Oracle Berkeley DB XML

API Reference for C++

12c Release 1

Library Version 12.1.6.0

ORACLE

BERKELEY DB

Legal Notice

Published 7/10/2015

Table of Contents

Preface

Welcome to Berkeley DB XML 12*c* Release 1 (BDB XML). This document describes the C++ API for BDB XML library version 12.1.6.0. It is intended to describe the BDB XML API, including all classes, methods, and functions. As such, this document is intended for developers who are actively writing or maintaining applications that make use of BDB XML databases.

Conventions Used in this Book

The following typographical conventions are used within in this manual:

Variable or non-literal text is presented in *italics*. For example: "Go to your *DB_INSTALL* directory."

Program examples are displayed in a `monospaced font` on a shaded background. For example:

Note

Finally, notes of interest are represented using a note block such as this.

For More Information

Beyond this manual, you may also find the following sources of information useful when building a BDB XML application:

•

•

•

•

•

- Berkeley DB TCL API Reference Guide
- Berkeley DB Installation and Build Guide
- Berkeley DB Programmer's Reference Guide
- Berkeley DB Getting Started with the SQL APIs

To download the latest Berkeley DB XML documentation along with white papers and other collateral, visit http://www.oracle.com/technetwork/indexes/documentation/index.html.

For the latest version of the Oracle Berkeley DB XML downloads, visit http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/downloads/index.html.

Contact Us

You can post your comments and questions at the Oracle Technology (OTN) forum for Oracle Berkeley DB XML at: https://community.oracle.com/community/database/high_availability/berkeley_db_family/berkeley_db_xml.

For sales or support information, email to: berkeleydb-info_us@oracle.com You can subscribe to a low-volume email announcement list for the Berkeley DB product family by sending email to: bdb-join@oss.oracle.com

Chapter 1. Introduction to Berkeley DB XML APIs

Welcome to the Berkeley DB XML API Reference Manual for C++.

Welcome to Berkeley DB XML (BDB XML). BDB XML is an embedded database specifically designed for the storage and retrieval of XML-formatted documents. Built on the award-winning Berkeley DB, BDB XML provides for efficient queries against millions of XML documents using XQuery. XQuery is a query language designed for the examination and retrieval of portions of XML documents.

This manual describes the various APIs and command line utilities available for use in the BDB XML library.

For a general description of using BDB XML beyond the reference material available in this manual, see the Getting Started Guides which are identified in this manual's preface.

This manual is broken into chapters, each one of which describes a series of APIs designed to work with one particular aspect of the BDB XML library. Each such chapter is organized around a "handle", or class, which provides an interface to BDB XML structures such as containers, transactions or result sets.

Within each chapter, methods, functions and command line utilities are organized alphabetically.

Chapter 2. DbXml

```
#include <DbXml.hpp>

void DbXml::setLogLevel(LogLevel level, bool enabled)

void DbXml::setLogCategory(LogCategory category, bool enabled)

void DbXml::dbxml_version(int *majorp, int *minorp, int *patchp)
```

This chapter describes several utility functions used to interact with the DB XML library on a global level.

BDB XML Utility Functions

BDB XML Utility Functions	Description
DbXml::setLogLevel	Sets BDB XML's logging level.
DbXml::setLogCategory	Sets BDB XML's logging category.
DbXml::dbxml_version	Returns the Berkeley DB XML release number

DbXml::setLogLevel

```
#include <DbXml.hpp>

void DbXml::setLogLevel(LogLevel level, bool enabled)
```

Berkeley DB XML can be configured to generate a stream of messages to help application debugging. The messages are categorized by subsystem, and by importance. The messages are sent to the output stream that is configured in the Berkeley DB environment associated with the XmlManager (page 274) generating the message. The output is sent to `std::cerr` if no environment is associated with the XmlManager (page 274).

Parameters

level

The log level to enable or disable. Must be one of the following:

- `DbXml::LEVEL_DEBUG`

  Enable program execution tracing messages.
- `DbXml::LEVEL_INFO`

  Enable informational messages.
- `DbXml::LEVEL_WARNING`

  Enable warning messages.
- `DbXml::LEVEL_ERROR`

  Enable fatal error messages.
- `DbXml::LEVEL_ALL`

  Enable all debug levels.

enabled

A Boolean flag that specifies whether to enable or disable the level.

Class

DbXml (page 2)

See Also

BDB XML Utility Functions (page 3)

DbXml::setLogCategory

```
#include <DbXml.hpp>

DbXml::setLogCategory(LogCategory category, bool enabled)
```

Berkeley DB XML can be configured to generate a stream of messages to help application debugging. The messages are categorized by subsystem, and by importance. The messages are sent to the output stream that is configured in the Berkeley DB environment associated with the XmlManager (page 274) generating the message. The output is sent to `std::cerr` if no environment is associated with the XmlManager (page 274).

Parameters

category

The log category to enable or disable. Must be one of the following:

- `DbXml::CATEGORY_INDEXER`

  Enable indexer messages.

- `DbXml::CATEGORY_QUERY`

  Enable query processor messages.

- `DbXml::CATEGORY_OPTIMIZER`

  Enable optimizer messages.

- `DbXml::CATEGORY_DICTIONARY`

  Enable dictionary messages.

- `DbXml::CATEGORY_CONTAINER`

  Enable container messages.

- `DbXml::CATEGORY_NODESTORE`

  Enable node storage messages.

- `DbXml::CATEGORY_MANAGER`

  Enable manager messages.

- `DbXml::CATEGORY_ALL`

  Enable all messages.

enabled

A Boolean flag that specifies whether to enable or disable the category.

Class

DbXml (page 2)

See Also

BDB XML Utility Functions (page 3)

DbXml::dbxml_version

```
#include <DbXml.hpp>

DbXml::dbxml_version(int *majorp, int *minorp, int *patchp)
```

Returns the Berkeley DB XML release number.

Parameters

majorp

The release's major version number.

minorp

The release's minor version number.

patchp

The release's patch version number.

Class

DbXml (page 2)

See Also

BDB XML Utility Functions (page 3)

Chapter 3. XmlArguments

```
#include <DbXml.hpp>

class DbXml::XmlArguments {
public:
    XmlResults getArguments(size_t index) const;
    unsigned int getNumberOfArgument() const;
};
```

The `XmlArguments` class is used by implementors of XmlExternalFunction (page 217) to access function arguments passed to the XmlExternalFunction::execute() (page 219) method.

XmlArguments Methods

XmlArguments Methods	Description
XmlArguments::getArguments()	Get an Xml Argument
XmlArguments::getNumberOfArguments()	Get the number of Xml Arguments

XmlArguments::getArguments()

```
#include <DbXml.hpp>

XmlResults XmlArguments::getArguments(size_t index) const;
```

Get the argument at the specified index.

Parameters

index

The index for the desired argument, zero-based.

Class

XmlArguments (page 8)

See Also

XmlArguments Methods (page 9), XmlResults (page 380)

XmlArguments::getNumberOfArguments()

```
#include <DbXml.hpp>

unsigned in XmlArguments::getNumberOfArguments() const;
```

Get the number of arguments available in the object.

Class

XmlArguments (page 8)

See Also

XmlArguments Methods (page 9)

Chapter 4. XmlCompression

```
#include <DbXml.hpp>

XmlCompression::XmlCompression()
virtual XmlCompression::~XmlCompression()
```

`XmlCompression` is a base class for implementations of custom compression for a container. Compression is only used by whole document storage containers. To use customized compression the implementation must be registered with the `XmlManager` using XmlManager::registerCompression (page 319) and the container must be created by passing an XmlContainerConfig (page 76) object to XmlManager::createContainer (page 281) that includes the name under which the instance was registered. Compression is a persistent attribute of a container. A container created with custom compression requires that the same named compression instance be registered with the `XmlManager` or any attempt to open the container will fail.

`XmlCompression` instances must be free-threaded and safe to use concurrently.

XmlCompression Methods

XmlCompression Methods	Description
XmlCompression::compress	Compresses data before it is added to a container.
XmlCompression::decompress	Decompresses data when it is retrieved from a container.

XmlCompression::compress

```
#include <DbXml.hpp>

bool XmlCompression::compress(
    XmlTransaction &txn, const XmlData &source, XmlData &dest) = 0
```

This function is called when data is placed in a container that has compression enabled. The function compresses the data from source into dest. The method should return true if compression was successful and false if not.

Parameters

txn

XmlTransaction (page 407) The transaction used by the operation that is putting data into the container.

source

XmlData (page 118) Contains the data to be compressed.

dest

XmlData (page 118) The buffer for the compressed data returned by this method.

Class

XmlCompression (page 12)

See Also

XmlCompression Methods (page 13)

XmlCompression::decompress

```
#include <DbXml.hpp>

bool XmlCompression::decompress(
    XmlTransaction &txn, const XmlData &source, XmlData &dest) = 0
```

This function is called when data is retrieved from a container that has compression enabled. The function decompresses the data from source into dest. The method should return true if compression was successful and false if not.

Parameters

txn

XmlTransaction (page 407) The transaction used by the operation that is retrieving data from the container.

source

XmlData (page 118) Contains the data to be decompressed.

dest

XmlData (page 118) The buffer for the decompressed data returned by this method.

Class

XmlCompression (page 12)

See Also

XmlCompression Methods (page 13)

Chapter 5. XmlContainer

```
#include <DbXml.hpp>

class DbXml::XmlContainer {
public:
 XmlContainer()
 XmlContainer(const XmlContainer &o)
 XmlContainer &operator=(const XmlContainer &o)
 ~XmlContainer()
 ...
};
```

The `XmlContainer` class encapsulates a document container and its related indices and statistics. `XmlContainer` exposes methods for managing (putting and deleting) XmlDocument (page 135) objects, managing indices, and retrieving container statistics.

If the container has never before been opened, use XmlManager::createContainer (page 281) to instantiate an `XmlContainer` object. If the container already exists, use XmlManager::openContainer (page 310) instead. `XmlContainers` are always opened until the last referencing handle is destroyed.

You can delete containers using XmlManager::removeContainer (page 323) and rename containers using XmlManager::renameContainer (page 324).

A copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body. This object is free threaded, and can be safely shared among threads in an application.

XmlContainer Methods

XmlContainer Methods	Description
XmlContainer::addAlias	Adds a alias for the container name.
XmlContainer::addDefaultIndex	Add a default index to the container.
XmlContainer::addIndex	Adds an index specification.
XmlContainer::deleteDefaultIndex	Delete the container's default index.
XmlContainer::deleteDocument	Delete a document from the container.
XmlContainer::deleteIndex	Delete the specified index.
XmlContainer::getAllDocuments	Get all documents in the container.
XmlContainer::getAutoIndexing	Get the state of automatic indexing.
XmlContainer::getContainerConfig	Get the container's settings.
XmlContainer::getContainerType	Get the container's type.
XmlContainer::getDocument	Get the specified document.
XmlContainer::getFlags	Get the flags used to open the container.
XmlContainer::getIndexNodes	Return true if indexing nodes.
XmlContainer::getIndexSpecification	Get the index specification.
XmlContainer::getManager	Get the XmlManager object for this container.
XmlContainer::getName	Get the container's name.
XmlContainer::getNode	Get the specified node.
XmlContainer::getNumDocuments	Get the number of documents in the container.
XmlContainer::getPageSize	Return database page size.
XmlContainer::lookupIndex	Return all the documents matching a specified index.
XmlContainer::lookupStatistics	Return an XmlStatistics object for a specified index.
XmlContainer::putDocument	Add a document to the container.
XmlContainer::putDocumentAsEventWriter	Add a document to the container using XmlEventWriter.
XmlContainer::removeAlias	Remove a named alias for the container.
XmlContainer::replaceDefaultIndex	Replace the container's default index.
XmlContainer::replaceIndex	Replace an index of a specified type.
XmlContainer::setAutoIndexing	Set the state of automatic indexing.
XmlContainer::setIndexSpecification	Set the index specification.
XmlContainer::sync	Flush container database state to disk.
XmlContainer::updateDocument	Update an existing XmlDocument

XmlContainer::addAlias

```
#include <DbXml.hpp>

bool addAlias(const std::string &alias)
```

The `XmlContainer::addAlias` method adds a new name alias to the list maintained by the containing XmlManager (page 274). The new alias can then be used as a parameter to the collection() function in an XQuery expression. Returns true if the alias is successfully added. If the alias is already used by the containing XmlManager (page 274) object, false is returned.

Parameters

alias

The new alias to be added.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::addDefaultIndex

```
#include <DbXml.hpp>

void
XmlContainer::addDefaultIndex(
    const std::string &index,
    XmlUpdateContext &context)

XmlContainer::addDefaultIndex(
    XmlTransaction &txn,
    const std::string &index,
    XmlUpdateContext &context)
```

Adds a default index to the container This method is for convenience — see XmlIndexSpecification::addDefaultIndex (page 235) for more information.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time

decimal	gMonthDay

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

context

The update context to use for the index operation.

Errors

The `XmlContainer::addDefaultIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

UNKNOWN_INDEX

Unknown index specification

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::addIndex

```
#include <DbXml.hpp>

void
XmlContainer::addIndex(
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)

XmlContainer::addIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)

XmlContainer::addIndex(
    const std::string &uri,
    const std::string &name,
    XmlIndexSpecification::Type type,
    XmlValue::Type syntax,
    XmlUpdateContext &context)

XmlContainer::addIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    XmlIndexSpecification::Type type,
    XmlValue::Type syntax,
    XmlUpdateContext &context)
```

Adds an index of the specified type for the named document node. These method are for convenience — see XmlIndexSpecification::addIndex (page 239) for more information.

You can provide an index specification as a string value, or as enumerated types.

Specifying indexes as strings

```
#include <DbXml.hpp>

void
XmlContainer::addIndex(
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)

XmlContainer::addIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)
```

Identifies one or more indexing strategies to set for the identified node. The strategies are identified as a space-separated listing of strings.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string

dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be none or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

context

The update context to use for the index insertion.

Specifying indexes as enumerated values

```
#include <DbXml.hpp>

XmlContainer::addIndex(
    const std::string &uri,
    const std::string &name,
    XmlIndexSpecification::Type type,
    XmlValue::Type syntax,
    XmlUpdateContext &context)

XmlContainer::addIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    XmlIndexSpecification::Type type,
    XmlValue::Type syntax,
    XmlUpdateContext &context)
```

Identifies an indexing strategy to set for the identified node. The strategy is set using enumeration values for the index and the syntax.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`
- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`
- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`

- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`
- `XmlValue::DURATION`
- `XmlValue::FLOAT`
- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`
- `XmlValue::STRING`
- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

context

The update context to use for the index insertion.

Errors

The `XmlContainer::addIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

UNKNOWN_INDEX

Unknown index specification.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::deleteDefaultIndex

```
#include <DbXml.hpp>

void
XmlContainer::deleteDefaultIndex(const std::string &index,
    XmlUpdateContext &context)

void
XmlContainer::deleteDefaultIndex(XmlTransaction &txn,
    const std::string &index, XmlUpdateContext &context)
```

Deletes the default index for the container. This method is for convenience - see XmlIndexSpecification::deleteDefaultIndex (page 244) for more information.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
     unique-node-element-presence

     node-element-equality-string

     edge-element-presence-none

     node-element-equality-float
```

context

The XmlUpdateContext (page 415) to use for this operation.

Errors

The `XmlContainer::deleteDefaultIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

UNKNOWN_INDEX

Unknown index specification

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::deleteDocument

```
#include <DbXml.hpp>

void XmlContainer::deleteDocument(const std::string name,
    XmlUpdateContext &context)

void XmlContainer::deleteDocument(XmlDocument &document,
    XmlUpdateContext &context)

void XmlContainer::deleteDocument(XmlTransaction &txn,
    const std::string name, XmlUpdateContext &context)

void XmlContainer::deleteDocument(XmlTransaction &txn,
   XmlDocument &document, XmlUpdateContext &context)
```

The `XmlContainer::deleteDocument` method removes the specified XmlDocument (page 135) from the XmlContainer (page 16).

You can specify the document by name, or as a reference to an XmlDocument (page 135) object.

Deleting document by name

```
#include <DbXml.hpp>

void XmlContainer::deleteDocument(const std::string name,
    XmlUpdateContext &context)

void XmlContainer::deleteDocument(XmlTransaction &txn,
    const std::string name, XmlUpdateContext &context)
```

Delete the document with the given name. Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The name of the XmlDocument (page 135) to be deleted from the container.

context

The XmlUpdateContext (page 415) object to use for this deletion.

Deleting document by XmlDocument object

```
#include <DbXml.hpp>

void XmlContainer::deleteDocument(XmlDocument &document,
    XmlUpdateContext &context)

void XmlContainer::deleteDocument(XmlTransaction &txn,
   XmlDocument &document, XmlUpdateContext &context)
```

Removes the specified XmlDocument (page 135) from the XmlContainer (page 16).

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

document

The XmlDocument (page 135) to be deleted from the container. The name of the document to be deleted is extracted from this parameter.

context

The XmlUpdateContext (page 415) object to use for this deletion.

Errors

The `XmlContainer::deleteDocument` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

DOCUMENT_NOT_FOUND

The specified document is not in the XmlContainer (page 16).

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::deleteIndex

```
#include <DbXml.hpp>

void
XmlContainer::deleteIndex(
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)

void
XmlContainer::deleteIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)
```

Deletes an index of the specified type for the named document node. This method is for convenience — see XmlIndexSpecification::deleteIndex (page 248) for more information.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.

- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

context

The XmlUpdateContext (page 415) to use for this operation.

Errors

The `XmlContainer::deleteIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

UNKNOWN_INDEX

Unknown index specification

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getAllDocuments

```
#include <DbXml.hpp>

XmlResults XmlContainer::getAllDocuments(u_int32_t flags)

XmlResults XmlContainer::getAllDocuments(XmlTransaction &txn,
    u_int32_t flags)
```

Return all of the documents in the container in a lazily evaluated `XmlResult` set.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DBXML_WELL_FORMED_ONLY`

  Force the use of a scanner that will neither validate nor read schema or dtds associated with the document during parsing. This is efficient, but can cause parsing errors if the document references information that might have come from a schema or dtd, such as entity references.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_REVERSE_ORDER`

  Return results in reverse order relative to the sort of the index.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getAutoIndexing

```
#include <DbXml.hpp>

bool XmlContainer::getAutoIndexing() const

bool XmlContainer::getAutoIndexing(XmlTransaction &txn) const
```

Returns the current value of the auto-indexing state for the container. This state can be modified using XmlContainer::setAutoIndexing (page 70). See that page for a description of this feature.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

Errors

The `XmlContainer::getAutoIndexing` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getContainerConfig

```
#include <DbXml.hpp>

XmlContainerConfig XmlContainer::getContainerConfig() const;
```

Returns a copy of the `XmlContainerConfig` with the settings used by XmlManager::createContainer (page 281) or XmlManager::openContainer (page 310). This method replaces XmlContainer::getFlags (page 40). The settings of an open container cannot be changed. Some settings can be changed by closing the container and opening it again except for those properties which cannot be changed for existing containers. See the documentation for XmlContainerConfig (page 76) for more information.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getContainerType

```
#include <DbXml.hpp>

ContainerType XmlContainer::getContainerType() const
```

Returns the container's type. Possible return values are:

- `XmlContainer::NodeContainer`

  Documents are broken down into their component nodes, and these nodes are stored individually in the container. This is the preferred container storage type.

- `XmlContainer::WholedocContainer`

  Documents are stored intact; all white space and formatting is preserved.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getDocument

```
#include <DbXml.hpp>

XmlDocument getDocument(const std::string &name, u_int32_t flags = 0)

XmlDocument getDocument(
    XmlTransaction &txn, const std::string &name, u_int32_t flags = 0)
```

The `XmlContainer::getDocument` method returns the XmlDocument (page 135) with the specified name.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The name of the XmlDocument (page 135) to be retrieved from the container.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

Errors

The `XmlContainer::getDocument` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

DOCUMENT_NOT_FOUND

The specified document is not in the `XmlContainer`.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getFlags

```
#include <DbXml.hpp>

XmlContainerConfig XmlContainer::getFlags() const;
```

The `XmlContainer::getFlags` method returns a copy of the `XmlContainerConfig` with the settings used by XmlManager::createContainer (page 281) or XmlManager::openContainer (page 310).

This method is deprecated in favor of XmlContainer::getContainerConfig (page 36).

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getIndexNodes

```
#include <DbXml.hpp>

bool XmlContainer::getIndexNodes() const
```

Returns true if the container is configured to create node indices.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getIndexSpecification

```
#include <DbXml.hpp>

XmlIndexSpecification XmlContainer::getIndexSpecification()

XmlIndexSpecification
XmlContainer::getIndexSpecification(XmlTransaction &txn,
    u_int32_t flags = 0)
```

Retrieves the current indexing specification for the container. The indexing specification can be modified using XmlContainer::setIndexSpecification (page 72).

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

flags

This parameter must be set to one of the following values:

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

Errors

The `XmlContainer::getIndexSpecification` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getManager

```
#include <DbXml.hpp>

XmlManager &XmlContainer::getManager() const;
```

The `XmlContainer::getManager` method returns the XmlManager (page 274) object for the `XmlContainer`.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getName

```
#include <DbXml.hpp>

const std::string &XmlContainer::getName() const;
```

The `XmlContainer::getName` method returns the name of the `XmlContainer`.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getNode

```
#include <DbXml.hpp>

XmlValue getNode(const std::string &nodeHandle, u_int32_t flags = 0)

XmlDocument getNode(
    XmlTransaction &txn, const std::string &nodeHandle,
    u_int32_t flags = 0)
```

The `XmlContainer::getNode method returns the XmlValue (page 416) of type XmlValue::NODE` representing the specified handle. The handle must represent a node in a document in the XmlContainer (page 16). If the document or node has been removed, the operation may fail.

Node handles are guaranteed to remain stable in the absence of modifications to a document. If a document is modified, a handle may cease to exist, or may belong to a different node.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

nodeHandle

The handle representing the node which must have been obtained using XmlValue::getNodeHandle (page 432).

flags

This parameter must be set to 0 or one the following values:

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

Errors

The `XmlContainer::getNode` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

DOCUMENT_NOT_FOUND

The node and/or document specified by the handle is not in the `XmlContainer`.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getNumDocuments

```
#include <DbXml.hpp>

size_t XmlContainer::getNumDocuments() const

size_t XmlContainer::getNumDocuments(XmlTransaction &txn) const
```

The `XmlContainer::getNumDocuments` method returns the number of documents in the `XmlContainer`.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::getPageSize

```
#include <DbXml.hpp>

u_int32_t XmlContainer::getPageSize() const
```

Returns the actual database page size for the container.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::lookupIndex

```
#include <DbXml.hpp>

XmlResults XmlContainer::lookupIndex(XmlQueryContext &context,
    const std::string &uri, const std::string &name,
    const std::string &index, const XmlValue &value = XmlValue(),
    u_int32_t flags = 0)

XmlResults XmlContainer::lookupIndex(XmlTransaction &txn,
    XmlQueryContext &context, const std::string &uri,
    const std::string &name, const std::string &index,
    const XmlValue &value = XmlValue(), u_int32_t flags = 0)

XmlResults XmlContainer::lookupIndex(XmlQueryContext &context,
    const std::string &uri, const std::string &name,
    const std::string &parent_uri, const std::string &parent_name,
    const std::string &index,const XmlValue &value = XmlValue(),
    u_int32_t flags = 0)

XmlResults XmlContainer::lookupIndex(XmlTransaction &txn,
    XmlQueryContext &context, const std::string &uri,
    const std::string &name, const std::string &parent_uri,
    const std::string &parent_name, const std::string &index,
    const XmlValue &value = XmlValue(),u_int32_t flags = 0)
```

NOTE: this interface is deprecated, in favor of using XmlManager::createIndexLookup (page 287) and XmlIndexLookup::execute (page 223).

For a specified index, return all the data referenced by the index's keys, optionally matching a specific value. There are two forms of this method: one that you use for edge indexes, and one that you use for all other types of indexes.

Looking up non-edge indexes

```
#include <DbXml.hpp>

XmlResults XmlContainer::lookupIndex(XmlQueryContext &context,
    const std::string &uri, const std::string &name,
    const std::string &index, const XmlValue &value = XmlValue(),
    u_int32_t flags = 0)

XmlResults XmlContainer::lookupIndex(XmlTransaction &txn,
    XmlQueryContext &context, const std::string &uri,
    const std::string &name, const std::string &index,
    const XmlValue &value = XmlValue(), u_int32_t flags = 0)
```

Return all the targets for which the identified index has keys. By default, entire documents are returned by this method. However, if the container is of type `XmlContainer::NodeStorage` (the default container type), and if `DBXML_INDEX_NODES` is set for the container, then this method will return the individual nodes referenced by the index keys.

Note that you cannot use this form of this method to examine edge indices.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

context

The XmlQueryContext (page 341) to use for this query.

uri

The namespace of the node to which this index is applied.

name

The name of the node to which this index is applied.

index

Identifies the index for which you want the documents returned. The value supplied here must be a valid index. See XmlIndexSpecification::addIndex (page 239) for a description of valid index specifications.

value

Provides the value to which equality indices must be equal. This parameter is required when returning documents on equality indices, and it is ignored for all other types of indices.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_CACHE_DOCUMENTS`

  Enables use of a cache mechanism that optimizes XmlIndexLookup::execute (page 223) operations that a large number of nodes from the same document.

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DB_READ_UNCOMMITTED`

This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DBXML_NO_INDEX_NODES`

  Relevant for node storage containers with node indices only. Causes the XmlIndexLookup::execute (page 223) operations to return document nodes rather than direct pointers to the interior nodes. This is more efficient if all that is desired is a reference to target documents.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_REVERSE_ORDER`

  Return results in reverse order relative to the sort of the index.

Looking up edge indexes

```
#include <DbXml.hpp>

XmlResults XmlContainer::lookupIndex(XmlQueryContext &context,
    const std::string &uri, const std::string &name,
    const std::string &parent_uri, const std::string &parent_name,
    const std::string &index,const XmlValue &value = XmlValue(),
    u_int32_t flags = 0)

XmlResults XmlContainer::lookupIndex(XmlTransaction &txn,
    XmlQueryContext &context, const std::string &uri,
    const std::string &name, const std::string &parent_uri,
    const std::string &parent_name, const std::string &index,
    const XmlValue &value = XmlValue(),u_int32_t flags = 0)
```

Return all the targets for which the identified index has keys. By default, entire documents are returned by this method. However, if the container is of type `XmlContainer::NodeStorage` (the default container type), and if `DBXML_INDEX_NODES` is set for the container, then this method will return the individual nodes referenced by the index keys.

Use this form of this method to return documents indexed by edge indices.

Edge indices are indices maintained for those locations in a document where two nodes (a parent node and a child node) meet. See the Berkeley DB XML Getting Started Guide for details.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

context

The XmlQueryContext (page 341) to use for this query.

uri

The namespace of the node to which this index is applied.

name

The name of the node to which this index is applied.

parent_uri

The namespace of the parent node to which this edge index is applied.

parent_name

The name of the parent node to which this edge index is applied.

index

Identifies the index for which you want the documents returned. The value supplied here must be a valid index. See XmlIndexSpecification::addIndex (page 239) for a description of valid index specifications.

value

Provides the value to which equality indices must be equal. This parameter is required when returning documents on equality indices, and it is ignored for all other types of indices.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

Errors

The `XmlContainer::lookupIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::lookupStatistics

