FileOutputStream_newFromName (page 213)
- XmlDocument_destroy (page 222)
- XmlDocument_newFromBytes (page 223)

ByteArrayOutputStream_destroy

Destroys a ByteArrayOutputStream object returned from FileOutputStream_newFromName or FileOutputStream_newFromFILE method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI ByteArrayOutputStream_destroy (
   OutputStream   *outputStream);
```

Parameters

outputStream

(IN) Points to the output stream to destroy.

ByteArrayOutputStream_getBytes()

Returns the data in the ByteArrayOutputStream.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
int IFAPI ByteArrayOutputStream_getBytes (
   OutputStream    *outputStream,
   unsigned char   *buffer);
```

Parameters

outputStream

(IN) Points to the pointer returned by the ByteArrayOutputStream_new method.

buffer

(OUT) Points to the buffer for the data.

Return Values

Returns the amount of data returned in the passed buffer.

Remarks

Notes:The passed buffer must be at least as large as the value returned from ByteArrayOutputStream_getDataSize().

ByteArrayOutputStream_getDataSize()

Returns the size of the data in the ByteArrayOutputStream

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
int IFAPI ByteArrayOutputStream_getDataSize (
   OutputStream   *outputStream);
```

Parameters

outputStream

(IN) Points to the pointer returned by the ByteArrayOutputStream_new method.

Remarks

See also the ByteArrayOutputStream_getBytes method.

ByteArrayOutputStream_new

Creates a ByteArrayOutputStreamObject object with the passed initial buffer size and with the passed growIncrement.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
OutputStream * IFAPI ByteArrayOutputStream_new (
   int    initialSize,
   int    growIncrement);
```

Parameters

initialSize

(IN) Specifies the initial buffer size. To use the default value, set to zero.

growIncrement

(IN) Specifies the amount to grow the buffer when neede. To use the default value, set to zero.

Remarks

What are the defaults?

Document_destroyInstance

Destroys an interface returned from the Document_new method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI Document_destroyInstance (
   NAMESPACE(DOM)Document    *document);
```

Parameters

document

(IN) Points to the document interface to destroy.

Document_new

Returns a new DOM Document instance

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
NAMESPACE(DOM)Document * IFAPI Document_new ();
```

Return Values

Returns a pointer to a document interface.

Remarks

The returned document must be freed by calling the destroy method or the Document_destroyInstance method when finished.

DriverFilter_destroy

Destroys a driver filter instance returned from the DriverFilter_new method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI DriverFilter_destroy (
   DriverFilter   *driverFilter);
```

Parameters

driverFilter

(IN) Points to the driver filter instance to destroy.

DriverFilter_new

Creates a new DriverFilter instance based on the DOM representation of an XDS driver filter element.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
DriverFilter * IFAPI DriverFilter_new (
   NAMESPACE(DOM)Element    *driverFilterElement);
```

Parameters

driverFilterElement

(IN) Points to an XDS driver filter element.

Return Values

If successful, returns a pointer to a driver filter interface.

If the driver filter element is invalid, returns zero.

Remarks

See also Chapter 7, “Driver Filter Interface,” on page 197.

FileOutputStream_destroy

Destroys a FileOutputStream object returned from the FileOutputStream_newFromFILE or FileOutputStream_newFromName method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI FileOutputStream_destroy (
   OutputStream   *outputStream);
```

Parameters

outputStream

(IN) Points to the out put stream to destroy.

FileOutputStream_newFromFILE

Creates a FileOutputStream object from a FILE pointer.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
OutputStream * IFAPI FileOutputStream_newFromFILE (
   FILE   *file);
```

Parameters

file

(IN) Points to a FILE structure to create into an output stream object.

Remarks

This method should be used with the XmlDocument::writeDocument or the XmlDocument::getXmlWriter method from NativeInterface.h.

The passed FILE must be opened for writing.

FileOutputStream_newFromName

Creates a FileOutputStream object from a file name.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
OutputStream * IFAPI FileOutputStream_newFromName (
   const char   *filename);
```

Parameters

filename

(IN) Points to the file to create into an output stream object.

Remarks

This method should be used with XmlDocument::writeDocument or XmlDocument::getXmlWriter method from NativeIngerface.h.

The passed file name will be opened using fopen(), mode "wb+".

InputSource_destroy

Destroys a SAX input source instance returned from the InputSource_new method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI InputSource_destroy (
   NAMESPACE(SAX)InputSource   *inputSource);
```

Parameters

inputSource

(IN) Points to the input source to destroy.