```
#include <DbXml.hpp>

XmlStatistics XmlContainer::lookupStatistics(const std::string &uri,
    const std::string &name, const std::string &index,
    const XmlValue &value = XmlValue())

XmlStatistics XmlContainer::lookupStatistics(XmlTransaction &txn,
    const std::string &uri, const std::string &name,
    const std::string &index, const XmlValue &value = XmlValue())

XmlStatistics XmlContainer::lookupStatistics(const std::string &uri,
    const std::string &name, const std::string &parent_uri,
    const std::string &parent_name, const std::string &index,
    const XmlValue &value = XmlValue())

XmlStatistics XmlContainer::lookupStatistics(XmlTransaction &txn,
    const std::string &uri, const std::string &name,
    const std::string &parent_uri, const std::string &parent_name,
    const std::string &index, const XmlValue &value = XmlValue())
```

Returns an XmlStatistics (page 403) object for the identified index. This object identifies the number of keys (both total and unique) maintained for the identified index.

There are two forms of this method: one that you use with non-edge indexes, and one that you use with edge indexes.

Looking up non-edge indexes

```
#include <DbXml.hpp>

XmlStatistics XmlContainer::lookupStatistics(const std::string &uri,
    const std::string &name, const std::string &index,
    const XmlValue &value = XmlValue())

XmlStatistics XmlContainer::lookupStatistics(XmlTransaction &txn,
    const std::string &uri, const std::string &name,
    const std::string &index, const XmlValue &value = XmlValue())
```

Lookup statistics for the identified index. Note that this form of this method cannot be used to return statistics on edge indices.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

uri

The namespace of the node to which this index is applied.

name

The name of the node to which this index is applied.

index

Identifies the index for which you want statistics returned. The value supplied here must be a valid index. See XmlIndexSpecification::addIndex (page 239) for a description of valid index specifications.

value

Provides the value to which equality indices must be equal. This parameter is required when returning documents on equality indices, and it is ignored for all other types of indices.

Looking up edge indexes

```
#include <DbXml.hpp>

XmlStatistics XmlContainer::lookupStatistics(const std::string &uri,
    const std::string &name, const std::string &parent_uri,
    const std::string &parent_name, const std::string &index,
    const XmlValue &value = XmlValue())

XmlStatistics XmlContainer::lookupStatistics(XmlTransaction &txn,
    const std::string &uri, const std::string &name,
    const std::string &parent_uri, const std::string &parent_name,
    const std::string &index, const XmlValue &value = XmlValue())
```

Lookup statistics for the identified index. Use this form of this method to return statistics on edge indices.

Edge indices are indices maintained for those locations in a document where two nodes (a parent node and a child node) meet. See the Berkeley DB XML Getting Started Guide for details.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

uri

The namespace of the node to which this index is applied.

name

The name of the node to which this index is applied.

parent_uri

The namespace of the parent node to which this edge index is applied.

parent_name

The name of the parent node to which this edge index is applied.

index

Identifies the index for which you want statistics returned. The value supplied here must be a valid index. See XmlIndexSpecification::addIndex (page 239) for a description of valid index specifications.

value

Provides the value to which equality indices must be equal. This parameter is required when returning documents on equality indices, and it is ignored for all other types of indices.

Errors

The `XmlContainer::lookupStatistics` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::putDocument

```
#include <DbXml.hpp>

void XmlContainer::putDocument(XmlDocument &document,
    XmlUpdateContext &context, u_int32_t flags = 0)

void XmlContainer::putDocument(XmlTransaction &txn,
    XmlDocument &document, XmlUpdateContext &context,
    u_int32_t flags = 0)

std::string XmlContainer::putDocument(const std::string &name,
    XmlEventReader &reader, XmlUpdateContext &context,
    u_int32_t flags = 0)

std::string XmlContainer::putDocument(XmlTransaction &txn,
    const std::string &name, XmlEventReader &reader,
    XmlUpdateContext &context, u_int32_t flags = 0)

std::string XmlContainer::putDocument(const std::string &name,
    XmlInputStream *adopted_input,
    XmlUpdateContext &context, u_int32_t flags = 0)

std::string XmlContainer::putDocument(XmlTransaction &txn,
    const std::string &name, XmlInputStream *adopted_input,
    XmlUpdateContext &context, u_int32_t flags = 0)

std::string XmlContainer::putDocument(const std::string &name,
    const std::string &contents, XmlUpdateContext &context,
    u_int32_t flags = 0)

std::string XmlContainer::putDocument(XmlTransaction &txn,
    const std::string &name, const std::string &contents,
    XmlUpdateContext &context, u_int32_t flags = 0)
```

Inserts an XmlDocument (page 135) into the container. The value returned by this method is dependent upon the form of the method that you used to perform the insertion.

Note that the name used for the document must be unique in the container or an exception is thrown. The flag, `DBXML_GEN_NAME`, can be used to generate a name. To change a document that already exists in the container, use XmlContainer::updateDocument (page 74).

The document content is indexed according to the container indexing specification. The indexer supports the Xerces content encodings and expects the content to be well-formed , but it need not be valid .

There are four different forms of this method:

- Add a document as an XmlDocument object (page 58)
- Add a document using an XmlEventReader (page 59)

- Add a document using an XmlInputStream (page 60)
- Add a document using a string (page 61)

Each of these forms are described in the following sections.

Add a document as an XmlDocument object

```
#include <DbXml.hpp>

void XmlContainer::putDocument(XmlDocument &document,
    XmlUpdateContext &context, u_int32_t flags = 0)

void XmlContainer::putDocument(XmlTransaction &txn,
    XmlDocument &document, XmlUpdateContext &context,
    u_int32_t flags = 0)
```

Inserts the XmlDocument (page 135) provided on the call to the container. The name provided for the XmlDocument (page 135) must be unique to the container or an exception is thrown. To set the name, use XmlDocument::setName (page 151).

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

document

The XmlDocument (page 135) to be inserted into the XmlContainer (page 16)

context

The update context to use for the document insertion.

flags

This parameter must be set to 0 or the following values:

- DBXML_GEN_NAME

  Generate a unique name. If no name is set for this XmlDocument (page 135), a system-defined unique name is generated. If a name is specified, a unique string is appended to that name to ensure uniqueness.

- DBXML_WELL_FORMED_ONLY

  Force the use of a scanner that will neither validate nor read schema or dtds associated with the document during parsing. This is efficient, but can cause parsing errors if the document references information that might have come from a schema or dtd, such as entity references.

Add a document using an XmlEventReader

```
#include <DbXml.hpp>

std::string XmlContainer::putDocument(const std::string &name,
    XmlEventReader &reader, XmlUpdateContext &context,
    u_int32_t flags = 0)

std::string XmlContainer::putDocument(XmlTransaction &txn,
    const std::string &name, XmlEventReader &reader,
    XmlUpdateContext &context, u_int32_t flags = 0)
```

Inserts the XML document referenced by the XmlEventReader (page 152) into the container. The name used for the new document is returned by this method.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

Provides the name of the document to insert into the container. This name must be unique in the container. If `DBXML_GEN_NAME` is set, a system-defined string is appended to create a unique name. This applies if the name parameter is provided or empty. If the name is not unique within the container, an exception is thrown.

reader

Identifies the source of information used to create the document. The content will be created by calling methods on this object. When the end of the document is read, the XmlEventReader::close (page 156) method will be called. The XmlEventReader (page 152) object must have been created via one of these methods: XmlDocument::getContentAsEventReader (page 139), XmlValue::asEventReader (page 422), or an application-created derived class of XmlEventReader (page 152).

context

The update context to use for the document insertion.

flags

This parameter must be set to 0 or the following values:

- `DBXML_GEN_NAME`

  Generate a unique name. If no name is set for this XmlDocument (page 135), a system-defined unique name is generated. If a name is specified, a unique string is appended to that name to ensure uniqueness.

Add a document using an XmlInputStream

```
#include <DbXml.hpp>

std::string XmlContainer::putDocument(const std::string &name,
    XmlInputStream *adopted_input,
    XmlUpdateContext &context, u_int32_t flags = 0)

std::string XmlContainer::putDocument(XmlTransaction &txn,
    const std::string &name, XmlInputStream *adopted_input,
    XmlUpdateContext &context, u_int32_t flags = 0)
```

Inserts the XML document contained in the XmlInputStream (page 270) into the container. The name used for the new document is returned by this method.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

Provides the name of the document to insert into the container. This name must be unique in the container. If `DBXML_GEN_NAME` is set, a system-defined string is appended to create a unique name. This applies if the name parameter is provided or empty. If the name is not unique within the container, an exception is thrown.

adopted_input

Identifies the input stream to use to read the document. Create the input stream using one of XmlManager::createLocalFileInputStream (page 290), XmlManager::createMemBufInputStream (page 291), XmlManager::createStdInInputStream (page 295), XmlManager::createURLInputStream (page 300), or XmlDocument::getContentAsXmlInputStream (page 141). The content read by the input stream must well-formed XML, or an exception is thrown. The XmlInputStream (page 270) object provided is consumed (deleted) by this method.

context

The update context to use for the document insertion.

flags

This parameter must be set to 0 or the following values:

- `DBXML_GEN_NAME`

  Generate a unique name. If no name is set for this XmlDocument (page 135), a system-defined unique name is generated. If a name is specified, a unique string is appended to that name to ensure uniqueness.

Add a document using a string

```
#include <DbXml.hpp>

std::string XmlContainer::putDocument(const std::string &name,
    const std::string &contents, XmlUpdateContext &context,
    u_int32_t flags = 0)

std::string XmlContainer::putDocument(XmlTransaction &txn,
    const std::string &name, const std::string &contents,
    XmlUpdateContext &context, u_int32_t flags = 0)
```

Inserts the XML document contained in the string into the container. The name used for the new document is returned by this method.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

Provides the name of the document to insert into the container. This name must be unique in the container. If `DBXML_GEN_NAME` is set, a system-defined string is appended to create a unique name. This applies if the name parameter is provided or empty. If the name is not unique within the container, an exception is thrown.

contents

The XML content to insert into the container. The content contained in this string must well-formed XML, or an exception is thrown.

context

The update context to use for the document insertion.

flags

This parameter must be set to 0 or the following values:

- `DBXML_GEN_NAME`

  Generate a unique name. If no name is set for this XmlDocument (page 135), a system-defined unique name is generated. If a name is specified, a unique string is appended to that name to ensure uniqueness.

Errors

The `XmlContainer::putDocument` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

EVENT_ERROR
event reader

An error occured while adding a document using an XmlEventReader (page 152). This error can only be thrown if you are adding documents to your container using an XmlEventReader (page 152).

INDEXER_PARSER_ERROR

The XML Indexer could not parse the document. This error can not be thrown if you are using an XmlEventReader (page 152) to add the document to your container.

UNIQUE_ERROR

The document does not have a name that is unique for the container.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::putDocumentAsEventWriter

```
#include <DbXml.hpp>

XmlEventWriter &XmlContainer::putDocumentAsEventWriter(
    XmlDocument &document, XmlUpdateContext &context,
    u_int32_t flags = 0)

XmlEventWriter &XmlContainer::putDocumentAsEventWriter(
    XmlTransaction &txn, XmlDocument &document,
    XmlUpdateContext &context, u_int32_t flags = 0)
```

Begins insertion of an XmlDocument (page 135) into the container through use of an XmlEventWriter (page 192) object. Methods must be called on the returned XmlEventWriter (page 192) to create content for the document, which is completed by calling XmlEventWriter::close (page 194). If XmlEventWriter::close (page 194) is never called, the document insertion will not be complete, and the container may be left in an inconsistent state.

The name used for the document must be unique in the container or an exception is thrown. The flag, `DBXML_GEN_NAME`, can be used to generate a name. To change a document that already exists in the container, use XmlContainer::updateDocument (page 74).

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

document

The XmlDocument (page 135) to be inserted into the `XmlContainer`.

context

The update context to use for the document insertion.

flags

This parameter must be set to 0 or the following value:

- `DBXML_GEN_NAME`

  Generate a unique name. If no name is set for this XmlDocument (page 135), a system-defined unique name is generated. If a name is specified, a unique string is appended to that name to ensure uniqueness.

Errors

The `XmlContainer::putDocumentAsEventWriter` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

UNIQUE_ERROR

Uniqueness constraint violation for key

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::removeAlias

```
#include <DbXml.hpp>

bool removeAlias(const std::string &alias)
```

The `XmlContainer::removeAlias` method removes the named alias from the list maintained by the containing XmlManager (page 274). If the alias does not exist, or matches a different `XmlContainer`, the call fails. Return value is true upon success, false upon failure.

Parameters

alias

The alias to remove.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::replaceDefaultIndex

```
#include <DbXml.hpp>

void
XmlContainer::replaceDefaultIndex(
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)

XmlContainer::replaceDefaultIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)
```

Replaces the container's default index. This method is for convenience -- see XmlIndexSpecification::replaceDefaultIndex (page 260) for more information.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear

base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be none or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

context

The update context to use for the index replacement.

Errors

The `XmlContainer::replaceDefaultIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::replaceIndex

```
#include <DbXml.hpp>

void
XmlContainer::replaceIndex(
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)

XmlContainer::replaceIndex(
    XmlTransaction &txn,
    const std::string &uri,
    const std::string &name,
    const std::string &index,
    XmlUpdateContext &context)
```

Replaces an index of the specified type for the named document node. This method is for convenience -- see XmlIndexSpecification::replaceIndex (page 264) for more information.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.

- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

context

The update context to use for this operation.

Errors

The `XmlContainer::replaceIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::setAutoIndexing

```
#include <DbXml.hpp>

void XmlContainer::setAutoIndexing(bool value,
    XmlUpdateContext &context)

void XmlContainer::setAutoIndexing(XmlTransaction &txn,
    bool value, XmlUpdateContext &context)
```

Sets the auto-indexing state of the container.

If the value on the container is true (the default for newly-created containers) then indexes are added automatically for leaf elements and attributes. The indexes added are "node-equality-string" and "node-equality-double" for elements and attributes. If auto-indexing is not desired it should be disabled using this interface immediately after container creation. Auto-indexing is recognized by insertion of new documents as well as updates of existing documents, including modification via XQuery Update. The auto-indexing state is persistent and will remain stable across container close/re-open operations. Indexes added via auto-indexing are normal indexes and can be removed using the normal mechanisms.

A significant implication of auto-indexing is that any operation that may add an index (e.g. XmlContainer::putDocument (page 57)) can have the side effect of reindexing the entire container. For this reason auto-indexing is not recommended for containers of heterogenous documents and that it be disabled once a representative set of documents has been inserted.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

value

A boolean value indicating whether auto-indexing behavior should be enabled or disabled.

Errors

The `XmlContainer::setAutoIndexing` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::setIndexSpecification

```
#include <DbXml.hpp>

void
XmlContainer::setIndexSpecification(const XmlIndexSpecification &index,
    XmlUpdateContext &context)

void
XmlContainer::setIndexSpecification(XmlTransaction &txn,
    const XmlIndexSpecification &index, XmlUpdateContext &context)
```

Defines the type of indexing to be maintained for a container of documents. The currently defined indexing specification can be retrieved with the XmlContainer::getIndexSpecification (page 42) method.

If the container is not empty then the contained documents are incrementally indexed. Index keys for disabled index strategies are removed and index keys for enabled index strategies are added. Note that the length of time taken to perform this re-indexing operation is proportional to the size of the container.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

index

The indexing specification for the container.

context

The update context to use for the index modification.

Errors

The `XmlContainer::setIndexSpecification` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::sync

```
#include <DbXml.hpp>

void XmlContainer::sync() const;
```

The `XmlContainer::sync` method flushes database pages for the container to disk.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

XmlContainer::updateDocument

```
#include <DbXml.hpp>

void XmlContainer::updateDocument(XmlDocument &document,
    XmlUpdateContext &context)

void XmlContainer::updateDocument(XmlTransaction &txn,
    XmlDocument &document, XmlUpdateContext &context)
```

Updates an XmlDocument (page 135) in the container. The document must have been retrieved from the container using XmlContainer::getDocument (page 38), XmlManager::query (page 316), or XmlQueryExpression::execute (page 364). It is possible to use a constructed `XmlDocument` object, if its name is set to a valid name in the container. The document must still exist within the container. The document content is indexed according to the container indexing specification, with index keys being removed for the previous document content, and added for the updated document content.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

document

The XmlDocument (page 135) to be updated in the `XmlContainer`.

context

The update context to use for the document insertion.

Errors

The `XmlContainer::updateDocument` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

DOCUMENT_NOT_FOUND

The specified document is not in the `XmlContainer`.

INDEXER_PARSER_ERROR

The XML Indexer could not parse the document.

Class

XmlContainer (page 16)

See Also

XmlContainer Methods (page 17)

Chapter 6. XmlContainerConfig

```
#include <DbXml.hpp>

class DbXml::XmlContainerConfig {
public:
 XmlContainerConfig();
 XmlContainerConfig(const XmlContainerConfig &);
 ~XmlContainerConfig();
 XmlContainerConfig &operator = (const XmlContainerConfig &)
 enum ConfigState {
      On
      Off
      UseDefault
 };
 ...
};
```

The `XmlContainerConfig` class encapsulates all the properties with which a container can be created or opened. It is passed as an argument to XmlManager::createContainer (page 281) and XmlManager::openContainer (page 310) as well as other methods that previously accepted unsigned int flags paramters.

The default settings of the properties in an `XmlContainerConfig` object are listed in the following table.

Property	Value
AllowCreate	false
AllowValidation	false
Checksum	false
CompressionName	DEFAULT_COMPRESSION — on for WholedocContainer; off for NodeContainer
ContainerType	`XmlContainer::NodeContainer`
Encrypted	false
ExclusiveCreate	false
IndexNodes	On for NodeContainer; Off for WholedocContainer
Mode	0
Multiversion	false
NoMMap	false
PageSize	8192 for NodeContainer; 16384 for WholedocContainer
ReadOnly	false
ReadUncommitted	false

Property	Value
SequenceIncrement	5
Statistics	On
Threaded	false
TransactionNotDurable	false

This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlContainerConfig Methods

XmlContainerConfig Methods	Description
XmlContainerConfig::getAllowCreate	Returns whether the container will be created if it does not exist.
XmlContainerConfig::getAllowValidation	Returns whether XML is validated.
XmlContainerConfig::getChecksum	Gets whether checksum verification is in use.
XmlContainerConfig::getCompressionName	Returns the compression to be used.
XmlContainerConfig::getContainerType	Returns the container type.
XmlContainerConfig::getEncrypted	Returns whether encryption is in use.
XmlContainerConfig::getExclusiveCreate	Returns whether an exception is thrown if the container exists.
XmlContainerConfig::getIndexNodes	Returns whether nodes and documents are indexed.
XmlContainerConfig::getMode	Returns the mode of the container files.
XmlContainerConfig::getMultiversion	Returns whether multiversion concurrency control is enabled.
XmlContainerConfig::getNoMMap	Returns whether containers are mapped into process memory.
XmlContainerConfig::getPageSize	Returns the page size used by the container.
XmlContainerConfig::getReadOnly	Returns whether the container is read only.
XmlContainerConfig::getReadUncommitted	Returns whether dirty reads are enabled for the container.
XmlContainerConfig::getSequenceIncrement	Returns the sequence number generation cache size.
XmlContainerConfig::getStatistics	Returns whether to store structural statistics.
XmlContainerConfig::getThreaded	Returns whether the container handle is free-threaded.
XmlContainerConfig::getTransactional	Returns whether transactions are used.
XmlContainerConfig::getTransactionNotDurable	Returns whether operations are not durable.
XmlContainerConfig::setAllowCreate	Sets whether to create the container if it does not exist.
XmlContainerConfig::setAllowValidation	Sets whether XML is validated.
XmlContainerConfig::setChecksum	Sets whether to use checksum verification.
XmlContainerConfig::setCompressionName	Sets the compression to be used.
XmlContainerConfig::setContainerType	Sets the container type.
XmlContainerConfig::setEncrypted	Sets whether to use encryption.

XmlContainerConfig Methods	Description
XmlContainerConfig::setExclusiveCreate	Sets whether to throw an exception if the container exists.
XmlContainerConfig::setIndexNodes	Sets whether to index nodes or documents.
XmlContainerConfig::setMode	Sets the mode of the container files.
XmlContainerConfig::setMultiversion	Enable multiversion concurrency control.
XmlContainerConfig::setNoMMap	Sets whether to map containers into process memory.
XmlContainerConfig::setPageSize	Sets the page size used by the container.
XmlContainerConfig::setReadOnly	Sets whether the container is read only.
XmlContainerConfig::setReadUncommitted	Enable dirty reads.
XmlContainerConfig::setSequenceIncrement	Set the sequence number generation cache size.
XmlContainerConfig::setStatistics	Sets whether to store structural statistics.
XmlContainerConfig::setThreaded	Sets whether to return the container handle as free-threaded.
XmlContainerConfig::setTransactional	Sets whether transactions are used.
XmlContainerConfig::setTransactionNotDurable	Sets whether operations are not durable.

XmlContainerConfig::getAllowCreate

```
#include <DbXml.hpp>

bool XmlContainerConfig::getAllowCreate() const
```

Returns whether a container can be created during a call to XmlManager::openContainer (page 310) if it does not already exist. This value can be set using the XmlContainerConfig::setAllowCreate (page 99) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getAllowValidation

```
#include <DbXml.hpp>

bool XmlContainerConfig::getAllowValidation() const
```

Returns whether XML is validated when it refers to a DTD or XML Schema. This value can be set using the XmlContainerConfig::setAllowValidation (page 100) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getChecksum

```
#include <DbXml.hpp>

bool XmlContainerConfig::getChecksum() const
```

Returns whether checksum verification is in use. You can manage this value using the XmlContainerConfig::setChecksum (page 101) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getCompressionName

```
#include <DbXml.hpp>

const char *XmlContainerConfig::getCompressionName() const
```

Returns the compression object in use by the container. You can manage this value using the XmlContainerConfig::setCompressionName (page 102) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getContainerType

```
#include <DbXml.hpp>

XmlContainer::ContainerType
XmlContainerConfig::getContainerType() const
```

Returns the type of container that will be created. This value is set using the XmlContainerConfig::setContainerType (page 103) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getEncrypted

```
#include <DbXml.hpp>

bool XmlContainerConfig::getEncrypted() const
```

Returns whether underlying databases are encrypted. This value can be managed using the XmlContainerConfig::setEncrypted (page 104) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getExclusiveCreate

```
#include <DbXml.hpp>

bool XmlContainerConfig::getExclusiveCreate() const
```

Returns whether the container is configured to use exclusive create. This value can be managed using the XmlContainerConfig::setExclusiveCreate (page 105) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getIndexNodes

```
#include <DbXml.hpp>

XmlContainerConfig::ConfigState
XmlContainerConfig::getIndexNodes() const
```

Returns whether the index targets reference nodes or documents. This value can be managed using the XmlContainerConfig::setIndexNodes (page 106) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getMode

```
#include <DbXml.hpp>

int XmlContainerConfig::getMode() const
```

Returns the mode used for files created for the container on UNIX or IEEE/ANSI Std 1003.1 (POSIX) environments. This value is ignored on Windows systems. You can use the XmlContainerConfig::setMode (page 107) method to manage this value.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getMultiversion

```
#include <DbXml.hpp>

bool XmlContainerConfig::getMultiversion() const
```

Returns whether multiversion concurrency support is configured for the container. You can manage this value by using the XmlContainerConfig::setMultiversion (page 108) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getNoMMap

```
#include <DbXml.hpp>

bool XmlContainerConfig::getNoMMap() const
```

Returns whether the container will be mapped into process memory. This value can be set using the XmlContainerConfig::setNoMMap (page 109) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getPageSize

```
#include <DbXml.hpp>


u_int32_t XmlContainerConfig::getPageSize()
```

Returns the underlying database page size, in bytes. This value can be managed using the XmlContainerConfig::setPageSize (page 110) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getReadOnly

```
#include <DbXml.hpp>

bool XmlContainerConfig::getReadOnly() const
```

Returns whether the container is open for read-only. This property can be set using the XmlContainerConfig::setReadOnly (page 111) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getReadUncommitted

```
#include <DbXml.hpp>

bool XmlContainerConfig::getReadUncommitted() const
```

Returns whether dirty reads are enabled for the container. This value can be managed using the XmlContainerConfig::setReadUncommitted (page 112) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getSequenceIncrement

```
#include <DbXml.hpp>

u_int32_t XmlContainerConfig::getSequenceIncrement()
```

Returns the integer increment to be used when pre-allocating document ids for new documents created by XmlContainer::putDocument (page 57). This value can be managed using the XmlContainerConfig::setSequenceIncrement (page 113) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getStatistics

```
#include <DbXml.hpp>

XmlContainerConfig::ConfigState
XmlContainerConfig::getStatistics() const
```

Returns whether structural statistics are stored for the container. This value can be managed using the XmlContainerConfig::setStatistics (page 114) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getThreaded

```
#include <DbXml.hpp>

bool XmlContainerConfig::getThreaded() const
```

Returns whether the container is configured to be thread-safe. This property can be managed using the XmlContainerConfig::setThreaded (page 115) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getTransactional

```
#include <DbXml.hpp>

bool XmlContainerConfig::getTransactional() const
```

Returns whether the container is configured for transactional use. This value can be managed using the XmlContainerConfig::setTransactional (page 116) method.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::getTransactionNotDurable

```
#include <DbXml.hpp>

bool XmlContainerConfig::getTransactionNotDurable() const
```

Returns whether log records are written for updates made to this container. This value can be managed using the XmlContainerConfig::setTransactionNotDurable (page 117) method. If this method returns `true`, then log records are not written for updates to this container, and so the updates are not durable.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setAllowCreate

```
#include <DbXml.hpp>

void XmlContainerConfig::setAllowCreate(bool value)
```

If set to `true` a container can be created during a call to XmlManager::openContainer (page 310) if it does not already exist. The default value is `false`.

Parameters

value

If set to `true` a container will be created on calls to XmlManager::openContainer (page 310) if it does not exist. If set to `false` an exception will be thrown when XmlManager::openContainer (page 310) is called for a container that does not exist.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setAllowValidation

```
#include <DbXml.hpp>

void XmlContainerConfig::setAllowValidation(bool value)
```

Sets whether to validate XML if it refers to a DTD or XML Schema. If enabled validation is only performed on document insertion or update and not when modified via XQuery Update expressions. The default value is `false` and is used by container open.

Parameters

value

Set to `true` in order to validate XML.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setChecksum

```
#include <DbXml.hpp>

void XmlContainerConfig::setChecksum(bool value)
```

Sets whether to do checksum verification of pages read into the cache from the backing filestore. Berkeley DB XML uses the SHA1 Secure Hash Algorithm if encryption is configured and a general hash algorithm if it is not. The default value is `false`.

Parameters

value

Set to true to perform checksum verification.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setCompressionName

```
#include <DbXml.hpp>

void XmlContainerConfig::setCompressionName(const char *name)
```

Sets the name of the compression object to be used by the container. Compression is only used by whole document storage containers. If a compression name is set and a container is created or opened the name must match one that has been registered with the XmlManager (page 274) using XmlManager::registerCompression (page 319). The default name is the name of the default compression algorithm. If compression is disabled this setting is ignored. The compression name can only be set at creation time, afterwards the container must always be open with the same compression name and registered object.

Parameters

name

The name of the registered compression object to be used by the container. Built-in values are `XmlContainerConfig::NO_COMPRESSION`, which skips compression and `XmlContainerConfig::DEFAULT_COMPRESSION` which uses the built in compression algorithm. Built-in types are pre-registered with the XmlManager (page 274) if compression was enabled during the product build.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setContainerType

```
#include <DbXml.hpp>

void XmlContainerConfig::setContainerType(
        XmlContainer::ContainerType type)
```

Sets the type of container to be created. The default value is `XmlContainer::NodeContainer`. This value is ignored if the container already exists.

Parameters

type

Identifies the type of container to create. The container type must be one of the following values:

- `XmlContainer::NodeContainer`

  Documents are broken down into their component nodes, and these nodes are stored individually in the container. This is the preferred container storage type.

- `XmlContainer::WholedocContainer`

  Documents are stored intact; all white space and formatting is preserved.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setEncrypted

```
#include <DbXml.hpp>

void XmlContainerConfig::setEncrypted(bool value)
```

Sets whether to encrypt the database using the cryptographic password specified to `DbEnv::set_encrypt()`. The default setting is `false`.

Parameters

value

Set to true to use encryption.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setExclusiveCreate

```
#include <DbXml.hpp>

void XmlContainerConfig::setExclusiveCreate(bool value)
```

If set to `true` XmlManager::openContainer (page 310) and XmlManager::createContainer (page 281) will throw exceptions if the container already exists and XmlContainerConfig::setAllowCreate (page 99) has been set to `true`. The default value is `false`.

Parameters

value

Set to `true` to use exclusive create.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setIndexNodes

```
#include <DbXml.hpp>

void XmlContainerConfig::setIndexNodes(
            XmlContainerConfig::ConfigState state)
```

Sets whether the index targets reference nodes or documents. The default setting is `UseDefault`. This value is ignored unless the container is being created or reindexed.

If set to `On` it causes the indexer to create index targets that reference nodes rather than documents. This allows index lookups during query processing to more efficiently find target nodes and avoid walking the document tree. It can apply to both container types, and is the default for containers of type `XmlContainer::NodeContainer`.

If set to `Off` it causes the indexer to create index targets that reference documents rather than nodes. This can be more desirable for simple queries that only need to return documents and do relatively little navigation during querying. It can apply to both container types, and is the default for containers of type `XmlContainer::WholedocContainer`.

If set to `UseDefault`, then the container type will decide whether nodes or documents are referenced.

Parameters

state

Whether the index references nodes or documents. The container property must have one of the following values:

- `XmlContainerConfig::On`

  The container property is turned on.

- `XmlContainerConfig::Off`

  The container property is turned off.

- `XmlContainerConfig::UseDefaults`

  The container property is set to whatever the default is for the given container type.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setMode

```
#include <DbXml.hpp>

void XmlContainerConfig::setMode(int mode)
```

Sets the mode for the files created for the container. This value is ignored on Windows or if the container already exists.

Parameters

mode

On Windows systems, mode is ignored.

On UNIX systems or in IEEE/ANSI Std 1003.1 (POSIX) environments, files are created with mode `mode` (as described in `chmod(2)`) and modified by the process' umask value at the time of creation (see `umask(2)`). Created files are owned by the process owner; the group ownership of created files is based on the system and directory defaults, and is not further specified by Berkeley DB. System shared memory segments created by are created with mode `mode`, unmodified by the process' umask value. If `mode` is 0, DB XML will use a default mode of readable and writable by both owner and group.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setMultiversion

```
#include <DbXml.hpp>

void XmlContainerConfig::setMultiversion(bool value)

bool XmlContainerConfig::getMultiversion() const
```

If set to true then the database will be opened with support for multiversion concurrency control. multiversion concurrency control. This will cause updates to the container to follow a copy-on-write protocol which is required to support snapshot isolation. The `DB_MULTIVERSION` flag requires that the container be transactionally protected during its open. The default value is false.

Parameters

value

Set to true to support multiversion concurrency control.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setNoMMap

```
#include <DbXml.hpp>

void XmlContainerConfig::setNoMMap(bool value)
```

If set to true then the container will not be mapped into process memory (see the `DbEnv::set_mp_mmapsize()` method for further information). The default value is `false`.

Parameters

value

If `true`, then the container will not be mapped into process memory.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setPageSize

```
#include <DbXml.hpp>

void XmlContainerConfig::setPageSize(u_int32_t pageSize)
```

This method sets the size of the pages used to store documents in the database. The size is specified in bytes in the range 512 bytes to 64K bytes. If no page size is specified the system will create containers with page size 8192 for node storage containers and 16384 for wholedoc storage containers. Page size affects the amount of I/O performed as well as granularity of locking as Berkeley DB performs page-level locking. These needs are often at odds with one another. The page size cannot be changed once a container has been created. This value is ignored unless the container is being created.

Parameters

pageSize

The page size in bytes.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setReadOnly

```
#include <DbXml.hpp>

void XmlContainerConfig::setReadOnly(bool value)
```

If set to `true` the container is opened for reading only. Any attempt to modify items in the container will fail regardless of the permissions of the underlying files. If set to `false` then the container can be modified. The default value is `false` and can only be set to `true` for existing containers.

Parameters

value

When set to `true`, the container cannot be modified.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setReadUncommitted

```
#include <DbXml.hpp>

void XmlContainerConfig::setReadUncommitted(bool value)
```

If set to true then the container will support degree 1 isolation; that is, read operations may return information that has been modified by another transaction but has not yet been committed. This setting should be used rarely if at all. The default value is `false`.

Parameters

value

If set to `true` then the container will support degree 1 isolation.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setSequenceIncrement

```
#include <DbXml.hpp>

void XmlContainerConfig::setSequenceIncrement(u_int32_t incr)
```

Sets the integer increment to be used when pre-allocating document ids for new documents created by XmlContainer::putDocument (page 57).

Every document added to an `XmlContainer` is assigned an internal unique ID, and BDB XML performs an internal database operation to obtain these IDs. In order to increase database concurrency and improve performance of ID allocation, BDB XML pre-allocates a sequence of these numbers. The size of this sequence is determined by the value specified here. The default ID sequence size is 5.

Be aware that when a container is closed, any unused IDs in the current sequence are lost. Under some extreme cases, this can result in a container to which documents can no longer be added. For example, setting this value to a very large number (such as, say, 1 million) and then repeatedly opening and closing the container while adding a few documents may eventually cause the container to run out of IDs. Once out of IDs, the container will never again be able to accept new documents. However, document IDs are 64-bit quantities so this is extremely unlikely.

You should almost always leave this value alone. However, if you are loading a large number of documents to a container all at once, you may find a small performance benefit to setting the sequence number to a larger value. If you do this, be aware that this value is persistent across container opens, so you should take care to reset the value to its default once you are done loading the documents.

Parameters

incr

The increment to use for pre-allocated IDs.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setStatistics

```
#include <DbXml.hpp>

void XmlContainerConfig::setStatistics(
        XmlContainerConfig::ConfigState state)
```

Sets whether structural statistics are stored in the container. These statisitics are used in query optimization. The default setting is `UseDefault` and the default is to use statistics. This value is ignored unless the container is being created or reindexed.

Structural statistics information is very useful for cost based query optimisation. Containers created with these statistics will take slightly longer to load and update, since the statistics must also be updated. In addition the statistics affect the concurrent behavior in the face of updates.

If the state is set to `Off` then the container is created without structural statistics. If the state is set to `On` or `UseDefault` the container is created with structural statistics.

Parameters

state

Whether the container includes structural statistics. The container property must have one of the following values:

- `XmlContainerConfig::On`

  The container property is turned on.

- `XmlContainerConfig::Off`

  The container property is turned off.

- `XmlContainerConfig::UseDefaults`

  The container property is set to whatever the default is for the given container type.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setThreaded

```
#include <DbXml.hpp>

void XmlContainerConfig::setThreaded(bool value)
```

Causes the container handle to be free-threaded; that is, concurrently usable by multiple threads in the address space. If multiple threads access a container that does not have this property set the results are unpredictable. The default value is `false`.

Parameters

value

Set to `true` if multiple threads will use the container.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setTransactional

```
#include <DbXml.hpp>

void XmlContainerConfig::setTransactional(bool value)
```

If set to `true`, the container is configured for transactional use. Even if the environment has been configured for transactions, this property must be used in order to create or open a transactional container. In other words, a transactional environment can support both transactional and non-transactional containers. If the environment has not also been configured for transactions then use of this property when opening a container will result in an exception.

If this property is set, an XmlTransaction (page 407) object can and should be passed to any method that supports it. If a container is transactional and an explicit XmlTransaction (page 407) object is not passed to a modifying method (e.g. XmlContainer::putDocument (page 57) a transaction is automatically created, used and committed on behalf of the application. While this is handy it is highly recommended that applications manage their own transactions in order to better handle deadlock and other exceptions that may occur. The default value for this property is `false` and it affects containers being created and containers that already exist.

Parameters

value

Set to `true` to enable transactions.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

XmlContainerConfig::setTransactionNotDurable

```
#include <DbXml.hpp>

void XmlContainerConfig::setTransactionNotDurable(bool value)
```

If set to `true`, Berkeley DB XML will not write log records for this container. This means that updates of this container exhibit the ACI (atomicity, consistency, and isolation) properties, but not D (durability); that is, database integrity will be maintained, but if the application or system fails, integrity will not persist. The underlying database file must be verified and/or restored from backup after a failure. The default value is `false`.

Parameters

value

If set to `true` then updates will not be durable.

Class

XmlContainerConfig (page 76)

See Also

XmlContainerConfig Methods (page 78)

Chapter 7. XmlData

```
#include <DbXml.hpp>

class DbXml::XmlData {
public:
    XmlData::XmlData()
    XmlData::XmlData(const Dbt &dbt)
    XmlData::XmlData(void *data, size_t size)
    XmlData::XmlData(const XmlData &o)
    XmlData &operator = (const XmlData &o)
    virtual XmlData::~XmlData()
    ...

    void set(const void *data, size_t size);
    void append(const void *data, size_t size);

    void * get_data() const;

    size_t get_size() const; void set_size(size_t size);

   size_t reserve(size_t size); void getReservedSize() const;

   void adoptBuffer(XmlData &src);
};
```

The `XmlData` class encapsulates a buffer for storing and retrieving binary data (uninterpreted bytes). The default and copy constructors for `XmlData` manage their own memory and the application need not be aware of it. The constructors, `XmlData(const Dbt &dbt)` and `XmlData(void *data, size_t size)`, create "wrapper" objects for the memory passed in and in those instances it is up to the application to own and manage the memory. If a wrapper object is assigned data that is larger than its memory buffer an exception will be thrown.

XmlData Methods

XmlData Methods	Description
XmlData::get_data()	Get the data buffer
XmlData::set()	Set data to the buffer
XmlData::append()	Append data to the buffer
XmlData::get_size()	Get the size of the data buffer
XmlData::set_size()	Sets the size of the data buffer
XmlData::reserve()	Sets the minimum size of the buffer
XmlData::getReservedSize()	Get the size of the data buffer
XmlData::adoptBuffer()	Take ownership of the data buffer

XmlData::get_data()

```
#include <DbXml.hpp>

void *XmlData::get_data()
```

Returns the data buffer.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::set()

```
#include <DbXml.hpp>

void XmlData::set(void *data, size_t size)
```

Copies `size` bytes from `data` to the start of the underlying buffer, which will expand to fit the data if it is not a wrapper, otherwise an exception will be thrown. This method will change the size of the data.

Parameters

data

The data to be copied into the underlying buffer.

size

The number of bytes to copy.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::append()

```
#include <DbXml.hpp>

void XmlData::append(void *data, size_t size)
```

Copies `size` bytes from `data` to the end of the existing data in the underlying buffer, which will expand to fit the data if it is not a wrapper, otherwise an exception will be thrown. This method will change the size of the data.

Parameters

data

A pointer to the data to be copied.

size

The number of bytes to copy.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::get_size()

```
#include <DbXml.hpp>

size_t XmlData::get_size()
```

Returns the size of the real data in the buffer.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::set_size()

```
#include <DbXml.hpp>

void XmlData::set_size(size_t size)
```

Sets the size of the data held in the buffer. An exception will be thrown if the size is larger than the buffer.

Parameters

size

Specifies the size of the data in the buffer.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::reserve()

```
#include <DbXml.hpp>

void XmlData::reserve(size_t size)
```

Ensures that the underlying buffer has at least `size` bytes, starting at offset 0. Existing data is not affected. If buffer expansion is needed and the object is a wrapper an exception is thrown.

Parameters

size

The number of bytes to reserve.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::getReservedSize()

```
#include <DbXml.hpp>

size_t XmlData::getReservedSize()
```

Returns the size of the underlying buffer. This is at least the size of the actual data stored.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

XmlData::adoptBuffer()

```
#include <DbXml.hpp>

void XmlData::adoptBuffer(XmlData &src)
```

Takes ownership of the buffer in `src`, leaving `src` with an empty buffer.

Parameters

src

The `XmlData` object from which to take the buffer.

Class

XmlData (page 118)

See Also

XmlData Methods (page 119)

Chapter 8. XmlDebugListener

```
#include <DbXml.hpp>

class DbXml::XmlDebugListener {
public:
    virtual ~XmlDebugListener()
    virtual void start(const XmlStackFrame *stack)
    virtual void end(const XmlStackFrame *stack)
    virtual void enter(const XmlStackFrame *stack)
    virtual void exit(const XmlStackFrame *stack)
    virtual void error(const XmlException &error,
                       const XmlStackFrame *stack)
};
```

The `XmlDebugListener` class allows the user to track the progress of a query as it executes. The XmlStackFrame (page 395) argument to the methods of this class gives access to the point in the query plan corresponding to the current execution state, as well as the execution stack trace and parts of the dynamic context for that stack frame.

During evaluation of a query, BDB XML will evaluate the sub-expressions of the query. The XmlDebugListener::start() (page 130) method is called when evaluation of a sub-expression starts and the XmlDebugListener::end() (page 131) method is called when it ends. This can occur more than once for the same sub-expression if the expression is in a loop or in a function that is called more than once.

When evaluating a sub-expression BDB XML calls into that sub-expression a number of times to retrieve parts of its result. For eager evaluation, this will happen only once, but for lazy evalution this will happen once per item in the result. The XmlDebugListener::enter() (page 132) method is called when BDB XML requests results from a sub-expression and the XmlDebugListener::exit() (page 133) method is called when the results requested have been calculated.

XmlDebugListener Methods

XmlDebugListener Methods	Description
XmlDebugListener::start()	Start evaluation of a sub-expression
XmlDebugListener::end()	End evaluation of a sub-expression
XmlDebugListener::enter()	Request results from a sub-expression
XmlDebugListener::exit()	Calculation for a requested result has finished
XmlDebugListener::error()	An error occurred during query evaluation

XmlDebugListener::start()

```
#include <DbXml.hpp>

void XmlDebugListener::start(const XmlStackFrame *stack))
```

This method is called when BDB XML starts evaluation of a sub-expression.

Parameters

stack

The XmlStackFrame (page 395) for the current execution context.

Class

XmlDebugListener (page 128)

See Also

XmlDebugListener Methods (page 129)

XmlDebugListener::end()

```
#include <DbXml.hpp>

void XmlDebugListener::end(const XmlStackFrame *stack))
```

This method is called when BDB XML ends evaluation of a sub-expression.

Parameters

stack

The XmlStackFrame (page 395) for the current execution context.

Class

XmlDebugListener (page 128)

See Also

XmlDebugListener Methods (page 129)

XmlDebugListener::enter()

```
#include <DbXml.hpp>

void XmlDebugListener::enter(const XmlStackFrame *stack))
```

This method is called when BDB XML requests results from a sub-expression.

Parameters

stack

The XmlStackFrame (page 395) for the current execution context.

Class

XmlDebugListener (page 128)

See Also

XmlDebugListener Methods (page 129)

XmlDebugListener::exit()

```
#include <DbXml.hpp>

void XmlDebugListener::exit(const XmlStackFrame *stack))
```

This method is called when a sub-expression has finished calculating the results requested.

Parameters

stack

The XmlStackFrame (page 395) for the current execution context.

Class

XmlDebugListener (page 128)

See Also

XmlDebugListener Methods (page 129)

XmlDebugListener::error()

```
#include <DbXml.hpp>

void XmlDebugListener::error(const XmlException &error,
                             const XmlStackFrame *stack)
```

This method is called if an error occurs during query evaluation. It is normal to throw the XmlException (page 206) argument at the end of an implementation of this method.

Parameters

error

The XmlException (page 206) representing the error.

stack

The XmlStackFrame (page 395) for the current execution context.

Class

XmlDebugListener (page 128)

See Also

XmlDebugListener Methods (page 129)

Chapter 9. XmlDocument

```
#include <DbXml.hpp>

class DbXml::XmlDocument {
public:
 XmlDocument();
 XmlDocument(const XmlDocument &);
 ~XmlDocument();
 XmlDocument &operator = (const XmlDocument &)
 ...
};
```

An `XmlDocument` is the unit of storage within an XmlContainer (page 16). A document consists of content, a name, and a set of metadata attributes.

The document content is a byte stream. It must be well formed XML, but need not be valid.

The document name is a unique identifier for the document. The name is specified when the document is first placed in the container. It can either be explicitly specified by the user, or it can be auto-generated by Berkeley DB XML. See XmlContainer::putDocument (page 57) for details.

The user can retrieve the document by name using XmlContainer::getDocument (page 38). In addition, the document name can be referenced in an XQuery expression using the doc() navigation function. For example, suppose your document's name is 'doc1.xml' and the container that it exists in is 'container1.bdbxml'. In this case, you can explicitly request the document by it's name using:

```
doc('dbxml:/container1.bdbxml/doc1.xml')
```

The metadata attributes provide a means of associating information with a document, without having to store it within the document itself. Example metadata attributes might be: document owner, creation time, receipt time, originating source, final destination, and next processing phase. They are analogous to the attributes of a file in a file system. Each metadata attribute consists of a name-value pair. The document's name is an implicit metadata attribute.

You can access the metadata for a given document by using either XmlDocument::getMetaData (page 142) or by iterating over the document's metadata. Use XmlDocument::getMetaDataIterator (page 143) to retrieve an XmlMetaDataIterator (page 337) object.

You can instantiate an empty `XmlDocument` object using XmlManager::createDocument (page 286). The copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body. This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlDocument Methods

XmlDocument Methods	Description
XmlDocument::fetchAllData	Retrieve all document content and metadata.
XmlDocument::getContent	Retrieve content.
XmlDocument::getContentAsEventReader	Retrieve content as an XmlEventReader.
XmlDocument::getContentAsEventWriter	Retrieve content into an XmlEventWriter.
XmlDocument::getContentAsXmlInputStream	Retrieve content as an input stream.
XmlDocument::getMetaData	Retrieve a single metadata value.
XmlDocument::getMetaDataIterator	Get an XmlMetaDataIterator.
XmlDocument::getName	Get the document's name.
XmlDocument::removeMetaData	Removes a single metadata value.
XmlDocument::setContent	Set the document's content.
XmlDocument::setContentAsEventReader	Set the document's content from an XmlEventReader.
XmlDocument::setContentAsXmlInputStream	Set the document's content from an input stream.
XmlDocument::setMetaData	Set a metadata value for the document.
XmlDocument::setName	Set the document's name.

XmlDocument::fetchAllData

```
#include <DbXml.hpp>

void XmlDocument::fetchAllData()
```

If a document was retrieved using DBXML_LAZY_DOCS, then document content and metadata is only retrieved from the container on an as-needed basis. This method causes all document data and metadata to be retrieved. Note that documents in node storage containers are implicitly lazy.

Note that if DBXML_LAZY_DOCS was not used to retrieve the document, then use of this method has no significant performance impact. However, if the document was retrieved lazily, then repeatedly calling this method on any given document may hurt your application's performance. This is because each time this method is called, Berkeley DB XML must walk the entire document tree in order to ensure that it has retrieved the entire document.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getContent

```
#include <DbXml.hpp>

std::string &XmlDocument::getContent(std::string &content) const

XmlData XmlDocument::getContent() const
```

Returns a reference to the document content. The returned value is owned by the `XmlDocument`, and is destroyed when the document is destroyed.

There are two forms to this method. The first copies the content of the document into a string and as a convenience returns a reference to the string.

The second form of this method returns a reference to the content as an XmlData (page 118) object.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getContentAsEventReader

```
#include <DbXml.hpp>

XmlEventReader &XmlDocument::getContentAsEventReader() const
```

Returns an XmlEventReader (page 152) object that can be used to obtain the document content as a series of events. When the caller is done with the event reader, the XmlEventReader::close (page 156) method must be called to release its resources.

If the document comes from a container of type `XmlContainer::WholedocContainer`, it will be parsed in order to provide the event stream. If the document was obtained inside of a transaction, its events must be read while still inside the transaction.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getContentAsEventWriter

```
#include <DbXml.hpp>

void
XmlDocument::getContentAsEventWriter(XmlEventWriter &writer) const
```

Causes the document's content to be written as events to the provided XmlEventWriter (page 192).

If the document comes from a container of type `XmlContainer::WholedocContainer`, it will be parsed in order to provide the event stream.

Parameters

writer

The event writer to which the content events are written. When the event writing is complete, the XmlEventWriter::close (page 194) method is called. To create the writer, use one of XmlContainer::putDocumentAsEventWriter (page 63), or implement an application-defined class derived from XmlEventWriter (page 192).

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getContentAsXmlInputStream

```
#include <DbXml.hpp>

XmlInputStream *XmlDocument::getContentAsXmlInputStream() const
```

Returns the document's content as an XmlInputStream (page 270). The returned value is owned by the caller, and must be explicitly deleted.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getMetaData

```
#include <DbXml.hpp>

bool XmlDocument::getMetaData(const std::string &uri,
    const std::string &name, XmlValue &value);

bool XmlDocument::getMetaData(const std::string &uri,
    const std::string &name, XmlData &value) const;
```

Returns the value of the specified metadata. The value of the metadata attribute can be retrieved as a typed or untyped value. Typed values are retrieved by passing an `XmlValue` to the API. Untyped values are retrieved by passing an XmlData (page 118) object (Dbt) through the API.

This method returns true if metadata is found for the `XmlDocument` that matches the given URI and name; otherwise, it returns false.

Parameters

uri

The namespace within which the name resides. The empty string refers to the default namespace.

name

The name of the metadata attribute.

value

The XmlValue (page 416) or XmlData (page 118) object in which the metadata value is to be placed.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getMetaDataIterator

```
#include <DbXml.hpp>
```

Returns an XmlMetaDataIterator (page 337). Using this iterator, you can examine the individual metadata items set for the document by looping over them using XmlMetaDataIterator::next (page 339).

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::getName

```
#include <DbXml.hpp>

std::string XmlDocument::getName() const;
```

The `XmlDocument::getName` method returns the `XmlDocument` name.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::removeMetaData

```
#include <DbXml.hpp>

void XmlDocument::removeMetaData(const std::string &uri,
    const std::string &name)
```

Removes the identified metadata from the document.

Parameters

uri

The namespace within which the name resides. The empty string refers to the default namespace.

name

The name of the metadata attribute to be removed.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::setContent

```
#include <DbXml.hpp>

void XmlDocument::setContent(const std::string &content)

void XmlDocument::setContent(const XmlData &content)
```

Sets the document's content to the provided content. If this document is a new document (that is, its name is currently not in use by another document in the container), you can add it to a container using XmlContainer::putDocument (page 57). If you are updating an already existing document, you can update the document in the container using XmlContainer::updateDocument (page 74).

Parameters

content

The string or XmlData (page 118) object containing the new document contents. Note that the document contents must be well-formed XML. However, in the event of incorrect content, an exception is not thrown until an attempt is made to place the contents into a container using either XmlContainer::putDocument (page 57) or XmlContainer::updateDocument (page 74).

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::setContentAsEventReader

```
#include <DbXml.hpp>

void XmlDocument::setContentAsEventReader(XmlEventReader &reader)
```

Sets the document's content to the provided reader. If this document is a new document (that is, its name is currently not in use by another document in the container), you can add it to a container using XmlContainer::putDocument (page 57). If you are updating an already existing document, you can update the document in the container using XmlContainer::updateDocument (page 74).

The XmlEventReader (page 152) reference is used to read the content on demand. When the reading is done, the XmlEventReader::close (page 156) method is called.

The content provided by the reader must be well-formed XML. However, in the event of incorrect content, an exception is not thrown until an attempt is made to place the contents into a container using either XmlContainer::putDocument (page 57) or XmlContainer::updateDocument (page 74).

Parameters

reader

The event reader to be used for content creation. To create the reader, use one of XmlDocument::getContentAsEventReader (page 139), XmlValue::asEventReader (page 422), or implement an application-defined class derived from XmlEventReader (page 152).

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::setContentAsXmlInputStream

```
#include <DbXml.hpp>

void
XmlDocument::setContentAsXmlInputStream(XmlInputStream *adopted_str)
```

Sets the document's content to the provided content. If this document is a new document (that is, its name is currently not in use by another document in the container), you can add it to a container using XmlContainer::putDocument (page 57). If you are updating an already existing document, you can update the document in the container using XmlContainer::updateDocument (page 74).

Note that the document contents must be well-formed XML. However, in the event of incorrect content, an exception is not thrown until an attempt is made to place the contents into a container using either XmlContainer::putDocument (page 57) or XmlContainer::updateDocument (page 74). After this call, the adopted stream is owned by the document, which will delete the object.

Parameters

adopted_str

The input stream that points to the well-formed XML to be used as this document's content. To create the input stream, use one of XmlManager::createLocalFileInputStream (page 290), XmlManager::createMemBufInputStream (page 291), XmlManager::createStdInInputStream (page 295), XmlManager::createURLInputStream (page 300), or XmlDocument::getContentAsXmlInputStream (page 141).

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::setMetaData