InputSource_new

Returns a new SAX input source instance for use with an instance of the SAX parser.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
NAMESPACE(SAX)InputSource * IFAPI InputSource_new ();
```

Remarks

The returned interface must be freed by calling the InputSource_destroy method when finished.

Parser_destroy

Destroys a SAX Parser instance returned from the Parser_new method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI Parser_destroy (
   NAMESPACE(SAX)Parser   *parser);
```

Parameters

parser

(IN) Points to the parser interface to destroy.

Parser_new

Returns a new SAX parser instance.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
NAMESPACE(SAX)Parser * IFAPI Parser_new ();
```

Return Values

Returns a pointer to a SAX parser interface.

Remarks

The returned interface must be freed by calling the Parser_destroy method when finished.

SAXException_new

Creates a new SAXException instance.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
SAXException * IFAPI SAXException_new (
   const unicode    *message);
```

Parameters

message

(IN) Points to the unicode message for the SAX exception.

Return Values

Returns a pointer to a SAXException interface.

Remarks

This method is provided as a convenience for SAX Parser, SAX DocumentHandler, and DTDHandler implementations. For more information, see Chapter 3, “Simple API for XML (SAX) Interfaces,” on page 111.

SAXParseException_new

Creates a new SAXExceptionParse instance.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
SAXParseException * IFAPI SAXParseException_new (
   const unicode   *message);
```

Parameters

message

(IN) Points to the unicode message for the exception.

Return Values

Returns a pointer to a SAXParseException interface.

Remarks

This method is provided as a convenience for SAX Parser implemententations (see Section 3.10, “SAXParseException,” on page 165).

Trace_new

Creates a new trace instance with the passed identifying label.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
Trace * IFAPI Trace_new (
   const char   *identifier);
```

Parameters

identifier

(IN) Points to a string that will appear at start of each trace message.

Return Values

Returns a pointer to the trace instance.

Remarks

This method .

Trace_destroy

Destroys the trace instance created with the Trace_new method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void * IFAPI Trace_destroy (
   Trace   *trace);
```

Parameters

trace

(IN) Points to the trace instance to destroy.

Return Values

Returns a pointer to the trace instance.

XmlDocument_destroy

Destroys an XML document instance returned from the XmlDocument_newFromSAX or XmlDocument_newFromDOM method.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
void IFAPI XmlDocument_destroy (
   XmlDocument   *xmlDocument);
```

Parameters

xmlDocument

(IN) Points to the interface to destroy.

XmlDocument_newFromBytes

Creates a new XML document instance from an array of bytes

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
XmlDocument * IFAPI XmlDocument_newFromBytes (
   const unsigned char   *bytes,
   int                   length,
   const unicode         *encoding,
   int                    endian);
```

Parameters

bytes

(IN) Points to an array of bytes with a serialized XML document.

length

(IN) Specifies the length of the byte array.

encoding

(IN) Points to a NULL-terminated unicode string that specifies the character encoding of the byte array.

endian

(IN) Specifies the byte-order of the encoding in the array: zero for Intel and non-zero for Motorola.

Remarks

The returned instance must be destroyed by calling the XmlDocument_destroy method.

The XmlDocument instance does not take ownership of the passed byte array and so it must be freed after the XmlDocument instance is destroyed.

XmlDocument_newFromDOM

Creates a new XML document instance from a DOM tree.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
XmlDocument * IFAPI XmlDocument_newFromDOM (
   NAMESPACE(DOM)Document   *document);
```

Parameters

document

(IN) Points to a DOM document interface.

Return Values

Returns a pointer to an XML document interface.

Remarks

The getDocument method in NativeInterface.h returns an instace of a DOM document interface.

The returned instance must be destroyed with a call to XmlDocument_destroy()

The XmlDocument instance does NOT take ownership of the passed DOM Document instance and so it must be freed AFTER the XmlDocument instance is destroyed.

XmlDocument_newFromSAX

Creates a new XML document instance from a SAX Parser and a SAX InputSource.