```
#include <DbXml.hpp>

void XmlDocument::setMetaData(
    const std::string &uri,
    const std::string &name,
    const XmlValue &value);

void XmlDocument::setMetaData(
    const std::string &uri,
    const std::string &name,
    const XmlData &value);
```

Sets the value of the specified metadata attribute. A metadata attribute is a name-value pair, which is stored with the document, but not as part of the document content. The value of a metadata attribute may be typed or untyped.

A metadata attribute name consists of a namespace URI and a name. The namespace URI is optional but it should be used to avoid naming collisions.

Typed values are passed to the API as an instance of XmlValue (page 416), and may be of type Number, String, or Boolean.

The metadata attribute can be queried using an XQuery query that makes use of the special dbxml:metadata() function from within a predicate. To make use of this function, you must define a namespace for use with the query (it can be any random namespace). For example, suppose you had metadata whose name was "dateStamp". Then to query for documents that have a specific dateStamp value:

```
myQueryContext.setNamespace("ds", "http://randomNS/"); std::string
myQuery="/*dbxml:metadata('ds:dateStamp')='10/30/2004'";
```

Untyped values are passed to the API as a XmlData (page 118).

If a given metadata attribute is indexed, it is possible to use XmlContainer::lookupIndex (page 49) to perform fast lookup.

Parameters

uri

The namespace within which the name resides. The empty string refers to the default namespace.

name

The name of the metadata attribute.

value

The XmlValue (page 416) or XmlData (page 118) to be used for the metadata value.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

XmlDocument::setName

```
#include <DbXml.hpp>

void XmlDocument::setName(const std::string &name);
```

The `XmlDocument::setName` method sets the name of the document. Note that when the document is put in a container, either the name that you specify must be unique, or you must use the DBXML_GEN_NAME flag, or an exception is thrown.

Parameters

name

A string containing the name to be assigned to the `XmlDocument`.

Class

XmlDocument (page 135)

See Also

XmlDocument Methods (page 136)

Chapter 10. XmlEventReader

```
#include <DbXml.hpp>

virtual XmlEventReader::~XmlEventReader()
```

The `XmlEventReader` class enables applications to read document content via a pull interface without materializing XML as text. This can be efficient and allow closer integration of XML processing in an application.

The `XmlEventReader` acts as an iterator, where XmlEventReader::hasNext (page 176) indicates the presence of additional events, and XmlEventReader::next (page 182) moves the current location, returning the event type of the next event. At any given location, various methods on the object allow the application to retrieve the current state, such as element name and attributes. Character state (names, text values, etc) are returned in NULL-terminated const unsigned char * strings, encoded in UTF-8. Their values are valid only until another call is made on the `XmlEventReader` object. When processing of the object is completed, the XmlEventReader::close (page 156) method must be called to release resources. Some interfaces implicitly assume ownership of the object -- for example XmlDocument::setContentAsEventReader (page 147).

`XmlEventReader` does not include events for attributes. Attributes are retrieved via interfaces such as XmlEventReader::getAttributeLocalName (page 159) when the event type is StartElement.

`XmlEventReader` skips the EndElement event for empty elements (where XmlEventReader::isEmptyElement (page 178) returns true).

Event types are defined at global scope, and include:

- `XmlEventReader::StartElement`

  The current event is the start of an element.

- `XmlEventReader::EndElement`

  The current event is the end of an element.

- `XmlEventReader::Characters`

  The current event is text characters.

- `XmlEventReader::CDATA`

  The current event is CDATA text.

- `XmlEventReader::Comment`

  The current event is comment text.

- `XmlEventReader::Whitespace`

The current event is ignorable whitespace.

- `XmlEventReader::StartDocument`

  The current event is the start of the document.

- `XmlEventReader::EndDocument`

  The current event is the end of the document.

- `XmlEventReader::StartEntityReference`

  The current event marks the start of expanded entity text.

- `XmlEventReader::EndEntityReference`

  The current event marks the end of expanded entity text.

- `XmlEventReader::ProcessingInstruction`

  The current event is a processing instruction.

- `XmlEventReader::DTD`

  The current event is the text of a DTD.

Many of the class methods are context-dependent, as they are meaningful only within the context of a given event. See the documentation for the individual methods for details.

An `XmlEventReader` object may be obtained via XmlDocument::getContentAsEventReader (page 139) or XmlValue::asEventReader (page 422) or from an application-written class derived from `XmlEventReader`. This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlEventReader Methods

XmlEventReader Methods	Description
XmlEventReader::close	Release resources for reader.
XmlEventReader::encodingSet	Checks if the encoding of the document is set.
XmlEventReader::getAttributeCount	Return the number of attributes for an element event.
XmlEventReader::getAttributeLocalName	Get attribute's local name.
XmlEventReader::getAttributeNamespaceURI	Get attribute's namespace URI.
XmlEventReader::getAttributePrefix	Get attribute's namespace prefix.
XmlEventReader::getAttributeValue	Get attribute's value.
XmlEventReader::getEncoding	Return the encoding of the document.
XmlEventReader::getEventType	Return the type of the current event.
XmlEventReader::getExpandEntities	Get whether to expand entities.
XmlEventReader::getLocalName	Return the name of the current element event.
XmlEventReader::getNamespaceURI	Return the namespace URI of the current element event.
XmlEventReader::getPrefix	Return the namespace prefix for the current element event.
XmlEventReader::getReportEntityInfo	Get whether to report entity information.
XmlEventReader::getSystemId	Return the System ID for for the document.
XmlEventReader::getValue	Return the value of the current event.
XmlEventReader::getVersion	Return the XML version string for the document.
XmlEventReader::hasEmptyElementInfo	Check if object will return empty element state.
XmlEventReader::hasEntityEscapeInfo	Check if object has information about entities.
XmlEventReader::hasNext	Check if there are more events.
XmlEventReader::isAttributeSpecified	Check if an attribute is specified.
XmlEventReader::isEmptyElement	Check current element is empty.
XmlEventReader::isStandalone	Check if document is standalone XML.
XmlEventReader::isWhiteSpace	Check if current text value is white space.
XmlEventReader::needsEntityEscape	Check if text or attribute value needs escaping.
XmlEventReader::next	Move to the next event.

XmlEventReader Methods	Description
XmlEventReader::nextTag	Move to the next StartElement or EndElement event.
XmlEventReader::setExpandEntities	Set whether to expand entities.
XmlEventReader::setReportEntityInfo	Set whether to report entity information.
XmlEventReader::standaloneSet	Check if the standalone attribute is set.

XmlEventReader::close

```
#include <DbXml.hpp>

virtual void XmlEventReader::close()
```

Indicate that the application is done process the XmlEventReader (page 152) and release associated resources. The object must not be referenced after this method is called.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::encodingSet

```
#include <DbXml.hpp>

virtual bool XmlEventReader::encodingSet() const
```

Checks if the encoding was explicitly set. This method is valid only if the event type is `StartDocument`.

If the event type is not StartDocument, `XmlException::EVENT_ERROR` is thrown.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getAttributeCount

```
#include <DbXml.hpp>

virtual int XmlEventReader::getAttributeCount() const
```

If the current event is StartElement, return the number of attributes available.

Errors

The `XmlEventReader::getAttributeCount` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getAttributeLocalName

```
#include <DbXml.hpp>

virtual const unsigned char *
XmlEventReader::getAttributeLocalName(int index) const
```

If the current event is StartElement, return the local name for the attribute at the specified offset.

Parameters

index

The index into the attribute list.

Errors

The `XmlEventReader::getAttributeLocalName` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getAttributeNamespaceURI

```
#include <DbXml.hpp>

virtual const unsigned char *
XmlEventReader::getAttributeNamespaceURI(int index) const
```

If the current event is StartElement, return the namespace URI for the attribute at the specified offset.

Parameters

index

The index into the attribute list.

Errors

The `XmlEventReader::getAttributeNamespaceURI` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getAttributePrefix

```
#include <DbXml.hpp>

virtual const unsigned char *
XmlEventReader::getAttributePrefix(int index) const
```

If the current event is StartElement, return the namespace prefix for the attribute at the specified offset.

Parameters

index

The index into the attribute list.

Errors

The `XmlEventReader::getAttributePrefix` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getAttributeValue

```
#include <DbXml.hpp>

virtual const unsigned char *
XmlEventReader::getAttributeValue(int index) const
```

If the current event is StartElement, return the value for the attribute at the specified offset.

Parameters

index

The index into the attribute list.

Errors

The `XmlEventReader::getAttributeValue` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getEncoding

```
#include <DbXml.hpp>

virtual const unsigned char * XmlEventReader::getEncoding() const
```

Returns the encoding for the document, if available.

If the event type is not `StartDocument`, `XmlException::EVENT_ERROR` is thrown.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getEventType

```
#include <DbXml.hpp>

virtual enum XmlEventType XmlEventReader::getEventType() const
```

Return the event type of the current XmlEventReader (page 152) event.

Valid events include:

- `XmlEventReader::StartElement`

  The current event is the start of an element.

- `XmlEventReader::EndElement`

  The current event is the end of an element.

- `XmlEventReader::Characters`

  The current event is text characters.

- `XmlEventReader::CDATA`

  The current event is CDATA text.

- `XmlEventReader::Comment`

  The current event is comment text.

- `XmlEventReader::Whitespace`

  The current event is ignorable whitespace.

- `XmlEventReader::StartDocument`

  The current event is the start of the document.

- `XmlEventReader::EndDocument`

  The current event is the end of the document.

- `XmlEventReader::StartEntityReference`

  The current event marks the start of expanded entity text.

- `XmlEventReader::EndEntityReference`

  The current event marks the end of expanded entity text.

- `XmlEventReader::ProcessingInstruction`

  The current event is a processing instruction.

- `XmlEventReader::DTD`

  The current event is the text of a DTD.

Errors

The `XmlEventReader::getEventType` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getExpandEntities

```
#include <DbXml.hpp>

virtual bool XmlEventReader::getExpandEntities() const
```

Indicates whether entites are expanded when XML is parsed. You can change the value of this setting by using the XmlEventReader::setExpandEntities (page 185) method.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getLocalName

```
#include <DbXml.hpp>

virtual const unsigned char * XmlEventReader::getLocalName() const
```

If the current event is StartElement or EndElement, return the local name for the element. If the current event is ProcessingInstruction, return the target portion of the processing instruction.

Errors

The `XmlEventReader::getLocalName` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getNamespaceURI

```
#include <DbXml.hpp>

virtual const unsigned char *
XmlEventReader::getNamespaceURI() const
```

If the current event is StartElement or EndElement, return the namespace URI for the element.

Errors

The `XmlEventReader::getNamespaceURI` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getPrefix

```
#include <DbXml.hpp>

virtual const unsigned char * XmlEventReader::getPrefix() const
```

If the current event is StartElement or EndElement, return the namespace prefix for the element.

Errors

The `XmlEventReader::getPrefix` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getReportEntityInfo

```
#include <DbXml.hpp>

virtual bool XmlEventReader::getReportEntityInfo() const
```

Indicates whether the XmlEventReader (page 152) will include include events of type StartEntityReference and EndEntityReference, when possible. This value can be set using the XmlEventReader::setReportEntityInfo (page 186) method.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getSystemId

```
#include <DbXml.hpp>

virtual const unsigned char * XmlEventReader::getSystemId() const
```

Returns the document's system ID, if available.

If the event type is not `StartDocument`, `XmlException::EVENT_ERROR` is thrown.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getValue

```
#include <DbXml.hpp>

virtual const unsigned char *
XmlEventReader::getValue(int &len) const
```

If the current event is Characters, return the text value. If the current event is ProcessingInstruction, return the data portion of the processing instruction.

Parameters

len

The length of the value string returned, not including the null terminator.

Errors

The `XmlEventReader::getValue` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::getVersion

```
#include <DbXml.hpp>

virtual const unsigned char * XmlEventReader::getVersion() const
```

Returns the XML version string, if available.

If the event type is not `StartDocument`, `XmlException::EVENT_ERROR` is thrown.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::hasEmptyElementInfo

```
#include <DbXml.hpp>

virtual bool XmlEventReader::hasEmptyElementInfo() const
```

Returns true if the XmlEventReader (page 152) object will return whether an element is empty or not in the context of the StartElement event. If true, and an element is empty (has no content), there will be no corresponding EndElement event for the element.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::hasEntityEscapeInfo

```
#include <DbXml.hpp>

virtual bool XmlEventReader::hasEntityEscapeInfo() const
```

Returns true if the XmlEventReader (page 152) object is able to return information about text strings indicating that they may have entities requiring escaping for XML serialization. This allows applications performing serialization to avoid scanning strings for entities if it is not necessary. Most of the internal implementations of XmlEventReader (page 152) have this information available, and return true.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::hasNext

```
#include <DbXml.hpp>

virtual bool XmlEventReader::hasNext() const
```

Check if there are additional events available in the XmlEventReader (page 152) event stream.

Errors

The `XmlEventReader::hasNext` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::isAttributeSpecified

```
#include <DbXml.hpp>

virtual bool XmlEventReader::isAttributeSpecified(int index) const
```

If the current event is StartElement, return whether the attribute at the index indicated is specified.

Parameters

index

The index into the attribute list.

Errors

The `XmlEventReader::isAttributeSpecified` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::isEmptyElement

```
#include <DbXml.hpp>

virtual bool XmlEventReader::isEmptyElement() const
```

Return true if the current event is StartElement and the element has not content. If the current even is not StartElement, an exception will be thrown.

Errors

The `XmlEventReader::isEmptyElement` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::isStandalone

```
#include <DbXml.hpp>

virtual bool XmlEventReader::isStandalone() const
```

Return whether the document is standalone.

If the event type is not `StartDocument`, `XmlException::EVENT_ERROR` is thrown.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::isWhiteSpace

```
#include <DbXml.hpp>

virtual bool XmlEventReader::isWhiteSpace() const
```

Return true if the current text value is entirely white space. The current event must be one of Whitespace, Characters, or CDATA, or an exception will be thrown.

Errors

The `XmlEventReader::isWhiteSpace` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::needsEntityEscape

```
#include <DbXml.hpp>

virtual bool
XmlEventReader::needsEntityEscape(int index = 0) const
```

If the current event is Characters, and XmlEventReader::hasEntityEscapeInfo (page 175) is true, returns whether the current text string requires escaping of entities for XML serialization.

If the current event is StartElement, and XmlEventReader::hasEntityEscapeInfo (page 175) is true, returns whether the attribute value specified by the index parameter requires escaping of entities for XML serialization.

Parameters

index

If the current event is StartElement, index is the attribute index used to specify an attribute. If the current event is Characters, it is ignored.

Errors

The `XmlEventReader::needsEntityEscape` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::next

```
#include <DbXml.hpp>

virtual enum XmlEventType XmlEventReader::next()
```

Move to the next event in the XmlEventReader (page 152) object, returning the type of the event. If the current event is StartElement and the element is empty (XmlEventReader::isEmptyElement (page 178) returns true) the corresponding EndElement event is skipped.

Valid events include:

- `XmlEventReader::StartElement`

  The current event is the start of an element.

- `XmlEventReader::EndElement`

  The current event is the end of an element.

- `XmlEventReader::Characters`

  The current event is text characters.

- `XmlEventReader::CDATA`

  The current event is CDATA text.

- `XmlEventReader::Comment`

  The current event is comment text.

- `XmlEventReader::Whitespace`

  The current event is ignorable whitespace.

- `XmlEventReader::StartDocument`

  The current event is the start of the document.

- `XmlEventReader::EndDocument`

  The current event is the end of the document.

- `XmlEventReader::StartEntityReference`

  The current event marks the start of expanded entity text.

- `XmlEventReader::EndEntityReference`

  The current event marks the end of expanded entity text.

- `XmlEventReader::ProcessingInstruction`

  The current event is a processing instruction.

- `XmlEventReader::DTD`

  The current event is the text of a DTD.

Errors

The `XmlEventReader::next` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::nextTag

```
#include <DbXml.hpp>

virtual enum XmlEventType XmlEventReader::nextTag()
```

Move to the next StartElement or EndElement in the XmlEventReader (page 152) object, skipping the events in between, returning the type of the event. If the current event is StartElement and the element is empty (XmlEventReader::isEmptyElement (page 178) returns true) the corresponding EndElement event is skipped.

Errors

The `XmlEventReader::nextTag` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventReader (page 152) object. Most likely the error is requesting state that is not valid in the context of the current event.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::setExpandEntities

```
#include <DbXml.hpp>

virtual void XmlEventReader::setExpandEntities(bool value)
```

By default, entities are expanded when XML is parsed, and those entities are reported as their expanded events. If the XmlEventReader::getReportEntityInfo (page 170) method returns true, it is possible to suppress the expanded events, receiving just the StartEntityReference and EndEntityReference events associated with the original entity reference. This can be useful for serialization of XML that includes such expanded entities. It allows the entity references to be restored during serialization. Most of the internal implementations of XmlEventReader (page 152) have this Capability. By default, expanded events are reported.

Parameters

value

Indicates whether entities are expanded. A value of `true` (the default) causes the entity to be expanded.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::setReportEntityInfo

```
#include <DbXml.hpp>

virtual void XmlEventReader::setReportEntityInfo(bool value)
```

The events of type StartEntityReference and EndEntityReference are used to report the start and end of XML that was originally an entity reference in the XML text, but has since been expanded. These events, if available, are useful for serialization of XML that includes such expanded entities. It allows the entity references to be restored during serialization. Most of the internal implementations of XmlEventReader (page 152) have this information available. By default, these events are not reported.

Parameters

value

A value of `true` causes the XmlEventReader (page 152) to include events of type StartEntityReference and EndEntityReference, if it is possible.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

XmlEventReader::standaloneSet

```
#include <DbXml.hpp>

virtual bool XmlEventReader::standaloneSet() const
```

Checks if the standalone attribute was explicitly set.

If the event type is not `StartDocument`, `XmlException::EVENT_ERROR` is thrown.

Class

XmlEventReader (page 152)

See Also

XmlEventReader Methods (page 154)

Chapter 11. XmlEventReaderToWriter

```
#include <DbXml.hpp>

class DbXml::XmlEventReaderToWriter {
public:
 XmlEventReaderToWriter(XmlEventReader &reader,
                              XmlEventWriter &writer,
                              bool ownsReader = true)
 XmlEventReaderToWriter(XmlEventReader &reader,
                              XmlEventWriter &writer,
                              bool ownsReader,
          bool ownsWriter)
 ...
};
```

The `XmlEventReaderToWriter` class enables events read from an XmlEventReader (page 152) to be written directly to an XmlEventWriter (page 192). This is useful for processing XML document content, efficient copying, and other application integration tasks. Use XmlEventReaderToWriter::start (page 191) to begin processing.

The method constructs an object from the reader and writer that will pipe events from the reader directly to the writer.

This object is not thread-safe, and can only be safely used by one thread at a time in an application.

Parameters

reader

An XmlEventReader (page 152) object from which events will be read. To create the reader, use one of XmlDocument::getContentAsEventReader (page 139), XmlValue::asEventReader (page 422), or implement an application-defined class derived from XmlEventReader (page 152).

writer

An XmlEventWriter (page 192) object to which events will be written. When all events are processed, XmlEventWriter::close (page 194) is called on the object unless the **ownsWriter** parameter is `false`. To create the writer, use XmlContainer::putDocumentAsEventWriter (page 63), or implement an application-defined class derived from XmlEventWriter (page 192).

ownsReader

If `true` (the default), XmlEventReader::close (page 156) will be called when all events have been processed; otherwise, the reader is left intact upon completion.

ownsWriter

If `true` (the default), XmlEventReader::close (page 156) will be called when all events have been processed; otherwise, the writer is left intact upon completion. Setting this parameter to false allows applications to apply a number of XmlEventReader (page 152) objects to the same XmlEventWriter (page 192).

XmlEventReaderToWriter Methods

XmlEventReaderToWriter Methods	Description
XmlEventReaderToWriter::start	Begin processing events.

XmlEventReaderToWriter::start

```
#include <DbXml.hpp>

void XmlEventReaderToWriter::start()
};
```

Start processing events for the XmlEventReaderToWriter (page 188) instance, and continue until there are no events left to process from the XmlEventReader (page 152). All events are written directly to the XmlEventWriter (page 192) which is part of the object.

Class

XmlEventReaderToWriter (page 188)

See Also

XmlEventReaderToWriter Methods (page 190)

Chapter 12. XmlEventWriter

```
#include <DbXml.hpp>

virtual XmlEventWriter::~XmlEventWriter()
```

The `XmlEventWriter` class enables applications to construct document content without using serialized XML. This allows use of application-provided parsers and other XML processing mechanisms. The `XmlEventWriter` is a push-style interface, with content written via explicit interfaces.

The `XmlEventWriter` maintains state based on information written, and an exception is thrown if an operation is performed that is not valid for the current state. All strings must be null-terminated const unsigned char * values encoded in UTF-8. The values are copied by the underlying implementation, as necessary. When a document is completed, the XmlEventReader::close (page 156) method must be called to finalize the operation and release resources.

XmlContainer::putDocumentAsEventWriter (page 63) can be used to obtain an instance of `XmlEventWriter` that is used to create document content in Berkeley DB XML. XmlEventReaderToWriter (page 188) can be used to attach an XmlEventReader (page 152) directly to an `XmlEventWriter` for copying of content. An `XmlEventWriter` may also be obtained via an application-written class derived from `XmlEventWriter`. This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlEventWriter Methods

XmlEventWriter Methods	Description
XmlEventWriter::close	Release resources for the writer.
XmlEventWriter::writeAttribute	Write an attribute.
XmlEventWriter::writeDTD	Write a DTD or reference.
XmlEventWriter::writeEndDocument	Write an EndDocument event.
XmlEventWriter::writeEndElement	Write an EndElement event.
XmlEventWriter::writeEndEntity	Write an EndEntity event.
XmlEventWriter::writeProcessingInstruction	Write a ProcessingInstruction.
XmlEventWriter::writeStartDocument	Write a StartDocument event.
XmlEventWriter::writeStartElement	Write a StartElement event.
XmlEventWriter::writeStartEntity	Write a StartEntityReference event.
XmlEventWriter::writeText	Write a Text event.

XmlEventWriter::close

```
#include <DbXml.hpp>

virtual void XmlEventWriter::close()
```

Indicate that the application is done process the XmlEventWriter (page 192) and release associated resources. The object must not be referenced after this method is called.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeAttribute

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeAttribute(const unsigned char *localName,
    const unsigned char *prefix, const unsigned char *uri,
    const unsigned char *value, bool isSpecified)
```

Write a single attribute to the XmlEventWriter (page 192). Namespace declarations are written as normal attributes.

Parameters

localName

Local name of the attribute.

prefix

Namespace prefix, or NULL.

uri

Namespace uri, or NULL.

value

The attribute's value.

isSpecified

True if the attribute is specified, rather than defaulted from a DTD or schema.

Errors

The `XmlEventWriter::writeAttribute` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeDTD

```
#include <DbXml.hpp>

virtual void XmlEventWriter::writeDTD(const unsigned char *dtd,
    int length)
```

Write a DTD to the XmlEventWriter (page 192). This can be used to write an internal subset, or a reference to an external DTD.

Parameters

dtd

The string value of the DTD or reference as it would appear in a serialized XML document.

length

The length of the dtd parameter, not including the null terminator.

Errors

The `XmlEventWriter::writeDTD` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeEndDocument

```
#include <DbXml.hpp>

virtual void XmlEventWriter::writeEndDocument()
```

Write an EndDocument event to the XmlEventWriter (page 192). This indicates that the document is complete. The only valid call after this is made is XmlEventWriter::close (page 194).

Errors

The `XmlEventWriter::writeEndDocument` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeEndElement

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeEndElement(const unsigned char *localName,
    const unsigned char *prefix, const unsigned char *uri)
```

Write an EndElement event to the XmlEventWriter (page 192).

Parameters

localName

Local name of the element.

prefix

Namespace prefix, or NULL.

uri

Namespace uri, or NULL.

Errors

The `XmlEventWriter::writeEndElement` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeEndEntity

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeEndEntity(const unsigned char *name)
```

Write an EndEntityReference event to the XmlEventWriter (page 192). This must have been preceded by a call to XmlEventWriter::writeStartEntity (page 204). Writing of StartEntityReference and EndEntityReference events is optional, but helpful for round-tripping of documents. A given implementation of XmlEventWriter (page 192) may safely ignore such events.

Parameters

name

The name of the entity reference.

Errors

The `XmlEventWriter::writeEndEntity` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeProcessingInstruction

```
#include <DbXml.hpp>

virtual void XmlEventWriter::writeProcessingInstruction(
      const unsigned char *target,
      const unsigned char *data)
```

Write a ProcessingInstruction to the XmlEventWriter (page 192).

Parameters

target

The processing instruction target.

data

The processing instruction data.

Errors

The `XmlEventWriter::writeProcessingInstruction` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeStartDocument

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeStartDocument(const unsigned char *version,
    const unsigned char *encoding,
    const unsigned char *standalone)
```

Write a StartDocument element event to the XmlEventWriter (page 192). This must be the first event written.

Parameters

version

The XML version string, or NULL.

encoding

The encoding for the document, or NULL. Content must be written as UTF-8; however, the encoding may be used on output, if possible.

standalone

The standalone string, or NULL.

Errors

The `XmlEventWriter::writeStartDocument` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeStartElement

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeStartElement(const unsigned char *localName,
    const unsigned char *prefix, const unsigned char *uri,
    int numAttributes, bool isEmpty)
```

Write an element event to the XmlEventWriter (page 192). If the element is empty, and specified as such using the isEmpty parameter, it must not be followed by an XmlEventWriter::writeEndElement (page 198) call.

Parameters

localName

Local name of the element.

prefix

Namespace prefix, or NULL.

uri

Namespace uri, or NULL.

numAttributes

Number of attributes.

isEmpty

True if the element is empty. If this parameter is false, an EndElement event must be written later using XmlEventWriter::writeEndElement (page 198) if true, XmlEventWriter::writeEndElement (page 198) must not be called for this element.

Errors

The `XmlEventWriter::writeStartElement` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeStartEntity

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeStartEntity(const unsigned char *name
    bool expandedInfoFollows)
```

Write StartEntityReference event to the XmlEventWriter (page 192). This event must be followed, later in the document, by a call to XmlEventWriter::writeEndEntity (page 199). Writing of StartEntityReference and EndEntityReference events is optional, but helpful for round-tripping of documents. A given implementation of XmlEventWriter (page 192) may safely ignore such events.

Parameters

name

The name of the entity reference.

expandedInfoFollows

If the entity reference is expanded, and the expanded events will be written, this parameter should be specified as true. If the entity is not to be expanded, false should be used.

Errors

The `XmlEventWriter::writeStartEntity` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

XmlEventWriter::writeText

```
#include <DbXml.hpp>

virtual void
XmlEventWriter::writeText(XmlEventReader::XmlEventType type,
    const unsigned char *text, int length)
```

Write an text event to the XmlEventWriter (page 192). Valid text event types include `XmlEventReader::Characters`, `XmlEventReader::Whitespace`, `XmlEventReader::CDATA`, and `XmlEventReader::Comment`.

Parameters

type

The type of the event — must be one of `XmlEventReader::Characters`, `XmlEventReader::Whitespace`, `XmlEventReader::CDATA`, or `XmlEventReader::Comment`.

text

The text string value.

length

The length of the string, not including a null terminator.

Errors

The `XmlEventWriter::writeText` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

EVENT_ERROR

An error occurred during processing of an XmlEventWriter (page 192) object. Most likely the error is attempting to write a type or value that is not valid in the current state of the object.

Class

XmlEventWriter (page 192)

See Also

XmlEventWriter Methods (page 193)

Chapter 13. XmlException

```
#include <DbXml.hpp>

class DbXml::XmlException : public std::exception {
public:
 virtual const char *what() const throw();
 ExceptionCode getExceptionCode();
 int getDbErrno();
 ...
};
```

The `XmlException` class represents an error condition that has occurred within the Berkeley DB XML system. The system throws an `XmlException` if an API method call results in an error condition. Methods on this class may be used to get further information regarding the exception. Some of these include XmlException::what (page 216) and XmlException::getExceptionCode (page 211). This object is not thread-safe, and can only be safely used by one thread at a time in an application.

Below is a description of the exceptions codes that may be returned from XmlException::getExceptionCode (page 211):

- `XmlException::CONTAINER_CLOSED`

  An operation was performed that requires the container to be open.

- `XmlException::CONTAINER_EXISTS`

  An attempt was made to create an existing container.

- `XmlException::CONTAINER_NOT_FOUND`

  The specified container was not found.

- `XmlException::CONTAINER_OPEN`

  An operation was performed that requires the container to be closed.

- `XmlException::DATABASE_ERROR`

  An unexpected error was returned from Berkeley DB.

- `XmlException::DOCUMENT_NOT_FOUND`

  A named document was not found.

- `XmlException::EVENT_ERROR`

  An error occurred during the use of the event reader or writer interface.

- `XmlException::INDEXER_PARSER_ERROR`

There was an XML parsing error while attempting to index a document.

- `XmlException::INTERNAL_ERROR`

  An internal error has occurred inside BDB XML.

- `XmlException::INVALID_VALUE`

  An invalid operation was attempted, or an invalid value passed to an operation.

- `XmlException::LAZY_EVALUATION`

  An operation that requires eager evaluation was attempted on a lazily evaluated result set.

- `XmlException::NO_MEMORY_ERROR`

  An attempt to allocate memory failed.

- `XmlException::NULL_POINTER`

  An attempt was made to use a NULL pointer, most likely from a non-C++ API, such as Java or Python.

- `XmlException::OPERATION_INTERRUPTED`

  A query operation was interrupted by the application.

- `XmlException::OPERATION_TIMEOUT`

  A query operation timed out.

- `XmlException::TRANSACTION_ERROR`

  An attempt was made to refer to a transaction that was already committed or aborted.

- `XmlException::UNIQUE_ERROR`

  An index uniqueness constraint was violated.

- `XmlException::UNKNOWN_INDEX`

  The referenced indexing strategy is not known.

- `XmlException::VERSION_MISMATCH`

  The container version and library version are not compatible.

- `XmlException::QUERY_EVALUATION_ERROR`

  An error occurred during evaluation of an XQuery expression.

- `XmlException::QUERY_PARSER_ERROR`

An error occurred attempting to parse an XQuery expression.

XmlException Methods

XmlException Methods	Description
XmlException::getDbErrno	Get DB error number.
XmlException::getExceptionCode	Get ExceptionCode.
XmlException::getQueryColumn	Get column for query error.
XmlException::getQueryFile	Get file for query error.
XmlException::getQueryLine	Get line for query error.
XmlException::what	Get error string.

XmlException::getDbErrno

```
#include <DbXml.hpp>

int XmlException::getDbErrno() const
```

If the exception code is `XmlException::DATABASE_ERROR`, this method returns the Berkeley DB error associated with the exception, otherwise the return value of this method is not meaningful.

Class

XmlException (page 206)

See Also

XmlException Methods (page 209)

XmlException::getExceptionCode

```
#include <DbXml.hpp>

XmlException::ExceptionCode XmlException::getExceptionCode() const
```

Returns the exception code associated with the exception. The code may be one of the following:

- `XmlException::CONTAINER_CLOSED`

  An operation was performed that requires the container to be open.

- `XmlException::CONTAINER_EXISTS`

  An attempt was made to create an existing container.

- `XmlException::CONTAINER_NOT_FOUND`

  The specified container was not found.

- `XmlException::CONTAINER_OPEN`

  An operation was performed that requires the container to be closed.

- `XmlException::DATABASE_ERROR`

  An unexpected error was returned from Berkeley DB.

- `XmlException::DOCUMENT_NOT_FOUND`

  A named document was not found.

- `XmlException::EVENT_ERROR`

  An error occurred during the use of the event reader or writer interface.

- `XmlException::INDEXER_PARSER_ERROR`

  There was an XML parsing error while attempting to index a document.

- `XmlException::INTERNAL_ERROR`

  An internal error has occurred inside BDB XML.

- `XmlException::INVALID_VALUE`

  An invalid operation was attempted, or an invalid value passed to an operation.

- `XmlException::LAZY_EVALUATION`

  An operation that requires eager evaluation was attempted on a lazily evaluated result set.

- `XmlException::NO_MEMORY_ERROR`

  An attempt to allocate memory failed.

- `XmlException::NULL_POINTER`

  An attempt was made to use a NULL pointer, most likely from a non-C++ API, such as Java or Python.

- `XmlException::OPERATION_INTERRUPTED`

  A query operation was interrupted by the application.

- `XmlException::OPERATION_TIMEOUT`

  A query operation timed out.

- `XmlException::TRANSACTION_ERROR`

  An attempt was made to refer to a transaction that was already committed or aborted.

- `XmlException::UNIQUE_ERROR`

  An index uniqueness constraint was violated.

- `XmlException::UNKNOWN_INDEX`

  The referenced indexing strategy is not known.

- `XmlException::VERSION_MISMATCH`

  The container version and library version are not compatible.

- `XmlException::QUERY_EVALUATION_ERROR`

  An error occurred during evaluation of an XQuery expression.

- `XmlException::QUERY_PARSER_ERROR`

  An error occurred attempting to parse an XQuery expression.

Class

XmlException (page 206)

See Also

XmlException Methods (page 209)

XmlException::getQueryColumn

```
#include <DbXml.hpp>

int XmlException::getQueryColumn() const
```

If the exception code is `XmlException::QUERY_PARSER_ERROR`, this method returns the column number in the query expression that caused the error. See XmlException::getQueryLine (page 215) for the line number.

Class

XmlException (page 206)

See Also

XmlException Methods (page 209)

XmlException::getQueryFile

```
#include <DbXml.hpp>

const char * XmlException::getQueryFile() const
```

If the exception code is `XmlException::QUERY_PARSER_ERROR`, this method returns the name of the module containing the error if it is available; otherwise it returns null. Use XmlException::getQueryLine (page 215) and XmlException::getQueryColumn (page 213) to get more information.

Class

XmlException (page 206)

See Also

XmlException Methods (page 209)

XmlException::getQueryLine

```
#include <DbXml.hpp>

int XmlException::getQueryLine() const
```

If the exception code is `XmlException::QUERY_PARSER_ERROR`, this method returns the line number in the query expression that caused the error. See XmlException::getQueryColumn (page 213) for the column number.

Class

XmlException (page 206)

See Also

XmlException Methods (page 209)

XmlException::what

```
#include <DbXml.hpp>

const char *XmlException::what() const throw()
```

Returns a string description of the exception.

Class

XmlException (page 206)

See Also

XmlException Methods (page 209)

Chapter 14. XmlExternalFunction

```
#include <DbXml.hpp>

class DbXml::XmlExternalFunction {
public:
    virtual XmlResults execute(XmlTransaction &txn,
                               XmlManager &mgr,
                               const XmlArguments &args) const;
    virtual void close();
};
```

The `XmlExternalFunction` class is used to implement XQuery extension functions in C++. An implementor creates a subclass of `XmlExternalFunction` and returns an instance of the class from XmlResolver::resolveExternalFunction (page 376). Returned instances may be singleton objects which means the application must eventually free the memory, or new instances in which case the instance should delete itself in its XmlExternalFunction::close() (page 220) method.

XmlExternalFunction Methods

XmlExternalFunction Methods	Description
XmlExternalFunction::execute()	Execute the function for which the instance was implemented
XmlExternalFunction::close()	Query is finished with the object

XmlExternalFunction::execute()

```
#include <DbXml.hpp>

virtual XmlResults
XmlExternalFunction::execute(XmlTransaction &txn, XmlManager &mgr,
                             const XmlArguments &args) const
```

This method is called from within a query to execute the function for which the instance was implemented. It returns an XmlResults (page 380) object used by the query for further processing.

Parameters

txn

The XmlTransaction (page 407) in which the operation is running.

mgr

The XmlManager (page 274) instance controlling the operation.

args

The XmlArguments (page 8) object used to retrieve the function arguments.

Class

XmlExternalFunction (page 217)

See Also

XmlExternalFunction Methods (page 218)

XmlExternalFunction::close()

```
#include <DbXml.hpp>

virtual void XmlExternalFunction::close()
```

This method is called when the query has finished using the object. It allows the implementation to clean up if necessary. If a new instance of XmlExternalFunction (page 217) is returned from XmlResolver::resolveExternalFunction (page 376) then this method will probably need to delete itself to avoid memory leaks.

Class

XmlExternalFunction (page 217)

See Also

XmlExternalFunction Methods (page 218)

Chapter 15. XmlIndexLookup

```
#include <DbXml.hpp>

class DbXml::XmlIndexLookup {
public:
 XmlIndexLookup()
 XmlIndexLookup(const XmlIndexLookup &o)
 XmlIndexLookup &operator=(const XmlIndexLookup &o)
 ~XmlIndexLookup();
 ...
};
```

The `XmlIndexLookup` class encapsulates the context within which an index lookup operation can be performed on an XmlContainer (page 16) object. The lookup is performed using an `XmlIndexLookup` object, and a series of methods of that object that specify how the lookup is to be performed. Using these methods, it is possible to specify inequality lookups, range lookups, and simple value lookups, as well as the sort order of the results. By default, results are returned in the sort order of the index.

`XmlIndexLookup` objects are created using XmlManager::createIndexLookup (page 287).

`XmlIndexLookup` objects are not thread-safe, and may not be shared among threads while being configured. A constructed, read-only object may be shared among threads for multiple calls to XmlIndexLookup::execute (page 223).

XmlIndexLookup Methods

XmlIndexLookup Methods	Description
XmlIndexLookup::execute	Execute the index lookup operation.
XmlIndexLookup::setContainer	Set the target container for the lookup.
XmlIndexLookup::setHighBound	Set the high bound for a range lookup.
XmlIndexLookup::setIndex	Set the index to be used for the lookup.
XmlIndexLookup::setLowBound	Set the lower bound for the lookup.
XmlIndexLookup::setNode	Set the target node name for the lookup.
XmlIndexLookup::setParent	Set the parent node for edge index lookups.

XmlIndexLookup::execute

```
#include <DbXml.hpp>

XmlResults results = XmlIndexLookup::execute(
              XmlQueryContext &context,
       u_int32_t flags = 0) const

XmlResults results = XmlIndexLookup::execute(
        XmlTransaction &txn,
     XmlQueryContext &context,
     u_int32_t flags = 0) const
```

Executes the index lookup operation specified by the configuration of the XmlIndexLookup (page 221) object.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

context

The XmlQueryContext (page 341) to use for this query.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DBXML_WELL_FORMED_ONLY`

  Force the use of a scanner that will neither validate nor read schema or dtds associated with the document during parsing. This is efficient, but can cause parsing errors if the document references information that might have come from a schema or dtd, such as entity references.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_READ_COMMITTED`

This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_REVERSE_ORDER`

  Return results in reverse order relative to the sort of the index.

- `DBXML_NO_INDEX_NODES`

  Relevant for node storage containers with node indices only. Causes the `XmlIndexLookup::execute()` operations to return document nodes rather than direct pointers to the interior nodes. This is more efficient if all that is desired is a reference to target documents.

- `DBXML_CACHE_DOCUMENTS`

  Enables use of a cache mechanism that optimizes `XmlIndexLookup::execute()` operations that a large number of nodes from the same document.

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

XmlIndexLookup::setContainer

```
#include <DbXml.hpp>

void XmlIndexLookup::setContainer(XmlContainer &container)

const XmlContainer & XmlIndexLookup::getContainer() const
```

Sets the container to be used for the index lookup operation. The same `XmlIndexLookup` object may be used for lookup in multiple containers by changing this configuration.

Parameters

container

The container to be used for the lookup operation.

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

XmlIndexLookup::setHighBound

```
#include <DbXml.hpp>

void XmlIndexLookup::setHighBound(
     const XmlValue &value,
     XmlIndexLookup::Operation op)

XmlIndexLookup::Operation XmlIndexLookup::getHighBoundOperation() const

const XmlValue &XmlIndexLookup::getHighBoundValue() const
```

Sets the operation and value to be used for the upper bound for a range index lookup operation. The high bound must be specified to indicate a range lookup.

Parameters

value

The value to be used for the upper bound. Use of an empty value results in an inequality lookup, rather than a range lookup.

op

The operation to be used on the upper bound. Must be one of:

- `XmlIndexLookup::LT`

  less than

- `XmlIndexLookup::LTE`

  less than or equal to

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

XmlIndexLookup::setIndex

```
#include <DbXml.hpp>

void XmlIndexLookup::setIndex(const std::string &index)

const std::string &XmlIndexLookup::getIndex() const
```

Sets the indexing strategy to be used for the index lookup operation. Only one index may be specified, and substring indices are not supported.

Parameters

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none
    node-element-equality-float
```

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

XmlIndexLookup::setLowBound

```
#include <DbXml.hpp>

void XmlIndexLookup::setLowBound(
     const XmlValue &value,
     XmlIndexLookup::Operation op)

XmlIndexLookup::Operation XmlIndexLookup::getLowBoundOperation() const

const XmlValue &XmlIndexLookup::getLowBoundValue() const
```

Sets the operation and value to be used for the index lookup operation. If the operation is a simple inequality lookup, the lower bound is used as the single value and operation for the lookup. If the operation is a range lookup, in which an upper bound is specified, the lower bound is used as the lower boundary value and operation for the lookup.

Parameters

value

The value to be used for the lower bound. An empty value is specified using an uninitialized XmlValue (page 416) object.

op

Selects the operation to be performed. Must be one of:

- `XmlIndexLookup::NONE`

  None

- `XmlIndexLookup::EQ`

  Equal

- `XmlIndexLookup::LT`

  Less than

- `XmlIndexLookup::LTE`

  Less than or equal to

- `XmlIndexLookup::GT`

  Greater than

- `XmlIndexLookup::GTE`

  Greater than or equal

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

XmlIndexLookup::setNode

```
#include <DbXml.hpp>

void XmlIndexLookup::setNode(const std::string &uri,
        const std::string &name)

const std::string &XmlIndexLookup::getNodeURI() const

const std::string &XmlIndexLookup::getNodeName() const
```

Sets the name of the node to be used along with the indexing strategy for the index lookup operation.

Parameters

uri

The namespace of the node to be used. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be used.

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

XmlIndexLookup::setParent

```
#include <DbXml.hpp>

void XmlIndexLookup::setParent(const std::string &uri,
        const std::string &name)

const std::string &XmlIndexLookup::getParentURI() const

const std::string &XmlIndexLookup::getParentName() const
```

Sets the name of the parent node to be used for an edge index lookup operation. If the index is not an edge index, this configuration is ignored.

Parameters

uri

The namespace of the parent node to be used. The default namespace is selected by passing an empty string for the namespace.

name

The name of the parent element node to be used.

Class

XmlIndexLookup (page 221)

See Also

XmlIndexLookup Methods (page 222)

Chapter 16. XmlIndexSpecification

```
#include <DbXml.hpp>

class DbXml::XmlIndexSpecification {
public:
 XmlIndexSpecification();
 XmlIndexSpecification(const XmlIndexSpecification &);
 ~XmlIndexSpecification();
 XmlIndexSpecification &operator = (const XmlIndexSpecification &)
 ...
};
```

The `XmlIndexSpecification` class encapsulates the indexing specification of a container. An indexing specification can be retrieved with the XmlContainer::getIndexSpecification (page 42) method, and modified using the XmlContainer::setIndexSpecification (page 72) method.

The `XmlIndexSpecification` class provides an interface for manipulating the indexing specification through the XmlIndexSpecification::addIndex (page 239), XmlIndexSpecification::deleteIndex (page 248), and XmlIndexSpecification::replaceIndex (page 264) methods. The class interface also provides the XmlIndexSpecification::next (page 258) and XmlIndexSpecification::reset (page 268) methods for iterating through the specified indices. The XmlIndexSpecification::find (page 252) method can be used to search for the indexing strategy for a known node.

Finally, you can set a default index specification for a container using XmlIndexSpecification::addDefaultIndex (page 235). You can replace and delete the default index using XmlIndexSpecification::replaceDefaultIndex (page 260) and XmlIndexSpecification::deleteDefaultIndex (page 244).

Note that adding an index to a container results in re-indexing all of the documents in that container, which can take a very long time. It is good practice to design an application to add useful indices before populating a container.

A copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body. This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlIndexSpecification Methods

XmlIndexSpecification Methods	Description
XmlIndexSpecification::addDefaultIndex	Add a default index.
XmlIndexSpecification::addIndex	Add an index
XmlIndexSpecification::deleteDefaultIndex	Delete the index from the default specification.
XmlIndexSpecification::deleteIndex	Delete an index.
XmlIndexSpecification::find	Find the index for a specific node.
XmlIndexSpecification::getAutoIndexing	Get the current auto-indexing state.
XmlIndexSpecification::getDefaultIndex	Get the default index.
XmlIndexSpecification::getValueType	Get the XmlValue::type from an index string.
XmlIndexSpecification::next	Get the next index in the index specification.
XmlIndexSpecification::replaceDefaultIndex	Replace the default index.
XmlIndexSpecification::replaceIndex	Replace the index for a node.
XmlIndexSpecification::reset	Reset the index iterator.
XmlIndexSpecification::setAutoIndexing	Set the current auto-indexing state.

XmlIndexSpecification::addDefaultIndex

```
#include <DbXml.hpp>

void XmlIndexSpecification::addDefaultIndex(const std::string &index)

void XmlIndexSpecification::addDefaultIndex(Type type,
       XmlValue::Type syntax)
```

Adds an indexing strategy to the default index specification. That is, the index provided on this method is applied to all nodes in a container, except for those for which an explicit index is already declared. For more information on specifying indexing strategies, see XmlIndexSpecification::addIndex (page 239).

You can specify an index by using a string, or by using enumerated values.

Specifying indexes as strings

```
#include <DbXml.hpp>

void XmlIndexSpecification::addDefaultIndex(const std::string &index)
```

Identifies one or more indexing strategies to the default index.

Parameters are:

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary

date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Specifying indexes as enumerated values

```
#include <DbXml.hpp>

void XmlIndexSpecification::addDefaultIndex(Type type,
       XmlValue::Type syntax)
```

Adds a single indexing strategy to the default index.

Parameters are:

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`
- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`
- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`
- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`
- `XmlValue::DURATION`
- `XmlValue::FLOAT`
- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`
- `XmlValue::STRING`

- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

Errors

The `XmlIndexSpecification::addDefaultIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::addIndex

```
#include <DbXml.hpp>

void XmlIndexSpecification::addIndex(
    const std::string &uri, const std::string &name,
    const std::string &index)

void XmlIndexSpecification::addIndex(
    const std::string &uri, const std::string &name, Type type,
    XmlValue::Type syntax)
```

Adds an index to the index specification. You then set the index specification using XmlContainer::setIndexSpecification (page 72).

You identify the indexing strategy that you want to add to the index specification in one of two ways. The first way is to provide a string that identifies the desired indexing strategy. The second is to use `XmlIndexSpecification::Type` and `XmlValue::Type` constants to identify the same information.

Either way, an index strategy is set by providing the name of a node and one or more indexing strategies for that node. The node name can be either that of an element, attribute, or metadata node. Metadata nodes are used only for indexing metadata. Element and attribute nodes are used for indexing XML document content. For example, in the XML fragment:

```
<art title='...'/>
```

there are two node names that index strategies could be specified for, the element node name is 'art', and the attribute node name is 'title'.

When specifying indexing strategies, you must provide the following information:

- Whether the index value must be unique. Of all the information you provide for an index strategy, this is the only one you are not required to specify. If it is not specified, then the indexed value is not required to be unique.
- Whether the index is for a specific node, or whether it is for an edge. An edge occurs in an XML document when two nodes meet in the document. For example, in the document:

  ```
  "<a><b><c>foo</c>o</b></a>"
  ```

  there is an edge at `<a><b>` and another one at `<b><c>`.
- Whether the node to be indexed is an element, attribute, or metadata node. If you are indexing a metadata node, then the index must be a node index (not an edge index).
- Whether the index is a presence index (the index indicates whether the node or node edge exists in the document), an equality index (the index tracks the exact value set for the node), or an substring index (the index tracks all the substrings, 3 characters long and greater, that can be constructed for the node).
- The indexed value's syntax. There is a large list of available syntaxes, and they range everywhere from a boolean to a date to time information.

In addition to setting index strategies for specified nodes, applications can also specify a default indexing strategy for all nodes in a container by using XmlIndexSpecification::addDefaultIndex (page 235). When a container is first created the default indexing strategy is `unique-node-metadata-equality-string`.

For more information on designing an indexing strategy for your application, see the Berkeley DB XML Getting Started Guide.

You can specify an index by using a string, or by using enumerated values.

Specifying indexes as strings

```
#include <DbXml.hpp>

void XmlIndexSpecification::addIndex(
    const std::string &uri, const std::string &name,
    const std::string &index)
```

Identifies one or more indexing strategies to set for the identified node. The strategies are identified as a space-separated listing of strings.

Parameters are:

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Specifying indexes as enumerated values

```
#include <DbXml.hpp>

void XmlIndexSpecification::addIndex(
    const std::string &uri, const std::string &name, Type type,
    XmlValue::Type syntax)
```

Identifies an indexing strategy to set for the identified node. The strategy is set using enumeration values for the index and the syntax.

Parameters are:

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`
- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`
- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`
- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`
- `XmlValue::DURATION`
- `XmlValue::FLOAT`

- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`
- `XmlValue::STRING`
- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

Errors

The `XmlIndexSpecification::addIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::deleteDefaultIndex

```
#include <DbXml.hpp>

void XmlContainer::deleteDefaultIndex(const std::string &index)

void XmlContainer::deleteDefaultIndex(Type type,
        XmlValue::Type syntax)
```

Delete the identified index from the default index specification. You can add additional indices to the default index specification using XmlIndexSpecification::addDefaultIndex (page 235). For more information on specifying indices, see XmlIndexSpecification::addIndex (page 239).

You can specify an index by using a string, or by using enumerated values.

Specifying indexes as strings

```
#include <DbXml.hpp>

void XmlContainer::deleteDefaultIndex(const std::string &index)
```

Deletes one or more indexing strategies from the default index.

Parameters are:

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary

date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Specifying indexes as enumerated values

```
#include <DbXml.hpp>

void XmlContainer::deleteDefaultIndex(Type type,
        XmlValue::Type syntax)
```

Deletes a single indexing strategy from the default index.

Parameters are:

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`
- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`
- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`
- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`
- `XmlValue::DURATION`
- `XmlValue::FLOAT`
- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`
- `XmlValue::STRING`

- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

Errors

The `XmlIndexSpecification::deleteDefaultIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::deleteIndex

```
#include <DbXml.hpp>

void XmlIndexSpecification::deleteIndex(const std::string &uri,
    const std::string &name, const std::string &index)

void XmlIndexSpecification::deleteIndex(const std::string &uri,
    const std::string &name, Type type, XmlValue::type syntax)
```

Deletes indexing strategies for a named document or metadata node. To delete an index set for metadata, specify the URI and name used when the metadata was added to the document.

You can specify an index by using a string, or by using enumerated values.

Specifying indexes as strings

```
#include <DbXml.hpp>

void XmlIndexSpecification::deleteIndex(const std::string &uri,
    const std::string &name, const std::string &index)
```

Identifies one or more indexing strategies to set for the identified node. The strategies are identified as a space-separated listing of strings.

Parameters are:

uri

The namespace of the node for which you want the index deleted. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.

- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Specifying indexes as enumerated values

```
#include <DbXml.hpp>

void XmlIndexSpecification::deleteIndex(const std::string &uri,
    const std::string &name, Type type, XmlValue::type syntax)
```

Deletes a single index strategy from the identified node.

Parameters are:

uri

The namespace of the node for which you want the index deleted. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`

- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`
- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`
- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`
- `XmlValue::DURATION`

- `XmlValue::FLOAT`
- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`
- `XmlValue::STRING`
- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

Errors

The `XmlIndexSpecification::deleteIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::find

```
#include <DbXml.hpp>

bool XmlIndexSpecification::find(
    const std::string &uri, const std::string &name, std::string &index)
```

Returns the indexing strategies for a named document or metadata node. This method returns true if an index for the node is found; otherwise, it returns false.

See XmlIndexSpecification::addIndex (page 239) for more information on index strategies.

Parameters

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::getAutoIndexing

```
#include <DbXml.hpp>

bool XmlIndexSpecification::getAutoIndexing() const
```

Indicates whether auto-indexing is turned on. This value can be set using XmlContainer::setIndexSpecification (page 72).

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::getDefaultIndex

```
#include <DbXml.hpp>

std::string XmlIndexSpecification::getDefaultIndex() const
```

Retrieves the default index. The default index is the index used by all nodes in the document in the absence of any other index.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::getValueType

```
#include <DbXml.hpp>

void XmlIndexSpecification::getValueType(
    const std::string &index)
```

Gets the `XmlValue::Type` specified in the given index specification.

Parameters

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::next

```
#include <DbXml.hpp>

bool XmlIndexSpecification::next(std::string &uri, std::string &name,
    std::string &index)

bool XmlIndexSpecification::next(std::string &uri, std::string &name,
    Type &type, XmlValue::Type &syntax)
```

Obtains the next index in the XmlIndexSpecification (page 233). Use XmlIndexSpecification::reset (page 268) to reset this iterator.

This method returns `true` if additional indices exist in the index list, otherwise it returns `false`.

Indexes can be retrieved as a string, or as an enumerated value.

Retrieving indexes as strings

```
#include <DbXml.hpp>

bool XmlIndexSpecification::next(std::string &uri, std::string &name,
    std::string &index)
```

Returns the next index in the index specification in a string format.

Parameters are:

uri

Receives the namespace of the node to which this index is applied.

name

Receives the name of the node to which this index is applied.

index

Identifies the index type used by this index. See the XmlIndexSpecification::addIndex (page 239) method for a description of what this string means.