Syntax

C++

```
#include "InterfaceFactory.h"

DIRXML_EXPORT
XmlDocument * IFAPI XmlDocument_newFromSAX (
   NAMESPACE(SAX)Parser         *parser,
   NAMESPACE(SAX)InputSource    *inputSource);
```

Parameters

parser

(IN) Points to a SAX parser interface

inputSource

(IN) Points to a SAX input source interface.

Return Values

Returns a pointer to an XML document interface.

Remarks

In NativeInterface.h, the getDocumentSAX method returns a SAX parser and the getDocumentInputSource method returns an input source.

The returned instance must be destroyed by calling the XmlDocument_destroy method.

The XML document instance does not take ownership of the passed SAX parser instance or the passed SAX input source instance and so they must be freed after the XML document instance is destroyed.

OutputStream.h

9

The following methods provide an interface for writing bytes to a byte sink:

- “close” on page 228
- “flush” on page 229
- “write” on page 230
- “write” on page 231

close

Flushes any unwritten bytes and closes the stream.

Syntax

C++

```
#include "OutputStream.h"

void OutputStream::close ();
```

flush

Flushes any unwritten bytes to the stream.

Syntax

C++

```
#include "OutputStream.h"

void OutputStream::flush ();
```

write

Writes bytes to the stream.

Syntax

C++

```
#include "OutputStream.h"

int OutputStream::write (
   const unsigned char   *bytes,
   int                    length);
```

Parameters

bytes

(IN) Points to the array of bytes to write.

length

(IN) Specifies the length of the byte array.

Return Values

Returns the number of bytes written.

write

Writes a single byte to the stream.

Syntax

C++

```
#include "OutputStream.h"

int OutputStream::write (
   unsigned char   byte);
```

Parameters

byte

(IN) Specifies the byte to write.

Return Values

Returns the number of bytes written, either 0 or 1.

Trace Interface

10

The following Trace interface allows a DirXML driver to writing messages to DSTrace. See also the Trace_new and Trace_destroy methods (in Chapter 8, “Factory Interface,” on page 201), which create and destroy the trace object.

For DirXML trace messages to display, the DSTrace facility must be enabled for DirXML messages. Bring up the DSTrace screen on the NDS server, selcet "DirXML" and "DirXML Driver" messages, and restart NDS.

getTraceLevel

Returns the current trace level.

Syntax

C++

```
#include "Trace.h"

int Trace::getTraceLevel(
   void);
```

Remarks

This method returns one of the following levels:

Level	Description
NO_TRACE=0	Display no trace messages
DEFAULT_TRACE=1	Display trace messages
XML_TRACE=2	Display XML documents

pushIndent

Configures trace to indent the messages following the identifier string by the specified number of tabs.

Syntax

C++

```
#include "Trace.h"

void Trace::pushIndent(
   int    tabCount);
```

Parameters

tabCount

(IN) Specifies the number of tabs to indent.

popIndent

Restores the indent state after a pushIndent call.

Syntax

C++

```
#include "Trace.h"

void Trace::popIndent(
   void);
```

resetIndent

Resets the indent state to 0 regardless of any pushed levels.

Syntax

C++

```
#include "Trace.h"

void Trace::resetIndent(
   void);
```

trace

Displays a unicode message if the current trace level is greater than or equal to the DEFAULT_TRACE level.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   const unicode    *message);
```

Parameters

message

(IN) Points to the message to display.

trace

Displays a unicode message if the current trace level is greater than the specified level.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   const unicode    *message,
   int               level);
```

Parameters

message

(IN) Points to the message to display.

level

(IN) Specifies the trace level.

Remarks

The DirXML trace facility supports the following levels.

Level	Description
NO_TRACE=0	Display no trace messages
DEFAULT_TRACE=1	Display trace messages
XML_TRACE=2	Display XML documents

trace

Displays a character string message if the current trace level is greater than or equal to the DEFAULT_TRACE level.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   const char   *message);
```

Parameters

message

(IN) Points to the message to display.

trace

Displays a character string message if the current trace level is greater than the specified level.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   const char   *message,
   int           level);
```

Parameters

message

(IN) Points to the message to display.

level

(IN) Specifies the trace level.

Remarks

The trace facility supports the following levels.