Retrieving indexes as enumerated values

```
#include <DbXml.hpp>

bool XmlIndexSpecification::next(std::string &uri, std::string &name,
    Type &type, XmlValue::Type &syntax)
```

Returns the next index in the index specification using `XmlIndexSpecification::Type` and `XmlValue::Type` format.

Parameters are:

uri

Receives the namespace of the node to which this index is applied.

name

Receives the name of the node to which this index is applied.

type

Identifies the `XmlIndexSpecification::Type` used by this index. The value presented here is 3 or 4 different `XmlIndexSpecification::Type` values **or**'d together. See XmlIndexSpecification::addIndex (page 239) for a listing of these enumeration values.

syntax

Identifies the syntax used by this index. The value presented here is and `XmlValue::Type` value. See XmlIndexSpecification::addIndex (page 239) for a listing of these enumeration values.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::replaceDefaultIndex

```
#include <DbXml.hpp>

void XmlIndexSpecification::replaceDefaultIndex(
        const std::string &index)

void XmlIndexSpecification::replaceDefaultIndex(Type type,
        XmlValue::Type syntax)
```

Replaces the default indexing strategy for the container. The default index specification is used for all nodes in a document. You can add additional indices for specific document nodes using XmlIndexSpecification::addIndex (page 239).

You can specify an index by using a string, or by using enumerated values.

Specifying indexes as strings

```
#include <DbXml.hpp>

void XmlIndexSpecification::replaceDefaultIndex(
        const std::string &index)
```

Identifies one or more indexing strategies to set for the default index. The strategies are identified as a space-separated listing of strings.

Parameters are:

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear

base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Specifying indexes as enumerated values

```
#include <DbXml.hpp>

void XmlIndexSpecification::replaceDefaultIndex(Type type,
        XmlValue::Type syntax)
```

Identifies an indexing strategies to set for the default index.

Parameters are:

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`
- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`

- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`
- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`
- `XmlValue::DURATION`
- `XmlValue::FLOAT`
- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`

- `XmlValue::STRING`
- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

Errors

The `XmlIndexSpecification::replaceDefaultIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::replaceIndex

```
#include <DbXml.hpp>

void XmlIndexSpecification::replaceIndex(const std::string &uri,
    const std::string &name, const std::string &index)

void XmlIndexSpecification::replaceIndex(const std::string &uri,
    const std::string &name, Type type, XmlValue::Type syntax)
```

Replaces the indexing strategies for a named document or metadata node. All existing indexing strategies for that node are deleted, and the indexing strategy identified by this method is set for the node.

You can specify an index by using a string, or by using enumerated values.

Replacing indexes as strings

```
#include <DbXml.hpp>

void XmlIndexSpecification::replaceIndex(const std::string &uri,
    const std::string &name, const std::string &index)
```

Identifies one or more indexing strategies to set for the identified node. The strategies are identified as a space-separated listing of strings.

Parameters are:

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.

- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary
date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

Replacing indexes as enumerated values

```
#include <DbXml.hpp>

void XmlIndexSpecification::replaceIndex(const std::string &uri,
    const std::string &name, Type type, XmlValue::Type syntax)
```

Identifies a single indexing strategy to set for the identified node. The strategy is set using enumeration values for the index and the syntax.

Parameters are:

uri

The namespace of the node to be indexed. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be indexed.

type

A series of `XmlIndexSpecification::Type` values bitwise **OR**'d together to form the index strategy.

To indicate whether the indexed value must be unique container-wide, use one of the following, or leave the value out entirely:

- `XmlIndexSpecification::UNIQUE_OFF`
- `XmlIndexSpecification::UNIQUE_ON`

To identify the path type, use one of the following:

- `XmlIndexSpecification::PATH_NODE`
- `XmlIndexSpecification::PATH_EDGE`

To identify the node type, use one of the following:

- `XmlIndexSpecification::NODE_ELEMENT`
- `XmlIndexSpecification::NODE_ATTRIBUTE`
- `XmlIndexSpecification::NODE_METADATA`

Note that if `XmlIndexSpecification::NODE_METADATA` is used, then `XmlIndexSpecification::PATH_NODE` must also be used as well.

To identify the key type, use one of the following:

- `XmlIndexSpecification::KEY_PRESENCE`
- `XmlIndexSpecification::KEY_EQUALITY`
- `XmlIndexSpecification::KEY_SUBSTRING`

For example:

```
XmlIndexSpecification::PATH_NODE |
XmlIndexSpecification::NODE_ELEMENT |
XmlIndexSpecification::KEY_SUBSTRING
```

syntax

Identifies the type of information being indexed. The value must be one of the XmlValue (page 416) enumerated types:

- `XmlValue::NONE`
- `XmlValue::BASE_64_BINARY`
- `XmlValue::BOOLEAN`
- `XmlValue::DATE`
- `XmlValue::DATE_TIME`
- `XmlValue::DECIMAL`
- `XmlValue::DOUBLE`

- `XmlValue::DURATION`
- `XmlValue::FLOAT`
- `XmlValue::G_DAY`
- `XmlValue::G_MONTH`
- `XmlValue::G_MONTH_DAY`
- `XmlValue::G_YEAR`
- `XmlValue::G_YEAR_MONTH`
- `XmlValue::HEX_BINARY`
- `XmlValue::STRING`
- `XmlValue::TIME`

Note that if `XmlIndexSpecification::KEY_PRESENCE` is specified for the `type` parameter, then this parameter must be `XmlValue::NONE`.

Errors

The `XmlIndexSpecification::replaceIndex` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

UNKNOWN_INDEX

Unknown index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::reset

```
#include <DbXml.hpp>

void XmlIndexSpecification::reset()
```

Resets the index specification iterator to the beginning of the index list. Use XmlIndexSpecification::next (page 258) to iterate through the indices contained in the index specification.

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

XmlIndexSpecification::setAutoIndexing

```
#include <DbXml.hpp>

void XmlIndexSpecification::setAutoIndexing(bool value)
```

Sets the auto-indexing state which will affect the container.

If the value on the container is `true` (the default for newly-created containers) then indexes are added automatically for leaf elements and attributes. The indexes added are "node-equality-string" and "node-equality-double" for elements and attributes. If auto-indexing is not desired it should be disabled using this interface upon container creation. Auto-indexing is recognized by insertion of new documents as well as updates of existing documents, including modification via XQuery Update. The auto-indexing state is persistent and will remain stable across container close/re-open operations. Indexes added via auto-indexing are normal indexes and can be removed using the normal mechanisms.

A significant implication of auto-indexing is that any operation that may add an index (e.g. XmlContainer::putDocument (page 57)) can have the side effect of reindexing the entire container. For this reason auto-indexing is not recommended for containers of heterogenous documents and that it be disabled once a representative set of documents has been inserted.

Parameters

value

Indicates whether auto-indexing behavior should be enabled (`true`) or disabled (`false`).

Class

XmlIndexSpecification (page 233)

See Also

XmlIndexSpecification Methods (page 234)

Chapter 17. XmlInputStream

```
#include <DbXml.hpp>

virtual XmlInputStream::~XmlInputStream()
```

Used to read and write XML data. You can obtain an instance of this object using one of XmlManager::createLocalFileInputStream (page 290), XmlManager::createMemBufInputStream (page 291), XmlManager::createStdInInputStream (page 295), XmlManager::createURLInputStream (page 300), or XmlDocument::getContentAsXmlInputStream (page 141). You use instances of this class with XmlContainer::putDocument (page 57) and XmlDocument::setContentAsXmlInputStream (page 148).

You can manually retrieve the contents of the input stream using XmlInputStream::readBytes (page 273) and XmlInputStream::curPos (page 272).

`XmlInputStream` is a pure virtual interface. In C++, you can subclass `XmlInputStream`, and pass an instance of your class to any of the methods that take it as a parameter, such as XmlContainer::putDocument (page 57). This is especially useful for streaming XML from an application directly into Berkeley DB XML without first converting it to a string.

XmlInputStream Methods

XmlInputStream Methods	Description
XmlInputStream::curPos	Return the current position in the stream.
XmlInputStream::readBytes	Read bytes from the stream.

XmlInputStream::curPos

```
#include <DbXml.hpp>

virtual unsigned int XmlInputStream::curPos() const = 0
```

Returns the number of bytes currently read from the beginning of the input stream.

Class

XmlInputStream (page 270)

See Also

XmlInputStream Methods (page 271)

XmlInputStream::readBytes

```
#include <DbXml.hpp>

virtual unsigned int XmlInputStream::readBytes(
    char *toFill, const unsigned int maxToRead)
```

Reads `maxToRead` number of bytes from the input stream and places those bytes in `toFill`. Returns the number of bytes read, or 0 if the end of the stream has been reached.

Parameters

toFill

Specifies a pointer to a buffer used to place the bytes read from the input stream. It is the responsibility of the programmer to ensure that the buffer provided here is large enough for the amount of data to be read.

maxToRead

Identifies the maximum number of bytes to read from the input stream.

Class

XmlInputStream (page 270)

See Also

XmlInputStream Methods (page 271)

Chapter 18. XmlManager

```
#include <DbXml.hpp>

XmlManager::XmlManager(DB_ENV *dbenv, u_int32_t flags = 0)
XmlManager::XmlManager(u_int32_t flags)
XmlManager::XmlManager()
XmlManager::XmlManager(const XmlManager &o)
XmlManager &operator = (const XmlManager &o)
XmlManager::~XmlManager()
```

Provides a high-level object used to manage various aspects of Berkeley DB XML usage. You use `XmlManager` to perform activities such as container management (including creation and open), preparing XQuery queries, executing one-off queries, creating transaction objects, creating update and query context objects, and creating input streams.

A copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body.

This object is free threaded, and can be safely shared among threads in an application.

There are several forms of the constructor available for this class; one that accepts an environment handle, one that uses a private internal environment, and a default constructor.

Using the constructor with an environment handle

```
#include <DbXml.hpp>

XmlManager::XmlManager(DB_ENV *dbenv, u_int32_t flags = 0)
```

`XmlManager` constructor that uses the provided `DB_ENV` for the underlying environment. The Berkeley DB subsystems initiated by this environment (for example, transactions, logging, the memory pool), are the subsystems that are available to Berkeley DB XML when operations are performed using this manager object.

Parameters are:

dbenv

The `DB_ENV` to use for the underlying database environment. The environment provided here must be opened.

flags

Must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_ALLOW_EXTERNAL_ACCESS`

  If set, this flag allows XQuery queries to access data sources external to the container, such as files on disk or http URIs. By default, such access is not allowed.

- `DBXML_ALLOW_AUTO_OPEN`

If set, XQuery queries that reference unopened containers will automatically open those containers, and close them when references resulting from the query are released. By default, a query will fail if it refers to containers that are not open.

- `DBXML_ADOPT_DBENV`

  If set, the `XmlManager` object will close and delete the underlying `DB_ENV` handle at the end of the `XmlManager`'s life.

Using the constructor with an internal environment

```
#include <DbXml.hpp>

XmlManager::XmlManager(u_int32_t flags = 0)
```

`XmlManager` constructor that uses a private internal database environment. This environment is opened with `DB_PRIVATE|DB_CREATE|DB_INIT_MPOOL`. These flags allow the underlying environment to be created if it does not already exist. In addition, the memory pool (in-memory cache) is initialized and available. Finally, the environment is private, which means that no external processes can join the environment, but the `XmlManager` object can be shared between threads within the opening process.

Note that for this form of the constructor, the environment home is located in either the current working directory, or in the directory identified by the `DB_HOME` environment variable.

Parameters are:

flags

Must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_ALLOW_EXTERNAL_ACCESS`

  If set, this flag allows XQuery queries to access data sources external to the container, such as files on disk or http URIs. By default, such access is not allowed.

- `DBXML_ALLOW_AUTO_OPEN`

  If set, XQuery queries that reference unopened containers will automatically open those containers, and close them when references resulting from the query are released. By default, a query will fail if it refers to containers that are not open.

Default constructor

```
#include <DbXml.hpp>

XmlManager::XmlManager()
```

A default `XmlManager` constructor. This constructor provides the same behavior as passing a `flags` parameter of 0 to the constructor that takes a single flags parameter. This constructor is provided for convenience.

Errors

The `XmlManager` constructor may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INTERNAL_ERROR

You used a form of the constructor that does not take an external `DB_ENV` handle, and an error occured opening the default database environment.

INVALID_VALUE

You provided an invalid value to the `flags` parameter.

XmlManager Methods

XmlManager Methods	Description
XmlManager::compactContainer	Compact the databases comprising the container.
XmlManager::createContainer	Create an XmlContainer
XmlManager::createDocument	Instantiate an XmlDocument object.
XmlManager::createIndexLookup	Instantiate an XmlIndexLookup object.
XmlManager::createLocalFileInputStream	Create an input stream to a file on disk.
XmlManager::createMemBufInputStream	Create an input stream to a memory buffer.
XmlManager::createQueryContext	Instantiate an XmlQueryContext object.
XmlManager::createResults	Instantiate an empty XmlResults object.
XmlManager::createStdInInputStream	Create an input stream to the console.
XmlManager::createTransaction	Instantiate an XmlTransaction object.
XmlManager::createUpdateContext	Instantiate an XmlUpdateContext object.
XmlManager::createURLInputStream	Create an input stream to the specified URL.
XmlManager::dumpContainer	Dump the container.
XmlManager::existsContainer	Determine if container exists.
XmlManager::getDB_ENV	Get the database environment.
XmlManager::getDefaultContainerConfig	Get a copy of the default XmlContainerConfig.
XmlManager::getFlags	Get the flags used to open the manager.
XmlManager::getHome	Get the environment home directory.
XmlManager::getImplicitTimezone	Get the implicit timezone used for queries.
XmlManager::loadContainer	Load the container.
XmlManager::openContainer	Open an already existing XmlContainer.
XmlManager::prepare	Get an XmlQueryExpression object.
XmlManager::query	Execute a query.
XmlManager::registerCompression	The XmlCompression implementing user defined compression.
XmlManager::registerResolver	The XmlResolver that implements file resolution policy.
XmlManager::reindexContainer	Reindex the container.
XmlManager::removeContainer	Delete the container.
XmlManager::renameContainer	Rename the container.
XmlManager::setDefaultContainerConfig	Set the default XmlContainerConfig.
XmlManager::setDefaultContainerFlags	Set the default flags.
XmlManager::setDefaultContainerType	Set the default container type.

XmlManager Methods	Description
XmlManager::setDefaultPageSize	Set the underlying database page size.
XmlManager::setDefaultSequenceIncrement	Set the sequence number generation cache size.
XmlManager::setImplicitTimezone	Set the implicit timezone used for queries.
XmlManager::truncateContainer	Truncate the container.
XmlManager::upgradeContainer	Upgrade the container.
XmlManager::verifyContainer	Verify the container.

XmlManager::compactContainer

```
#include <DbXml.hpp>

void XmlManager::compactContainer(
 const std::string &name, XmlUpdateContext &context)

void XmlManager::compactContainer(
 XmlTransaction &txn, const std::string &name,
 XmlUpdateContext &context)

void XmlManager::compactContainer(
 const std::string &name, XmlUpdateContext &context,
 const XmlContainerConfig &flags)

void XmlManager::compactContainer(
 XmlTransaction &txn, const std::string &name,
 XmlUpdateContext &context, const XmlContainerConfig &flags)
```

Compacts all of the databases in the container using `Db::compact`.

The container must be closed; the system throws an exception if the container is open.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The name of the container to be compacted.

context

The XmlUpdateContext (page 415) object to be used for this operation.

flags

This parameter is unused.

Errors

The XmlManager::upgradeContainer (page 335) method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createContainer

```
#include <DbXml.hpp>

XmlContainer XmlManager::createContainer(const std::string &name)

XmlContainer XmlManager::createContainer(
    XmlTransaction &txn, const std::string &name)

XmlContainer XmlManager::createContainer(const std::string &name,
    const XmlContainerConfig &config)

XmlContainer XmlManager::createContainer(
    XmlTransaction &txn, const std::string &name,
    XmlContainerConfig config)

XmlContainer XmlManager::createContainer(const std::string &name,
    const XmlContainerConfig &flags,
    XmlContainer::ContainerType type, int mode = 0)

XmlContainer XmlManager::createContainer(XmlTransaction &txn,
    const std::string &name, const XmlContainerConfig &flags,
    XmlContainer::ContainerType type, int mode = 0)
```

Creates and opens a container, returning a handle to an XmlContainer (page 16) object. If the container already exists at the time this method is called, an exception is thrown.

Use XmlManager::openContainer (page 310) to open a container that has already been created.

Containers always remain open until the last handle referencing the container is destroyed.

There are two basic forms of this method: one that accepts an XmlContainerConfig (page 76) object, and a simpler form that creates a default container.

Creating a default container

```
#include <DbXml.hpp>

XmlContainer XmlManager::createContainer(const std::string &name)

XmlContainer XmlManager::createContainer(
    XmlTransaction &txn, const std::string &name)
```

Creates a default container. The container is created using the settings `XmlContainerConfig::setAllowCreate` and `XmlContainerConfig::setExclusiveCreate` set to `true` unless the default behavior has been overridden using `XmlManager::setDefaultContainerConfig`. In addition, the container is set up to use node-level storage unless `XmlManager::setDefaultContainerConfig` is used.

Parameters are:

txn

The XmlTransaction (page 407) object to use for this container creation.

name

The container's name. The container is created relative to the underlying environment's home directory (see the XmlManager (page 274) class description for more information) unless an absolute path is used for the name; in that case the container is created in the location identified by the path.

The name provided must be unique for the environment or an exception is thrown.

Creating a container using an XmlContainerConfig object

```
#include <DbXml.hpp>

XmlContainer XmlManager::createContainer(const std::string &name,
    const XmlContainerConfig &config)

XmlContainer XmlManager::createContainer(
    XmlTransaction &txn, const std::string &name,
    XmlContainerConfig config)

XmlContainer XmlManager::createContainer(const std::string &name,
    const XmlContainerConfig &flags,
    XmlContainer::ContainerType type, int mode = 0)

XmlContainer XmlManager::createContainer(XmlTransaction &txn,
    const std::string &name, const XmlContainerConfig &flags,
    XmlContainer::ContainerType type, int mode = 0)
```

Creates a container.

Parameters are:

txn

The XmlTransaction (page 407) object to use for this container creation.

name

The container's name. The container is created relative to the underlying environment's home directory (see the XmlManager (page 274) class description for more information) unless an absolute path is used for the name; in that case the container is created in the location identified by the path.

The name provided must be unique for the environment or an exception is thrown.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- DB_CREATE

  If the container does not currently exist, create it.

- DB_READ_UNCOMMITTED

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the DB_READ_UNCOMMITTED flag was not specified when the underlying container was opened.

- DB_EXCL

  Return an error if the container already exists. This flag is only meaningful when specified with the DB_CREATE flag.

- DB_MULTIVERSION

  Open the database with support for multiversion concurrency control. This will cause updates to the container to follow a copy-on-write protocol, which is required to support snapshot isolation. This flag requires that the container be transactionally protected during its open.

- DB_NOMMAP

  Do not map this container into process memory (see the DbEnv::set_mp_mmapsize() method for further information).

- DB_RDONLY

  Open the container for reading only. Any attempt to modify items in the container will fail, regardless of the actual permissions of any underlying files.

- DB_THREAD

  Cause the container handle to be *free-threaded*; that is, concurrently usable by multiple threads in the address space.

- DBXML_CHKSUM

  Do checksum verification of pages read into the cache from the backing filestore. Berkeley DB XML uses the SHA1 Secure Hash Algorithm if encryption is configured and a general hash algorithm if it is not.

- DBXML_ENCRYPT

  Encrypt the database using the cryptographic password specified to DbEnv::set_encrypt().

- DB_TXN_NOT_DURABLE

  If set, Berkeley DB XML will not write log records for this database. This means that updates of this database exhibit the ACI (atomicity, consistency, and isolation) properties, but not D (durability); that is, database integrity will be maintained, but if the application or system fails, integrity will not persist. The database file must be verified and/or restored from backup after a failure.

- `DBXML_INDEX_NODES`

  Causes the indexer to create index targets that reference nodes rather than documents. This allows index lookups during query processing to more efficiently find target nodes and avoid walking the document tree. It can apply to both container types, and is the default for containers of type `XmlContainer::NodeContainer`.

- `DBXML_NO_INDEX_NODES`

  Causes the indexer to create index targets that reference documents rather than nodes. This can be more desirable for simple queries that only need to return documents and do relatively little navigation during querying. It can apply to both container types, and is the default for containers of type `XmlContainer::WholedocContainer`.

- `DBXML_STATISTICS`

  Causes the container to be created to include structural statistics information, which is very useful for cost based query optimisation. Containers created with these statistics will take longer to load and update, since the statistics must also be updated. This is the default.

- `DBXML_NO_STATISTICS`

  Causes the container to be created without structural statistics information – by default structural statistics are created.

- `DBXML_TRANSACTIONAL`

  Cause the container to support transactions. If this flag is set, an XmlTransaction (page 407) object may be used with any method that supports transactional protection. Also, if this flag is used, and if an XmlTransaction (page 407) object is not provided to a method that modifies an XmlContainer (page 16) or XmlDocument (page 135) object, then auto commit is automatically used for the operation.

- `DBXML_ALLOW_VALIDATION`

  When loading documents into the container, validate the XML if it refers to a DTD or XML Schema.

Note that regardless the setting of your flags, `XmlContainerConfig::setAllowCreate` and `XmlContainerConfig::setExclusiveCreate` will be set to `true`.

type

The type of container to create. The container type must be one of the following values:

- `XmlContainer::NodeContainer`

Documents are broken down into their component nodes, and these nodes are stored individually in the container. This is the preferred container storage type.

- `XmlContainer::WholedocContainer`

  Documents are stored intact; all white space and formatting is preserved.

mode

On Windows systems, mode is ignored.

On UNIX systems or in IEEE/ANSI Std 1003.1 (POSIX) environments, files are created with mode `mode` (as described in `chmod(2)`) and modified by the process' umask value at the time of creation (see `umask(2)`). Created files are owned by the process owner; the group ownership of created files is based on the system and directory defaults, and is not further specified by Berkeley DB. System shared memory segments created by are created with mode `mode`, unmodified by the process' umask value. If `mode` is 0, DB XML will use a default mode of readable and writable by both owner and group.

Errors

The `XmlManager::createContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_EXISTS

The container name that you provided already exists, and the `DB_CREATE` flag is set to `false`.

INVALID_VALUE

You provided an invalid value to the `flags` parameter.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createDocument

```
#include <DbXml.hpp>

XmlDocument XmlManager::createDocument()
```

Instantiate an XmlDocument (page 135) object.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createIndexLookup

```
#include <DbXml.hpp>

XmlIndexLookup XmlManager::createIndexLookup(
    XmlContainer &container, const std::string &uri,
    const std::string &name, const std::string &index,
    const XmlValue &value = XmlValue(),
 XmlIndexLookup::Operation op = XmlIndexLookup::EQ)
```

Instantiates an new XmlIndexLookup (page 221) object for performing index lookup operations. Only a single index may be specified, and substring indices are not supported.

Parameters

container

The target container for the lookup operation.

uri

The namespace of the node to be used. The default namespace is selected by passing an empty string for the namespace.

name

The name of the element or attribute node to be used.

index

A comma-separated list of strings that represent the indexing strategy. The strings must contain the following information in the following order:

```
unique-{path type}-{node type}-{key type}-{syntax}
```

where:

- **unique** indicates that the indexed value is unique in the container. If this keyword does not appear on the index string, then the indexed value is not required to be unique in the container.
- {path type} is either `node` or `edge`.
- {node type} is one of `element`, `attribute`, or `metadata`. If `metadata` is specified, then {path type} must be `node`.
- {key type} is one of `presence`, `equality`, or `substring`.
- {syntax} identifies the type of information being indexed. It must be one of the following values:

none	double	gYear
base64Binary	duration	gYearMonth
boolean	float	hexBinary

date	gDay	string
dateTime	gMonth	time
decimal	gMonthDay	

Note that if {key type} is `presence`, then {syntax} must be `none` or simply not specified.

Some example index strings are:

```
    unique-node-element-presence

    node-element-equality-string

    edge-element-presence-none

    node-element-equality-float
```

value

The value to be used as the single value for an equality or inequality lookup, or as the lower bound of a range lookup. An empty value is specified using an uninitialized XmlValue (page 416) object.

op

Selects the operation to be performed. Must be one of:

- `XmlIndexLookup::NONE`

  None

- `XmlIndexLookup::EQ`

  Equal

- `XmlIndexLookup::LT`

  Less than

- `XmlIndexLookup::LTE`

  Less than or equal to

- `XmlIndexLookup::GT`

  Greater than

- `XmlIndexLookup::GTE`

  Greater than or equal

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createLocalFileInputStream

```
#include <DbXml.hpp>

XmlInputStream *XmlManager::createLocalFileInputStream(
    const std::string &filename) const
```

Returns a XmlInputStream (page 270) to the file `filename`. Use this input stream with XmlContainer::putDocument (page 57) or XmlDocument::setContentAsXmlInputStream (page 148).

If the input stream is passed to either of these methods, it will be adopted, and deleted. If it is not passed, it is the responsibility of the user to delete the object. Note that there is no attempt to ensure that the file referenced contains well-formed or valid XML. Exceptions may be thrown at the time that this input stream is actually read if the stream does not contain well-formed, or valid XML.

Parameters

filename

The file to which you want an input stream formed.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createMemBufInputStream

```
#include <DbXml.hpp>

XmlInputStream *
XmlManager::createMemBufInputStream(const char *srcDocBytes,
    const unsigned int byteCount, const char *const bufId,
    const bool adoptBuffer = false) const

XmlInputStream *
XmlManager::createMemBufInputStream(const char *srcDocBytes,
    const unsigned int byteCount, const bool copyBuffer) const
```

Returns a XmlInputStream (page 270) to the in-memory buffer `srcDocBytes`.
Use this input stream with XmlContainer::putDocument (page 57) or
XmlDocument::setContentAsXmlInputStream (page 148).

If the input stream is passed to either of these methods, it will be adopted, and deleted. If it is not passed, it is the responsibility of the user to delete the object. Note that there is no attempt to ensure that the memory referenced contains well-formed or valid XML. Exceptions may be thrown at the time that this input stream is actually read if the stream does not contain well-formed, or valid XML.

The form that takes the copyBuffer boolean parameter optionally copies the srcDocBytes buffer to an internal buffer. This method leaves the srcDocBytes buffer intact, unconditionally.