Level	Description
NO_TRACE=0	Display no trace messages
DEFAULT_TRACE=1	Display trace messages
XML_TRACE=2	Display XML documents

trace

Displays an XML document of type Document if the current trace level is XML_TRACE or greater.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   Document    *document);
```

Parameters

document

(IN) Points to the XML document to display.

trace

Displays an XML document of type Document if the current trace level is greater than or equal to the specified level.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   Document    *document
   int          level);
```

Parameters

document

(IN) Points to the XML document to display.

int

(IN) Specifies the trace level.

Remarks

The trace facility supports the following levels.

Level	Description
NO_TRACE=0	Display no trace messages
DEFAULT_TRACE=1	Display trace messages
XML_TRACE=2	Display XML documents

trace

Displays an XML document of type XmlDocument if the current trace level is XML_TRACE or greater.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   XmlDocument    *document);
```

Parameters

document

(IN) Points to the XML document to display.

trace

Displays an XML document of type XmlDocument if the current trace level is greater than or equal to the specified level.

Syntax

C++

```
#include "Trace.h"

void Trace::trace(
   XmlDocument    *document
   int             level);
```

Parameters

document

(IN) Points to the XML document to display.

int

(IN) Specifies the trace level.

Remarks

What levels are supported?

NdsDtd Interface

11

The following NdsDtd interfaces contains helper functions for creating input and output documents:

NdsDtd_getStrings

Returns a read-only pointer to an NdsDtd structure.

Syntax

C++

```
#include "NdsDtd.h"

DIRXML_EXPORT
const NdsDtd * NDAPI  NdsDtd_getStrings();
```

Remarks

The NdsDtd structure has the following format.

```
struct NdsDtd
{
//element tags
   const unicode   * TAG_NDS;                    //<nds>
   const unicode   * TAG_SOURCE;                 //<source>
   const unicode   * TAG_INPUT;                  //<input>
   const unicode   * TAG_OUTPUT;                 //<output>
   const unicode   * TAG_PRODUCT;                //<product>
   const unicode   * TAG_CONTACT;                //<contact>
   const unicode   * TAG_ADD;                    //<add>
   const unicode   * TAG_MODIFY;                 //<modify>
   const unicode   * TAG_DELETE;                 //<delete>
   const unicode   * TAG_RENAME;                 //<rename>
   const unicode   * TAG_MOVE;                   //<move>
   const unicode   * TAG_QUERY;                  //<query>
   const unicode   * TAG_QUERY_SCHEMA;           //<query-schema>
   const unicode   * TAG_ADD_ASSOCIATION;        //<add-association>
   const unicode   * TAG_MODIFY_ASSOCIATION;     //<modify-association>
   const unicode   * TAG_REMOVE_ASSOCIATION;     //<remove-association>
   const unicode   * TAG_INIT_PARAMS;            //<init-params>
   const unicode   * TAG_STATUS;                 //<status>
   const unicode   * TAG_CHECK_PASSWORD;         //<check-password>
   const unicode   * TAG_INSTANCE;               //<instance>
   const unicode   * TAG_SCHEMA_DEF;             //<schema-def>
   const unicode   * TAG_VALUE;                  //<value>
   const unicode   * TAG_COMPONENT;              //<component>
   const unicode   * TAG_ASSOCIATION;            //<association>
   const unicode   * TAG_PARENT;                 //<parent>
   const unicode   * TAG_SEARCH_CLASS;           //<search-class>
   const unicode   * TAG_SEARCH_ATTR;            //<search-attr>
   const unicode   * TAG_READ_ATTR;              //<read-attr>
   const unicode   * TAG_READ_PARENT;            //<read-parent>
   const unicode   * TAG_ADD_ATTR;               //<add-attr>
   const unicode   * TAG_PASSWORD;               //<password>
   const unicode   * TAG_MODIFY_ATTR;            //<modify-attr>
   const unicode   * TAG_REMOVE_VALUE;          //<remove-value>
   const unicode   * TAG_REMOVE_ALL_VALUES;     //<remove-all-values>
   const unicode   * TAG_ADD_VALUE;             //<add-value>
   const unicode   * TAG_NEW_NAME;              //<new-name>
   const unicode   * TAG_ATTR;                  //<attr>
   const unicode   * TAG_AUTHENTICATION_INFO;   //<authentication-info>
   const unicode   * TAG_DRIVER_FILTER;         //<driver-filter>
   const unicode   * TAG_DRIVER_OPTIONS;        //<driver-options>
   const unicode   * TAG_SUBSCRIBER_OPTIONS;    //<subscriber-options>
   const unicode   * TAG_PUBLISHER_OPTIONS;     //<publisher-options>
   const unicode   * TAG_DRIVER_STATE;          //<driver-state>
   const unicode   * TAG_SUBSCRIBER_STATE;      //<subscriber-state>
   const unicode   * TAG_PUBLISHER_STATE;       //<publisher-state>
   const unicode   * TAG_SERVER;                //<server>
   const unicode   * TAG_USER;                  //<user>
   const unicode   * TAG_ALLOW_CLASS;           //<allow-class>
   const unicode   * TAG_ALLOW_ATTR;            //<allow-attr>
   const unicode   * TAG_DRIVER_CONFIG;         //<driver-config>
   const unicode   * TAG_CONFIG_OBJECT;         //<config-object>
   const unicode   * TAG_CLASS_DEF;             //<class-def>
   const unicode   * TAG_ATTR_DEF;              //<attr-def>

//attribute names
   const unicode   * ATTR_NDSVERSION;           //ndsversion
   const unicode   * ATTR_DTDVERSION;           //dtdversion
   const unicode   * ATTR_VERSION;              //version
   const unicode   * ATTR_ASN1ID;               //asn1id
   const unicode   * ATTR_TYPE;                 //type
   const unicode   * ATTR_ASSOCIATION_REF;      //association-ref
   const unicode   * ATTR_NAMING;               //naming
   const unicode   * ATTR_TIMESTAMP;            //timestamp
   const unicode   * ATTR_NAME;                 //name
   const unicode   * ATTR_STATE;                //state
   const unicode   * ATTR_SRC_DN;               //src-dn
   const unicode   * ATTR_SRC_ENTRY_ID;         //src-entry-id
   const unicode   * ATTR_DEST_DN;              //dest-dn
   const unicode   * ATTR_DEST_ENTRY_ID;        //dest-entry-id
   const unicode   * ATTR_CLASS_NAME;           //class-name
   const unicode   * ATTR_SCOPE;                //scope
   const unicode   * ATTR_EVENT_ID;             //event-id
   const unicode   * ATTR_ATTR_NAME;            //attr-name
   const unicode   * ATTR_TEMPLATE_DN;          //template-dn
   const unicode   * ATTR_OLD_SRC_DN;           //src-old-src-dn
   const unicode   * ATTR_REMOVE_OLD_NAME;      //remove-old-name
   const unicode   * ATTR_LEVEL;                //level
   const unicode   * ATTR_DISPLAY_NAME;         //display-name
   const unicode   * ATTR_HIERARCHICAL;         //hierarchical
   const unicode   * ATTR_APPLICATION_NAME;     //application-name
   const unicode   * ATTR_CONTAINER;            //container
   const unicode   * ATTR_REQUIRED;             //required
   const unicode   * ATTR_MULTI_VALUED;         //multi-valued
   const unicode   * ATTR_CASE_SENSITIVE;       //case-sensitive
   const unicode   * ATTR_READ_ONLY;            //read-only
```