Parameters

srcDocBytes

The memory buffer containing the XML document that you want to read.

byteCount

The size of the buffer referenced by `srcDocBytes`.

bufId

The system ID to use for this input stream. This can be any arbitrary system ID; it is used only to satisfy the XML parser that will read this buffer.

adoptBuffer

Indicates whether the buffer should be adopted. If true, the buffer is deleted when this input stream is deleted.

copyBuffer

Make an internal copy of the input buffer referenced by `srcDocBytes`.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createQueryContext

```
#include <DbXml.hpp>

XmlQueryContext XmlManager::createQueryContext(
    XmlQueryContext::ReturnType rt =XmlQueryContext::LiveValues,
    XmlQueryContext::EvaluationType et = XmlQueryContext::Eager)
```

Creates a new XmlQueryContext (page 341).

Parameters

rt

Specifies whether to return live or dead values:

- `XmlQueryContext::LiveValues`

  Return references to the actual document stored in Berkeley DB XML.

- `XmlQueryContext::DeadValues`

  Return a copy of the data stored in Berkeley DB XML.

evaluationType

The evaluation type must be specified as either:

- `XmlQueryContext::Eager`

  The query is executed and its resultant values are derived and stored in-memory before query evaluation is completed.

- `XmlQueryContext::Lazy`

  The query is executed and its resultant values are calculated as you ask for them.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createResults

```
#include <DbXml.hpp>

XmlResults XmlManager::createResults()
```

Instantiates an new, empty XmlResults (page 380) object. You can then use XmlResults::add (page 383) to add XmlValue (page 416) objects to this result set.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createStdInInputStream

```
#include <DbXml.hpp>

XmlInputStream *XmlManager::createStdInInputStream() const
```

Returns an XmlInputStream (page 270) to the console. Use this input stream with XmlContainer::putDocument (page 57) or XmlDocument::setContentAsXmlInputStream (page 148).

If the input stream is passed to either of these methods, it will be adopted, and deleted. If it is not passed, it is the responsibility of the user to delete the object.

Use this input stream with XmlContainer::putDocument (page 57) or XmlDocument::setContentAsXmlInputStream (page 148).

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createTransaction

```
#include <DbXml.hpp>

XmlTransaction XmlManager::createTransaction(DB_TXN *toAdopt)
XmlTransaction XmlManager::createTransaction(u_int32_t flags = 0)
```

The `XmlManager::createTransaction` method method creates a new XmlTransaction (page 407) object. If a `DB_TXN` object is not provided to this method, then a new transaction is begun (a `DB_TXN` object is instantiated and `DbEnv::txn_begin` is called).

If transactions were not initialized when this `XmlManager` object was opened (that is, DB_INIT_TXN was not specified) then this method throws an exception.

Parameters

DB_TXN

If a `DB_TXN` handle is passed to this method, the new XmlTransaction (page 407) is simply another reference for the `DB_TXN` handle. In this case, if the XmlTransaction (page 407) object is destroyed or goes out of scope before XmlTransaction::commit (page 410) or XmlTransaction::abort (page 409) are called, the state of the underlying transaction is left unchanged. This allows a transaction to be controlled external to its XmlTransaction (page 407) object. If no `DB_TXN` is passed, and the XmlTransaction (page 407) object is destroyed or goes out of scope, the transaction is implicitly aborted.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_TXN_NOSYNC`

  Do not synchronously flush the log when this transaction commits or prepares. This means the transaction will exhibit the ACI (atomic, consistent, and isolated) properties, but not D (durable); that is, database integrity will be maintained but it is possible that this transaction may be undone during recovery.

This behavior may be set for a Berkeley DB environment using the `DbEnv::set_flags()` method. Any value specified to this method overrides that setting.

- `DBXML_IGNORE_LEASE`

  This flag is relevant only when using a replicated environment.

  Perform transactional operations irrespective of the state of master leases. The transactional operations will perform under all conditions: if master leases are not configured, if the request is made to a client, if the request is made to a master with a valid lease, or if the request is made to a master without a valid lease.

  Refer to Master Leases in the *Berkeley DB Programmer's Reference Guide* for more information.

- `DB_TXN_NOWAIT`

  If a lock is unavailable for any Berkeley DB operation performed in the context of this transaction, cause the operation to return `DB_LOCK_DEADLOCK` or throw an XmlException (page 206) with DB error code `DB_LOCK_DEADLOCK` immediately instead of blocking on the lock.

- `DB_TXN_SNAPSHOT`

  This transaction will execute with snapshot isolation. For containers with the `DB_MULTIVERSION` flag set, data values will be read as they are when the transaction begins, without taking read locks. Silently ignored for operations on databases with `DB_MULTIVERSION` not set on the underlying container (read locks are acquired).

  The `DB_LOCK_DEADLOCK` error is returned from update operations if a snapshot transaction attempts to update data which was modified after the snapshot transaction read it.

- `DB_TXN_SYNC`

  Synchronously flush the log when this transaction commits or prepares. This means the transaction will exhibit all of the ACID (atomic, consistent, isolated, and durable) properties.

  This behavior is the default for Berkeley DB environments unless the `DB_TXN_NOSYNC` flag was specified to the `DbEnv::set_flags()` method. Any value specified to this method overrides that setting.

- `DBXML_IGNORE_LEASE`

  This flag is relevant only when using a replicated environment.

  Perform transactional operations irrespective of the state of master leases. The transactional operations will perform under all conditions: if master leases are not configured, if the request is made to a client, if the request is made to a master with a valid lease, or if the request was made to a master without a valid lease.

For information on master leases, see Master Leases in the *Berkeley DB Programmer's Reference Guide*.

Errors

The `XmlManager::createTransaction` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_VALUE

Cannot call `XmlManager::createTransaction` when transactions are not initialized

INVALID_VALUE

`XmlManager::createTransaction`(DB_TXN*) requires a valid DB_TXN handle

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createUpdateContext

```
#include <DbXml.hpp>

XmlUpdateContext XmlManager::createUpdateContext()
```

Instantiates a new, default, XmlUpdateContext (page 415) object. This object is used for XmlContainer (page 16) operations that add, delete, and modify documents, and documents in containers.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::createURLInputStream

```
#include <DbXml.hpp>

XmlInputStream *
XmlManager::createURLInputStream(const std::string &base Id,
    const std::string &systemId, const std::string &publicId) const

XmlInputStream *
XmlManager::createURLInputStream(
    const std::string &base Id, const std::string &systemId) const
```

Creates an input stream to the identified URL. File URLs are always supported by this method. URLs that require network access (for example, `http://...`) are supported only if Xerces was compiled with socket support. Use this input stream with XmlContainer::putDocument (page 57) or XmlDocument::setContentAsXmlInputStream (page 148).

If the input stream is passed to either of these methods, it will be adopted, and deleted. If it is not passed, it is the responsibility of the user to delete the object. Note that there is no attempt to ensure that the URI referenced contains well-formed or valid XML. Exceptions may be thrown at the time that this input stream is actually read if the stream does not contain well-formed, or valid XML.

Two forms of this method exist: one that accepts a publicId, and the other that does not.

Parameters

baseId

The base ID to use for this URL.

systemId

The system ID to use for this URL.

publicId

The public ID to use for this URL.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::dumpContainer

```
#include <DbXml.hpp>

void XmlManager::dumpContainer(const std::string name, std::ostream *out)
```

Dumps the contents of the specified container to the specified output stream. The container can be reconstructed by a call to XmlManager::loadContainer (page 308).

The container must be closed; the system throws an exception if the container is open.

The container must be have been opened at least once; the system throws an exception if the underlying files have not yet been created.

Parameters

name

The name of the container to be dumped.

out

The output stream to which the container is to be dumped.

Errors

The `XmlManager::dumpContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::existsContainer

```
#include <DbXml.hpp>

int XmlManager::existsContainer(const std::string &name);
```

The `XmlManager::existsContainer` method examines the named file, and if it is a container, returns a non-zero database format version. If the file does not exist, or is not a container, zero is returned.

The container may be open or closed; no exceptions will be thrown from this method.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The name of the file to be examined.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::getDB_ENV

```
#include <DbXml.hpp>

DB_ENV *XmlManager::getDB_ENV()
```

Returns a handle to the underlying database environment (DB_ENV).

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::getDefaultContainerConfig

```
#include <DbXml.hpp>

void XmlManager::setDefaultContainerConfig(XmlContainerConfig &config)
```

Sets the default container configuration used for containers opened and created by this `XmlManager` object. If a form of XmlManager::createContainer (page 281) or XmlManager::openContainer (page 310) is used that takes an `XmlContainerConfig` argument, the settings provided using this method are ignored. This will only affect containers opened or created after the value is set.

Parameters

config

The `XmlContainerConfig` object.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::getFlags

```
#include <DbXml.hpp>

u_int32_t XmlManager::getFlags() const
```

Returns the flags used to construct the XmlManager (page 274) object.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::getHome

```
#include <DbXml.hpp>

const std::string &XmlManager::getHome() const
```

Returns the home directory for the underlying database environment. XmlContainer (page 16) files are placed relative to this directory unless an absolute path is used for the container name.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::getImplicitTimezone

```
#include <DbXml.hpp>

int XmlManager::getImplicitTimezone() const
```

Returns the implicit timezone to be used for queries referring to dates and times in the context of the XmlManager (page 274), as an offset in minutes from GMT.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::loadContainer

```
#include <DbXml.hpp>

void XmlManager::loadContainer(const std::string name, std::istream *in,
    unsigned long *lineno, XmlUpdateContext &context)
```

Loads data from the specified stream into the container. The container's existing contents are discarded and replaced with the documents from the stream.

The specified input stream should contain data as created by XmlManager::dumpContainer (page 301).

The container must be closed; the system throws an exception if the container is open.

The container must be have been opened at least once; the system throws an exception if the underlying files have not yet been created.

Parameters

name

The name of the container to load.

in

The input stream from which the container is to be loaded.

lineno

The application uses lineno to specify the starting line number in the stream that is to be read. The system uses the same parameter to return the line number of the last line read from the stream.

context

The XmlUpdateContext (page 415) object to use for the load.

Errors

The `XmlManager::loadContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::openContainer

```
#include <DbXml.hpp>

XmlContainer XmlManager::openContainer(const std::string &name)

XmlContainer XmlManager::openContainer(
    XmlTransaction &txn, const std::string &name)

XmlContainer XmlManager::openContainer(
    const std::string &name, const XmlContainerConfig &config)

XmlContainer XmlManager::openContainer(
    XmlTransaction &txn, const std::string &name,
    const XmlContainerConfig &config)

XmlContainer XmlManager::openContainer(
    const std::string &name, const XmlContainerConfig &flags,
    XmlContainer::ContainerType type, int mode)

XmlContainer XmlManager::openContainer(
    XmlTransaction &txn, const std::string &name,
    const XmlContainerConfig &flags, XmlContainer::ContainerType type,
    int mode)
```

Opens a container, returning a handle to an XmlContainer (page 16) object. Unless the `config` or `flags` parameter is set to `true` using XmlContainerConfig::setAllowCreate (page 99), the container must already exist at the time that this method is called or an exception is thrown.

To create and open a new container, either use XmlManager::createContainer (page 281), or set XmlContainerConfig::setAllowCreate (page 99) to `true` for the `flags` or `config` parameter on this method.

Containers always remain open until the last handle referencing the container is destroyed.

The name provided here must be unique for the environment or an exception is thrown.

There are two basic forms of this method: one that accepts an XmlContainerConfig (page 76) object, and a simpler form that opens a default container.

Opening a default container

```
#include <DbXml.hpp>

XmlContainer XmlManager::openContainer(const std::string &name)

XmlContainer XmlManager::openContainer(
    XmlTransaction &txn, const std::string &name)
```

Opens the container, using only default values for all configuration options. Unless XmlContainerConfig::setAllowCreate (page 99) was set to `true` and assigned using

XmlManager::setDefaultContainerConfig (page 325), the container must have previously been created or an exception is thrown.

Parameters are:

txn

The XmlTransaction (page 407) object to use for this container open.

name

The container's name. The container is located relative to the underlying environment's home directory (see the XmlManager (page 274) class description for more information) unless an absolute path is used for the name; in that case the container is exists in the location identified by the path.

Opening a container using an XmlContainerConfig object

```
#include <DbXml.hpp>

XmlContainer XmlManager::openContainer(
    const std::string &name, const XmlContainerConfig &config)

XmlContainer XmlManager::openContainer(
    XmlTransaction &txn, const std::string &name,
    const XmlContainerConfig &config)

XmlContainer XmlManager::openContainer(
    const std::string &name, const XmlContainerConfig &flags,
    XmlContainer::ContainerType type, int mode)

XmlContainer XmlManager::openContainer(
    XmlTransaction &txn, const std::string &name,
    const XmlContainerConfig &flags, XmlContainer::ContainerType type,
    int mode)
```

Opens a container using the supplied XmlContainerConfig (page 76) object. If you are creating the container with this method, you can provide a container type and a mode value.

Unless XmlContainerConfig::setAllowCreate (page 99) was set to `true` and assigned using either the `flags` parameter or the XmlManager::setDefaultContainerConfig (page 325) method, the container must have previously been created or an exception is thrown.

Parameters are:

txn

The XmlTransaction (page 407) object to use for this container open.

name

The container's name. The container is located relative to the underlying environment's home directory (see the XmlManager (page 274) class description for more information) unless an absolute path is used for the name; in that case the container is exists in the location identified by the path.

type

The type of container to create. This parameter is relevant only if you have set XmlContainerConfig::setAllowCreate (page 99) to `true`.

The container type must be one of the following values:

- `XmlContainer::NodeContainer`

  Documents are broken down into their component nodes, and these nodes are stored individually in the container. This is the preferred container storage type.

- `XmlContainer::WholedocContainer`

  Documents are stored intact; all white space and formatting is preserved.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_CREATE`

  If the container does not currently exist, create it.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_EXCL`

  Return an error if the container already exists. This flag is only meaningful when specified with the `DB_CREATE` flag.

- `DB_MULTIVERSION`

  Open the database with support for multiversion concurrency control. This will cause updates to the container to follow a copy-on-write protocol, which is required to support snapshot isolation. This flag requires that the container be transactionally protected during its open.

- `DB_NOMMAP`

  Do not map this container into process memory (see the `DbEnv::set_mp_mmapsize()` method for further information).

- `DB_RDONLY`

Open the container for reading only. Any attempt to modify items in the container will fail, regardless of the actual permissions of any underlying files.

- `DB_THREAD`

  Cause the container handle to be *free-threaded*; that is, concurrently usable by multiple threads in the address space.

- `DBXML_CHKSUM`

  Do checksum verification of pages read into the cache from the backing filestore. Berkeley DB XML uses the SHA1 Secure Hash Algorithm if encryption is configured and a general hash algorithm if it is not.

- `DBXML_ENCRYPT`

  Encrypt the database using the cryptographic password specified to `DbEnv::set_encrypt()`.

- `DB_TXN_NOT_DURABLE`

  If set, Berkeley DB XML will not write log records for this database. This means that updates of this database exhibit the ACI (atomicity, consistency, and isolation) properties, but not D (durability); that is, database integrity will be maintained, but if the application or system fails, integrity will not persist. The database file must be verified and/or restored from backup after a failure.

- `DBXML_INDEX_NODES`

  Causes the indexer to create index targets that reference nodes rather than documents. This allows index lookups during query processing to more efficiently find target nodes and avoid walking the document tree. It can apply to both container types, and is the default for containers of type `XmlContainer::NodeContainer`.

- `DBXML_NO_INDEX_NODES`

  Causes the indexer to create index targets that reference documents rather than nodes. This can be more desirable for simple queries that only need to return documents and do relatively little navigation during querying. It can apply to both container types, and is the default for containers of type `XmlContainer::WholedocContainer`.

- `DBXML_STATISTICS`

  Causes the container to be created to include structural statistics information, which is very useful for cost based query optimisation. Containers created with these statistics will take longer to load and update, since the statistics must also be updated. This is the default.

- `DBXML_NO_STATISTICS`

  Causes the container to be created without structural statistics information – by default structural statistics are created.

- `DBXML_TRANSACTIONAL`

  Cause the container to support transactions. If this flag is set, an XmlTransaction (page 407) object may be used with any method that supports transactional protection. Also, if this flag is used, and if an XmlTransaction (page 407) object is not provided to a method that modifies an XmlContainer (page 16) or XmlDocument (page 135) object, then auto commit is automatically used for the operation.

- `DBXML_ALLOW_VALIDATION`

  When loading documents into the container, validate the XML if it refers to a DTD or XML Schema.

mode

On Windows systems, mode is always ignored.

Otherwise, this parameter is relevant only if you have set XmlContainerConfig::setAllowCreate (page 99) to `true`.

On UNIX systems or in IEEE/ANSI Std 1003.1 (POSIX) environments, files are created with mode `mode` (as described in `chmod(2)`) and modified by the process' umask value at the time of creation (see `umask(2)`). Created files are owned by the process owner; the group ownership of created files is based on the system and directory defaults, and is not further specified by Berkeley DB. System shared memory segments created by are created with mode `mode`, unmodified by the process' umask value. If `mode` is 0, DB XML will use a default mode of readable and writable by both owner and group.

Errors

The `XmlManager::openContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_NOT_FOUND

The container name that you provided does not exist, and you did not allow the container to be automatically created by setting XmlContainerConfig::setAllowCreate (page 99) to `true`.

INVALID_VALUE

You provided an invalid value to the `flags` parameter.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::prepare

```
#include <DbXml.hpp>

XmlQueryExpression XmlManager::prepare(const std::string &xquery,
    XmlQueryContext &context)

XmlQueryExpression XmlManager::prepare(XmlTransaction &txn,
    const std::string &xquery, XmlQueryContext &context)
```

Compile an XQuery expression into an XmlQueryExpression (page 362) object. You can then run the XQuery expression repeatedly using XmlQueryExpression::execute (page 364).

Use this method to compile and evaluate XQuery expressions against your XmlContainer (page 16) and XmlDocument (page 135) objects any time you want to evaluate the expression more than once.

Note that the scope of the query provided here can be restricted using one of the XQuery navigational functions. For example:

```
"collection('mycontainer.dbxml')/foo"
```

or:

```
"doc('dbxml:/mycontainer.dbxml/mydoc.xml')/foo/@attr1='bar'"
```

The scope of a query can also be controlled by passing an appropriate contextItem object to XmlQueryExpression::execute (page 364).

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

xquery

The XQuery query string to compile.

context

The XmlQueryContext (page 341) to use for this query.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::query

```
#include <DbXml.hpp>

XmlResults XmlManager::query(const std::string &xquery,
    XmlQueryContext &context, u_int32_t flags = 0)

XmlResults XmlManager::query(XmlTransaction &txn,
    const std::string &xquery,
    XmlQueryContext &context, u_int32_t flags = 0)
```

Executes a query in the context of the `XmlManager` object. This method is the equivalent of calling XmlManager::prepare (page 315) and then XmlQueryExpression::execute (page 364) on the prepared query.

The scope of the query can be restricted using one of the XQuery navigational functions. For example:

```
"collection('mycontainer.dbxml')/foo"
```

or:

```
"doc('dbxml:/mycontainer.dbxml/mydoc.xml')/foo/@attr1='bar'"
```

The scope of a query can also be controlled by passing an appropriate contextItem object to XmlQueryExpression::execute (page 364).

Note that this method is suitable for performing one-off queries. If you want to execute a query more than once, you should use XmlManager::prepare (page 315) to compile the expression, and then use XmlQueryExpression::execute (page 364) to run it.

This method returns an XmlResults (page 380) object. You then iterate over the results set contained in that object using XmlResults::next (page 390) and XmlResults::previous (page 392).

For more information on querying containers and documents, see the Berkeley DB XML Getting Started Guide.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

xquery

The XQuery query string.

context

The XmlQueryContext (page 341) to use for this query.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DBXML_DOCUMENT_PROJECTION`

  When parsing a document in order to execute a query, use static analysis of the query to materialize only those portions of the document relevant to the query. This can significantly enhance performance of queries against documents from containers of type `XmlContainer::WholedocContainer` and documents not in a container. It should not be used if arbitrary navigation of the resulting nodes is to be performed, as not all nodes in the original document will be present and unexepcted results could be returned. This flag has no effect on documents in containers of type `XmlContainer::NodeContainer`.

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_WELL_FORMED_ONLY`

  Force the use of a scanner that will neither validate nor read schema or dtds associated with the document during parsing. This is efficient, but can cause parsing errors if the document references information that might have come from a schema or dtd, such as entity references.

Errors

The `XmlManager::query` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_VALUE

Invalid flags to method `XmlManager::query`

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::registerCompression

```
#include <DbXml.hpp>

void XmlManager::registerCompression(const char *name,
        XmlCompression &compression)
```

Identifies an XmlCompression (page 12) instance to be used for document compression in whole document storage containers.

Parameters

name

The name of the compression instance. This name is used to identify a compression instance when creating a container using the configuration value set by XmlContainerConfig::setCompressionName (page 102).

compression

An instance of XmlCompression (page 12) to be used to compress XML documents when they are inserted into the container, and decompress them when they are retrieved.

Errors

The `XmlManager::registerCompression` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_VALUE

Name has already been registered

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::registerResolver

```
#include <DbXml.hpp>

void XmlManager::registerResolver(const XmlResolver &resolver)
```

Identifies an XmlResolver (page 371) object to be used for file resolution. This object is used by the internal XML document parser to locate data based on URIs, or public and system ids. Custom XmlResolver (page 371) objects can be created by applications to provide a mechanism to name and retrieve collections, documents, and XML entities external to Berkeley DB XML.

Parameters

resolver

The `XmlResolver` instance to be used for file resolution.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::reindexContainer

```
#include <DbXml.hpp>

void XmlManager::reindexContainer(
 const std::string &name, XmlUpdateContext &context)

void XmlManager::reindexContainer(
 XmlTransaction &txn, const std::string &name,
 XmlUpdateContext &context)

void XmlManager::reindexContainer(
 const std::string &name, XmlUpdateContext &context,
 const XmlContainerConfig &flags)

void XmlManager::reindexContainer(
 XmlTransaction &txn, const std::string &name,
 XmlUpdateContext &context,
 const XmlContainerConfig &flags)
```

Reindex an entire container. The container should be backed up prior to using this method, as it destroys existing indices before reindexing. If the operation fails, and your container is not backed up, you may lose information.

Use this call to change the type of indexing used for a container between document-level indices and node-level indices. This method can take a very long time to execute, depending on the size of the container, and should not be used casually.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The path to the container to be reindexed.

context

The update context to use for the reindex operation.

flags

Set XmlContainerConfig::setIndexNodes (page 106) to `On` to change the container's index type to node indexes, and set XmlContainerConfig::setIndexNodes (page 106) to `Off` to change the index type to document indexes. Set XmlContainerConfig::setStatistics (page 114) to `On` to add a structural statistics database to the container during reindexing, and set XmlContainerConfig::setStatistics (page 114) to `Off` to remove an existing structural statistics database.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::removeContainer

```
#include <DbXml.hpp>

void XmlManager::removeContainer(XmlTransaction &txn,
        const std::string &name);
```

The `XmlManager::removeContainer` method removes the underlying file for the container from the file system.

The container must be closed; the system throws an exception if the container is open.

The container must have been opened at least once; the system throws an exception if the underlying file has not yet been created.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The name of the container to be removed.

Errors

The `XmlManager::removeContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::renameContainer

```
#include <DbXml.hpp>

void XmlManager::renameContainer(XmlTransaction &txn, const std::string
&oldName, const std::string &newName);
```

The `XmlManager::renameContainer` method renames the container's underlying file.

The container must be closed; the system throws an exception if the container is open.

The container must have been opened at least once; the system throws an exception if the underlying file has not yet been created.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

oldName

The name of the container whose name you want to change.

newName

The new container name.

Errors

The `XmlManager::renameContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::setDefaultContainerConfig

```
#include <DbXml.hpp>

void XmlManager::setDefaultContainerConfig(XmlContainerConfig &config)
```

Sets the default container configuration used for containers opened and created by this `XmlManager` object. If a form of XmlManager::createContainer (page 281) or XmlManager::openContainer (page 310) is used that takes an `XmlContainerConfig` argument, the settings provided using this method are ignored. This will only affect containers opened or created after the value is set.

Parameters

config

The `XmlContainerConfig` object.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::setDefaultContainerFlags

```
#include <DbXml.hpp>

void XmlManager::setDefaultContainerFlags(const XmlContainerConfig &flags)

XmlContainerConfig XmlManager::getDefaultContainerFlags() const
```

Sets the default flags used for containers opened and created by this `XmlManager` object. If a form of XmlManager::createContainer (page 281) or XmlManager::openContainer (page 310) is used that takes a flags argument, the settings provided using this method are ignored.

Parameters

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_CREATE`

  If the container does not currently exist, create it.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_EXCL`

  Return an error if the container already exists. This flag is only meaningful when specified with the `DB_CREATE` flag.

- `DB_MULTIVERSION`

  Open the database with support for multiversion concurrency control. This will cause updates to the container to follow a copy-on-write protocol, which is required to support snapshot isolation. This flag requires that the container be transactionally protected during its open.

- `DB_NOMMAP`

  Do not map this container into process memory (see the `DbEnv::set_mp_mmapsize()` method for further information).

- `DB_RDONLY`

  Open the container for reading only. Any attempt to modify items in the container will fail, regardless of the actual permissions of any underlying files.

- `DB_THREAD`

Cause the container handle to be *free-threaded*; that is, concurrently usable by multiple threads in the address space.

- `DBXML_CHKSUM`

  Do checksum verification of pages read into the cache from the backing filestore. Berkeley DB XML uses the SHA1 Secure Hash Algorithm if encryption is configured and a general hash algorithm if it is not.

- `DBXML_ENCRYPT`

  Encrypt the database using the cryptographic password specified to `DbEnv::set_encrypt()`.

- `DB_TXN_NOT_DURABLE`

  If set, Berkeley DB XML will not write log records for this database. This means that updates of this database exhibit the ACI (atomicity, consistency, and isolation) properties, but not D (durability); that is, database integrity will be maintained, but if the application or system fails, integrity will not persist. The database file must be verified and/or restored from backup after a failure.

- `DBXML_INDEX_NODES`

  Causes the indexer to create index targets that reference nodes rather than documents. This allows index lookups during query processing to more efficiently find target nodes and avoid walking the document tree. It can apply to both container types, and is the default for containers of type `XmlContainer::NodeContainer`.

- `DBXML_NO_INDEX_NODES`

  Causes the indexer to create index targets that reference documents rather than nodes. This can be more desirable for simple queries that only need to return documents and do relatively little navigation during querying. It can apply to both container types, and is the default for containers of type `XmlContainer::WholedocContainer`.

- `DBXML_STATISTICS`

  Causes the container to be created to include structural statistics information, which is very useful for cost based query optimisation. Containers created with these statistics will take longer to load and update, since the statistics must also be updated. This is the default.

- `DBXML_NO_STATISTICS`

  Causes the container to be created without structural statistics information – by default structural statistics are created.