```
//attribute values
   const unicode   * VAL_NOT_ASSOCIATED;       //"not-associated"
   const unicode   * VAL_ASSOCIATED;           //"associated"
   const unicode   * VAL_DISABLED;             //"disabled"
   const unicode   * VAL_MIGRATE;              //"migrate"
   const unicode   * VAL_PENDING;              //"pending"
   const unicode   * VAL_MANUAL;               //"manual"
   const unicode   * VAL_ENTRY;                //"entry"
   const unicode   * VAL_SUBORDINATES;         //"subordinates"
   const unicode   * VAL_SUBTREE;              //"subtree"
   const unicode   * VAL_FATAL;                //"fatal"
   const unicode   * VAL_ERROR;                //"error"
   const unicode   * VAL_WARNING;              //"warning"
   const unicode   * VAL_SUCCESS;              //"success"
   const unicode   * VAL_RETRY;                //"retry"
   const unicode   * VAL_STRING;               //"string"
   const unicode   * VAL_TELENUMBER;           //"teleNumber"
   const unicode   * VAL_INT;                  //"int"
   const unicode   * VAL_STATE;                //"state"
   const unicode   * VAL_COUNTER;              //"counter"
   const unicode   * VAL_DN;                   //"dn"
   const unicode   * VAL_INTERVAL;             //"interval"
   const unicode   * VAL_OCTET;                //"octet"
   const unicode   * VAL_TIME;                 //"time"
   const unicode   * VAL_STRUCTURED;           //"structured"
   const unicode   * VAL_DEFAULT;              //"default"
   const unicode   * VAL_XML;                  //"xml"
   const unicode   * VAL_TRUE;                 //"true"
   const unicode   * VAL_FALSE;                //"false"
   const unicode   * VAL_DOT;                  //"dot"
   const unicode   * VAL_QUALIFIED_DOT;        //"qualified-dot"
   const unicode   * VAL_SLASH;                //"slash"
   const unicode   * VAL_QUALIFIED_SLASH;      //"qualified-slash"
   const unicode   * VAL_LDAP;                 //"ldap"
   const unicode   * VAL_SUBSCRIBER;           //"subscriber"
   const unicode   * VAL_PUBLISHER;            //"publisher"
   const unicode   * VAL_CURRENT_NDS_VERSION;  //"8.5"
   const unicode   * VAL_CURRENT_DTD_VERSION;  //"1.0"
};

typedef struct NdsDtd NdsDtd;
```

NdsDtd_newInputDocument

Creates a new <input> document and returns a pointer to the <input> element.

Syntax

C++

```
#include "NdsDtd.h"

DIRXML_EXPORT
NAMESPACE(DOM)Element * NDAPI NdsDtd_newInputDocument(
   void);
```

Remarks

The method returns a pointer to the &lt;input> element in the new document or 0 if error occurs creating the document.

The created document has the following format:

```
<nds ndsversion="8.5" dtdversion="1.0"><input/></nds>
```

The caller is responsible for destroying the returned document. The document can be obtained by calling the getOwnerDocument method on the returned element.

NdsDtd_newOutputDocument

Creates a new <output> document returns a pointer to the <output> element.

Syntax

C++

```
#include "NdsDtd.h"

DIRXML_EXPORT
NAMESPACE(DOM)Element * NDAPI NdsDtd_newOutputDocument(
   void);
```

Remarks

This method returns a pointer to the &lt;output> element in the new document or 0 if an occurs error creating the document.

The created document has the following format:

```
<nds ndsversion="8.5" dtdversion="1.0"><output/></nds>
```

The caller is responsible for destroying the returned document. The document can be obtained by calling the getOwnerDocument method on the returned element.

NdsDtd_addStatus

Adds a <status> element to the input or output document.

Syntax

C++

```
#include "NdsDtd.h"

DIRXML_EXPORT
NAMESPACE(DOM)Element * NDAPI NdsDtd_addStatus(
   NAMESPACE(DOM)Element   *statusParent,
   int                      statusLevel,
   const unicode           *message,
   const unicode           *eventId);
```

Parameters

statusParent

(IN) Points to the <input> or <output> element to which the <status> element is to be added

statusLevel

(IN) Specifies the status level for the <status> element. The following values are supported:

STATUS_LEVEL_FATAL (0)
STATUS_LEVEL_ERROR (1)
STATUS_LEVEL_WARNING (2)
STATUS_LEVEL_SUCCESS (3)
STATUS_LEVEL_RETRY (4)

message

(IN) Points to null-terminated message string. It may be set to 0.

eventID

(IN) Points to a null-terminated event ID string. It may be set to 0. For an output document, it should be set to the event ID string sent with the requesting input document.

Remarks

This method returns a pointer to the new <status> element or 0 if an error occurs.

The status element will have the level attribute set based on the statusLevel parameter. It may have an eventId attribute or a text node child with any message passed.

Revision History

A

The following table lists all changes made to the XML Interfaces for C++ documentation:

March 1, 2006	Made minor technical edits.
October 5, 2005	Transitioned to revised Novell documentation standards.
September 2000	Added information about the XmlDocument, Trace, and NdsDtd interfaces.
May 2000	Published on the NDK as a Leading Edge component.