- `DBXML_TRANSACTIONAL`

  Cause the container to support transactions. If this flag is set, an XmlTransaction (page 407) object may be used with any method that supports transactional protection. Also, if this flag is used, and if an XmlTransaction (page 407) object is not provided to a method that modifies an XmlContainer (page 16) or XmlDocument (page 135) object, then auto commit is automatically used for the operation.

- `DBXML_ALLOW_VALIDATION`

  When loading documents into the container, validate the XML if it refers to a DTD or XML Schema.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::setDefaultContainerType

```
#include <DbXml.hpp>

void XmlManager::setDefaultContainerType(XmlContainer::ContainerType type)

XmlContainer::ContainerType XmlManager::getDefaultContainerType() const
```

Sets the default type used for containers opened and created by this `XmlManager` object. If a form of XmlManager::createContainer (page 281) or XmlManager::openContainer (page 310) is used that takes a type argument, the settings provided using this method are ignored.

Parameters

type

The type of container to create.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::setDefaultPageSize

```
#include <DbXml.hpp>

void XmlManager::setDefaultPageSize(u_int32_t pageSize)

u_int32_t XmlManager::getDefaultPageSize()
```

The `XmlManager::setDefaultPageSize` method sets the size of the pages used to store documents in the database. The size is specified in bytes in the range 512 bytes to 64K bytes. The system selects a page size based on the underlying file system I/O block size if one is not explicitly set by the application. The default page size has a lower limit of 512 bytes and an upper limit of 16K bytes. Documents that are larger than a single page are stored on multiple pages.

The `XmlManager::setDefaultPageSize` method will only affect containers created after it is set. It has no effect on existing containers.

Parameters

pagesize

The page size in bytes.

Errors

The `XmlManager::setDefaultPageSize` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The page size may only be set for new containers.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::setDefaultSequenceIncrement

```
#include <DbXml.hpp>

void XmlManager::setDefaultSequenceIncrement(u_int32_t increment)

u_int32_t XmlManager::getDefaultSequenceIncrement()
```

Sets the integer increment to be used when pre-allocating document ids for new documents created by XmlContainer::putDocument (page 57).

Every document added to an `XmlContainer` is assigned an internal unique ID, and BDB XML performs an internal database operation to obtain these IDs. In order to increase database concurrency and improve performance of ID allocation, BDB XML pre-allocates a sequence of these numbers. The size of this sequence is determined by the value specified here. The default ID sequence size is 5.

Be aware that when a container is closed, any unused IDs in the current sequence are lost. Under some extreme cases, this can result in a container to which documents can no longer be added. For example, setting this value to a very large number (such as, say, 1 million) and then repeatedly opening and closing the container while adding a few documents will eventually cause the container to run out of IDs. Once out of IDs, the container will never again be able to accept new documents. The maximum number of IDs that a container has available to it is currently 4 billion.

You should almost always leave this value alone. However, if you are loading a large number of documents to a container all at once, you may find a small performance benefit to setting the sequence number to a larger value. If you do this, be aware that this value is persistent across container opens, so you should take care to reset the value to its default once you are done loading the documents.

Parameters

increment

The increment to use for pre-allocated IDs.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::setImplicitTimezone

```
#include <DbXml.hpp>

void XmlManager::setImplicitTimezone(int tz)
```

Sets the implicit timezone to be used for queries referring to dates and times in the context of the XmlManager (page 274).

Parameters

tz

The timezone as an offset in minutes from GMT.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::truncateContainer

```
#include <DbXml.hpp>

void XmlManager::truncateContainer(
 const std::string &name, XmlUpdateContext &context)

void XmlManager::truncateContainer(
 XmlTransaction &txn, const std::string &name,
 XmlUpdateContext &context)

void XmlManager::truncateContainer(
 const std::string &name, XmlUpdateContext &context,
 const XmlContainerConfig &flags)

void XmlManager::truncateContainer(
 XmlTransaction &txn, const std::string &name,
 XmlUpdateContext &context,
 const XmlContainerConfig &flags)
```

Truncates all of the databases in the container using `Db::truncate`.

The container must be closed; the system throws an exception if the container is open.

Parameters

txn

If the operation is to be transaction-protected, the `txn` parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

name

The name of the container to be truncated.

context

The XmlUpdateContext (page 415) object to be used for this operation.

flags

This parameter is unused.

Errors

The XmlManager::upgradeContainer (page 335) method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::upgradeContainer

```
#include <DbXml.hpp>

void XmlManager::upgradeContainer(const std::string &name,
    XmlUpdateContext &context);
```

Upgrades the container from a previous version of Berkeley DB XML, or Berkeley DB, to the current version. A Berkeley DB upgrade is first performed using the `Db::upgrade` method, and then the Berkeley DB XML container is upgraded. If no upgrade is needed, then no changes are made.

Note

Container upgrades are done in place and are destructive. For example, if pages need to be allocated and no disk space is available, the container may be left corrupted. Backups should be made before containers are upgraded. See Upgrading databases for more information.

The container must be closed; the system throws an exception if the container is open.

Parameters

name

The name of the container to be upgraded.

context

The XmlUpdateContext (page 415) object to be used for this operation.

Errors

The `XmlManager::upgradeContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

XmlManager::verifyContainer

```
#include <DbXml.hpp>

void XmlManager::verifyContainer(const std::string &name,
    std::ostream *out, u_int32_t flags);
```

Checks that the container data files are not corrupt, and optionally writes the salvaged container data to the specified output stream.

The container must be closed; the system throws an exception if the container is open.

The container must have been opened at least once; the system throws an exception if the underlying files have not yet been created.

Parameters

name

The name of the container to be verified.

out

The stream the salvaged container data is to be dumped to.

flags

Flags must be set to zero, `DB_SALVAGE`, or `DB_SALVAGE` and `DB_AGGRESSIVE`. Please refer to the Berkeley DB reference manual for a full discussion of these values.

Errors

The `XmlManager::verifyContainer` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

CONTAINER_OPEN

The container is open.

DATABASE_ERROR

An error occurred in an underlying Berkeley DB database. The XmlException::getDbErrno (page 210) method will return the error code for the error.

Class

XmlManager (page 274)

See Also

XmlManager Methods (page 277)

Chapter 19. XmlMetaDataIterator

```
#include <DbXml.hpp>

XmlMetaDataIterator::XmlMetaDataIterator()
XmlMetaDataIterator::XmlMetaDataIterator(const XmlMetaDataIterator&)
XmlMetaDataIterator &operator = (const XmlMetaDataIterator &)
virtual XmlMetaDataIterator::~XmlMetaDataIterator()
```

Provides an iterator over an XmlDocument (page 135)'s metadata. Metadata is set on a document with XmlDocument::setMetaData (page 149). You can also use XmlDocument::getMetaData (page 142) to return a specific metadata item.

This object is instantiated using XmlDocument::getMetaDataIterator (page 143). This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlMetaDataIterator Methods

XmlMetaDataIterator Methods	Description
XmlMetaDataIterator::next	Retrieve the next metadata item.
XmlMetaDataIterator::reset	Retrieve the first metadata item.

XmlMetaDataIterator::next

```
#include <DbXml.hpp>

bool XmlMetaDataIterator::next(
    std::string &uri, std::string &name, XmlValue &value)
```

Returns the next item in the XmlDocument (page 135)'s metadata list. If there is no next item, this method returns false. Otherwise, it returns true.

Parameters

uri

Contains the URI used for the metadata item retrieved by this method.

name

Contains the name used for the metadata item retrieved by this method.

value

Contains the XmlValue (page 416) contained by the metadata item retrieved by this method.

Class

XmlMetaDataIterator (page 337)

See Also

XmlMetaDataIterator Methods (page 338)

XmlMetaDataIterator::reset

```
#include <DbXml.hpp>

void XmlMetaDataIterator::reset()
```

Sets the iterator to the beginning of the XmlDocument (page 135)'s metadata list.

Class

XmlMetaDataIterator (page 337)

See Also

XmlMetaDataIterator Methods (page 338)

Chapter 20. XmlQueryContext

```
#include <DbXml.hpp>

class DbXml::XmlQueryContext {
public:
 XmlQueryContext();
 XmlQueryContext(const XmlQueryContext &);
 ~XmlQueryContext();
 XmlQueryContext &operator = (const XmlQueryContext &)
 ...
};
```

The `XmlQueryContext` class encapsulates the context within which a query is performed against an XmlContainer (page 16). The context includes namespace mappings, variable bindings, and flags that indicate how the query result set should be determined and returned to the caller. Multiple queries can be executed within the same `XmlQueryContext`; however, XmlQueryContext is not thread-safe, and can only be used by one thread at a time.

`XmlQueryContext` objects are instantiated using XmlManager::createQueryContext (page 293).

`XmlQueryContext` allows you to define whether queries executed within the context are to be evaluated lazily or eagerly, and whether the query is to return live or dead values. For detailed descriptions of these parameters see XmlQueryContext::setReturnType (page 360) and XmlQueryContext::setEvaluationType (page 357). Note that these values are also set when you create a query context using XmlManager::createQueryContext (page 293).

The XQuery syntax permits expressions to refer to namespace prefixes, and to define them. The `XmlQueryContext` class provides namespace management methods so that the caller may manage the namespace prefix to URI mapping outside of a query. By default the prefix "dbxml" is defined to be "http://www.sleepycat.com/2002/dbxml".

The XQuery syntax also permits expressions to refer to externally defined variables. The `XmlQueryContext` class provides methods that allow the caller to manage the externally-declared variable to value bindings.

A copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body. This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlQueryContext Methods

XmlQueryContext Methods	Description
XmlQueryContext::clearNamespaces	Remove all namespaces.
XmlQueryContext::getBaseURI	Gets the Base URI.
XmlQueryContext::getDebugListener	Gets the XmlDebugListener.
XmlQueryContext::getDefaultCollection	Get the default collection for fn:collection().
XmlQueryContext::getEvaluationType	Get the evaluation type.
XmlQueryContext::getNamespace	Retrieve a namespace URI.
XmlQueryContext::getQueryTimeoutSeconds	Get query timeout value.
XmlQueryContext::getReturnType	Get the return type.
XmlQueryContext::getVariableValue	Return the variable's value.
XmlQueryContext::interruptQuery	Interrupt a running query.
XmlQueryContext::removeNamespace	Remove the specified namespace.
XmlQueryContext::setBaseURI	Sets the Base URI.
XmlQueryContext::setDebugListener	Sets the XmlDebugListener.
XmlQueryContext::setDefaultCollection	Set default collection for fn:collection().
XmlQueryContext::setEvaluationType	Set the evaluation type.
XmlQueryContext::setNamespace	Add a namespace.
XmlQueryContext::setQueryTimeoutSeconds	Set query timeout value.
XmlQueryContext::setReturnType	Set the return type.
XmlQueryContext::setVariableValue	Set an external XQuery variable.

XmlQueryContext::clearNamespaces

```
#include <DbXml.hpp>

void XmlQueryContext::clearNamespaces();
```

The `XmlQueryContext::clearNamespaces` method removes all namespace mappings from the query context.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getBaseURI

```
#include <DbXml.hpp>

std::string XmlQueryContext::getBaseURI();
```

Returns the base URI used for relative paths in query expressions. this URI is set using the XmlQueryContext::setBaseURI (page 354) method.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getDebugListener

```
#include <DbXml.hpp>

XmlDebugListener * XmlQueryContext::getDebugListener() const;
```

Retrieves the XmlDebugListener (page 128) associated with this XmlQueryContext (page 341), if any. The debug listener may be associated with the XmlQueryContext (page 341) using the XmlQueryContext::setDebugListener (page 355) method.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getDefaultCollection

```
#include <DbXml.hpp>

std::string XmlQueryContext::getDefaultCollection() const;
```

Returns the URI of the default collection. This value can be set using the XmlQueryContext::setDefaultCollection (page 356) method.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getEvaluationType

```
#include <DbXml.hpp>

EvaluationType XmlQueryContext::getEvaluationType();
```

Discover the evaluation type defined for this XmlQueryContext (page 341).

Returns the evaluation type defined for this XmlQueryContext (page 341).

The evaluation type can be set using the XmlQueryContext::setEvaluationType (page 357) method.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getNamespace

```
#include <DbXml.hpp>

std::string XmlQueryContext::getNamespace()
```

Returns the current namespace prefix. This prefix is set using the XmlQueryContext::setNamespace (page 358) method.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getQueryTimeoutSeconds

```
#include <DbXml.hpp>

unsigned int XmlQueryContext::getQueryTimeoutSeconds() const;
```

Retrieves the current query timeout value. This value is set using the XmlQueryContext::setQueryTimeoutSeconds (page 359) method.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getReturnType

```
#include <DbXml.hpp>

ReturnType XmlQueryContext::getReturnType();
```

Retrieves the current `ReturnType` set for the XmlQueryContext (page 341). This value can be set using the XmlQueryContext::setReturnType (page 360) method.

Parameters

type

The type parameter specifies which of documents or values to return, and must be set to one of the following values:

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::getVariableValue

```
#include <DbXml.hpp>

bool XmlQueryContext::getVariableValue(
    const std::string &name, XmlValue &value);

bool XmlQueryContext::getVariableValue(
    const std::string &name, XmlResults &value);
```

Returns the value that is bound to a specified variable. If there is no value binding, then this method returns `false` and `value` is set to a null value (`XmlValue::isNull()` or `XmlResults::isNull()` returns `true`).

Parameters

name

The name of the bound variable.

value

The value bound to the named variable. If `value` is an XmlResults (page 380) object, a sequence of values is bound to the variable.

Errors

The `XmlQueryContext::getVariableValue` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_VALUE

It is not valid to use the `XmlValue` get method for variables with more than one value

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::interruptQuery

```
#include <DbXml.hpp>

void XmlQueryContext::interruptQuery();
```

The `XmlQueryContext::interruptQuery` method interrupts a running query that was started using this object.

This call must be made in the context of another thread of control in the application. The query will terminate and throw the an XmlException (page 206) with the Exception code `XmlException::OPERATION_INTERRUPTED`.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::removeNamespace

```
#include <DbXml.hpp>

void XmlQueryContext::removeNamespace(const std::string &prefix);
```

The `XmlQueryContext::removeNamespace` method removes the namespace prefix to URI mapping for the specified prefix. A call to this method with a prefix that has no existing mapping is ignored.

Parameters

prefix

The namespace prefix to be removed.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setBaseURI

```
#include <DbXml.hpp>

void XmlQueryContext::setBaseURI(const std::string &baseURI)
```

Sets the base URI used for relative paths in query expressions. For example, a base URI of `file:///export/expression/`, and a relative path of `../another/expression`, resolves to `file:///export/another/expression`.

Parameters

baseURI

The base URI, as a string.

Errors

The `XmlQueryContext::setBaseURI` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_ERROR

The base URI string is not a valid URI scheme. It must begin with a string of the form, scheme:/.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setDebugListener

```
#include <DbXml.hpp>

void XmlQueryContext::setDebugListener(XmlDebugListener *listener);
```

Allows the application to associate an XmlDebugListener (page 128) with a query context in order to debug queries.

In order to prepare a query that contains debugging information an `XmlDebugListener` must be set on the `XmlQueryContext` used when the query is prepared. A different `XmlDebugListener` object (or none) can subsequently be set for query evaluation.

Parameters

listener

An instance of XmlDebugListener (page 128). The object is owned and managed by the caller.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setDefaultCollection

```
#include <DbXml.hpp>

void XmlQueryContext::setDefaultCollection(const std::string &uri);
```

The default collection is that which is used by fn:collection() without any arguments in an XQuery expression.

Parameters

uri

A URI specifying the name of the collection.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setEvaluationType

```
#include <DbXml.hpp>

void XmlQueryContext::setEvaluationType(EvaluationType type);
```

Allows the application to set the query evaluation type to "eager" or "lazy". Eager evaluation means that the whole query is executed and its resultant values derived and stored in-memory before evaluation of the query is completed. Lazy evaluation means that minimal processing is performed before the query is completed, and the remaining processing is deferred until the result set is enumerated. As each call to XmlResults::next (page 390) is called the next resultant value is determined.

Parameters

type

The evaluation type must be specified as either:

- `XmlQueryContext::Eager`

  The query is executed and its resultant values are derived and stored in-memory before evaluation of the query is completed.

- `XmlQueryContext::Lazy`

  Minimal processing is performed before evaluation of the query is completed, and the remaining processing is deferred until the result set is enumerated.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setNamespace

```
#include <DbXml.hpp>

void XmlQueryContext::setNamespace(const std::string &prefix,
                                   const std::string &uri)
```

Maps the specified URI to the specified namespace prefix.

Parameters

prefix

The namespace prefix. If this parament is the empty string, the uri will be used as the default element namespace.

uri

The namespace URI.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setQueryTimeoutSeconds

```
#include <DbXml.hpp>

void XmlQueryContext::setQueryTimeoutSeconds(unsigned int seconds);
```

The query timeout will cause a query using this XmlQueryContext (page 341) to terminate if the query time exceeds the timeout value. The execution of the query will throw an XmlException (page 206) with the ExceptionCode `XmlException::OPERATION_TIMEOUT` if the timeout occurs.

Parameters

seconds

The timeout, in seconds.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setReturnType

```
#include <DbXml.hpp>

void XmlQueryContext::setReturnType(ReturnType type);
```

The `XmlQueryContext::setReturnType` method allows the application to define the return type of the query result values.

Parameters

type

The type parameter specifies which of documents or values to return, and must be set to one of the following values:

- `XmlQueryContext::LiveValues`

  A reference to the data stored in Berkeley DB XML is returned.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

XmlQueryContext::setVariableValue

```
#include <DbXml.hpp>

void XmlQueryContext::setVariableValue(
    const std::string &name, const XmlValue &value);

void XmlQueryContext::setVariableValue(
    const std::string &name, const XmlResults &value);
```

Creates an externally-declared XQuery variable by binding the specified value, or sequence of values, to the specified variable name.

This method may be called at any time during the life of the application.

Parameters

name

The name of the variable to bind. Within the XQuery query, the variable can be referenced using the normal `$name` syntax.

value

The value to bind to the named variable. If `value` is an XmlResults (page 380) object, a sequence of values is bound to the variable.

Class

XmlQueryContext (page 341)

See Also

XmlQueryContext Methods (page 342)

Chapter 21. XmlQueryExpression

```
#include <DbXml.hpp>

class DbXml::XmlQueryExpression {
public:
 XmlQueryExpression ()
 virtual ~XmlQueryExpression ()
 XmlQueryExpression (const XmlQueryExpression &)
 XmlQueryExpression & operator= (const XmlQueryExpression &)
 ...
};
```

An `XmlQueryExpression` represents a parsed XQuery expression, and is created by a call to XmlManager::prepare (page 315). Parsed XQuery expressions are useful because they allow the cost of query parsing and optimization to be amortized over many evaluations. If indices of a container that may be referenced by a parsed XQuery expression are modified (e.g. add/drop an index) the `XmlQueryExpression` object should be regenerated by the application.

The copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body. This object is free threaded, and can be safely shared among threads in an application.

XmlQueryExpression Methods

XmlQueryExpression Methods	Description
XmlQueryExpression::execute	Evaluate the query.
XmlQueryExpression::getQuery	Return the query as a string.
XmlQueryExpression::getQueryPlan	Return the query plan as a string.
XmlQueryExpression::isUpdateExpression	Returns true if the query is an updating expression.

XmlQueryExpression::execute

```
#include <DbXml.hpp>

XmlResults XmlQueryExpression::execute(
    XmlQueryContext &queryContext, u_int32_t flags = 0)

XmlResults XmlQueryExpression::execute(
    XmlTransaction &txn, XmlQueryContext &queryContext,
    u_int32_t flags = 0)

XmlResults XmlQueryExpression::execute(const XmlValue &contextItem,
    XmlQueryContext &queryContext, u_int32_t flags = 0)

XmlResults XmlQueryExpression::execute(XmlTransaction &txn,
    const XmlValue &contextItem, XmlQueryContext &queryContext,
    u_int32_t flags = 0)
```

Evaluates (runs) an XQuery query that was previously prepared by XmlManager::prepare (page 315) and returns an XmlResults (page 380) set.

There are two basic forms of this method: one that takes an XmlValue (page 416) object, and another that does not. For methods that do not take an `XmlValue`, the XQuery must restrict the scope of the query using either the `collection()` or the `doc()` XQuery navigation functions, or an exception is thrown.

For those forms of this method that take an XmlValue (page 416), the query is applied against that object.

Evaluation without an XmlValue object

```
#include <DbXml.hpp>

XmlResults XmlQueryExpression::execute(
    XmlQueryContext &queryContext, u_int32_t flags = 0)

XmlResults XmlQueryExpression::execute(
    XmlTransaction &txn, XmlQueryContext &queryContext,
    u_int32_t flags = 0)
```

Evaluates the XQuery expression against the containers and documents identified by the query, from within the scope of the provided XmlTransaction (page 407) object. To use this form of the method, you must restrict the scope of the query using either the `collection()` or the `doc()` XQuery navigation functions, or an exception is thrown.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

queryContext

The XmlQueryContext (page 341) to use for this evaluation.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_RMW`

  Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DBXML_DOCUMENT_PROJECTION`

  When parsing a document in order to execute a query, use static analysis of the query to materialize only those portions of the document relevant to the query. This can significantly enhance performance of queries against documents from containers of type `XmlContainer::WholedocContainer` and documents not in a container. It should not be used if arbitrary navigation of the resulting nodes is to be performed, as not all nodes in the original document will be present and unexepcted results could be returned. This flag has no effect on documents in containers of type `XmlContainer::NodeContainer`.

- `DBXML_WELL_FORMED_ONLY`

  Force the use of a scanner that will neither validate nor read schema or dtds associated with the document during parsing. This is efficient, but can cause parsing errors if the document references information that might have come from a schema or dtd, such as entity references.

- `DBXML_NO_AUTO_COMMIT`

Do not create and automatically commit a transaction if one is not provided to this method. A query that performs an update under a transactional environment will automatically be transaction protected unless this flag is specified. This flag is only necessary if the update will make changes to both transactional and non-transactional containers.

Evaluation with an XmlValue object

```
#include <DbXml.hpp>

XmlResults XmlQueryExpression::execute(const XmlValue &contextItem,
    XmlQueryContext &queryContext, u_int32_t flags = 0)

XmlResults XmlQueryExpression::execute(XmlTransaction &txn,
    const XmlValue &contextItem, XmlQueryContext &queryContext,
    u_int32_t flags = 0)
```

Evaluates the XQuery expression against the provided context item.

Parameters are:

txn

If the operation is to be transaction-protected, this parameter is an XmlTransaction (page 407) handle returned from XmlManager::createTransaction (page 296).

contextItem

The XmlValue (page 416) object to perform the query against.

queryContext

The XmlQueryContext (page 341) to use for this evaluation.

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_RMW`

Acquire write locks instead of read locks when doing the read, if locking is configured. Setting this flag can eliminate deadlock during a read-modify-write cycle by acquiring the write lock during the read part of the cycle so that another thread of control acquiring a read lock for the same item, in its own read-modify-write cycle, will not result in deadlock.

- `DBXML_LAZY_DOCS`

  Retrieve the document lazily. That is, retrieve document content and document metadata only on an as needed basis when reading the document.

- `DBXML_DOCUMENT_PROJECTION`

  When parsing a document in order to execute a query, use static analysis of the query to materialize only those portions of the document relevant to the query. This can significantly enhance performance of queries against documents from containers of type `XmlContainer::WholedocContainer` and documents not in a container. It should not be used if arbitrary navigation of the resulting nodes is to be performed, as not all nodes in the original document will be present and unexepcted results could be returned. This flag has no effect on documents in containers of type `XmlContainer::NodeContainer`.

- `DBXML_WELL_FORMED_ONLY`

  Force the use of a scanner that will neither validate nor read schema or dtds associated with the document during parsing. This is efficient, but can cause parsing errors if the document references information that might have come from a schema or dtd, such as entity references.

- `DBXML_NO_AUTO_COMMIT`

  Do not create and automatically commit a transaction if one is not provided to this method. A query that performs an update under a transactional environment will automatically be transaction protected unless this flag is specified. This flag is only necessary if the update will make changes to both transactional and non-transactional containers.

Errors

The `XmlQueryExpression::execute` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_VALUE

You provided an invalid value to the `flags` parameter.

Class

XmlQueryExpression (page 362)

See Also

XmlQueryExpression Methods (page 363)

XmlQueryExpression::getQuery

```
#include <DbXml.hpp>

std::string XmlQueryExpression::getQuery() const
```

Returns the query as a string.

Class

XmlQueryExpression (page 362)

See Also

XmlQueryExpression Methods (page 363)

XmlQueryExpression::getQueryPlan

```
#include <DbXml.hpp>

std::string XmlQueryExpression::getQueryPlan() const
```

Returns the query plan as a string.

Class

XmlQueryExpression (page 362)

See Also

XmlQueryExpression Methods (page 363)

XmlQueryExpression::isUpdateExpression

```
#include <DbXml.hpp>

bool XmlQueryExpression::isUpdateExpression() const
```

Return true if the query is an updating expression.

Class

XmlQueryExpression (page 362)

See Also

XmlQueryExpression Methods (page 363)

Chapter 22. XmlResolver

```
#include <DbXml.hpp>

virtual XmlResolver::~XmlResolver()
```

Provides an unimplemented base class that is used for file resolution policy. Implementations of this class are used for URI resolution, or for public and system ids resolution. `XmlResolver` implementations are identified for use using XmlManager::registerResolver (page 320).

The `XmlResolver` class allows applications to provide named access to application-specific objects, such as documents, collections of documents, DTDs, and XML schema. Berkeley DB XML can resolve references to names within a container, or a file system; applications can create `XmlResolver` instances that can resolve entities in other locations.

For an example of an `XmlResolver` implementation, see the Berkeley DB XML distribution file `test/cpp/util/TestResolver.cpp`.

If an application uses multiple threads, custom implementations of `XmlResolver` must be free threaded, and allow multiple, simultaneous calls for resolution.

XmlResolver Methods

XmlResolver Methods	Description
XmlResolver::resolveCollection	Resolve URI to an XmlResults.
XmlResolver::resolveDocument	Resolve URI to an XmlValue.
XmlResolver::resolveEntity	Resolve an entity to an XmlInputStream.
XmlResolver::resolveExternalFunction	Resolve URI and name to an XmlExternalFunction.
XmlResolver::resolveModule	Resolve an XQuery module reference.
XmlResolver::resolveModuleLocation	Resolve an XQuery module namespace to locations.
XmlResolver::resolveSchema	Resolve schema to an XmlInputStream.

XmlResolver::resolveCollection

```
#include <DbXml.hpp>

virtual bool XmlResolver::resolveCollection(
    XmlTransaction *txn, XmlManager &mgr,
    const std::string &uri, XmlResults &result) const
```

When implemented, should resolve a URI to an XmlResults (page 380). If the URI cannot be resolved by this resolver, this method should return false. Otherwise, it should return true.

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

uri

The URI to resolve.

result

The `XmlResult` object in which the results of the resolution are to be placed.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

XmlResolver::resolveDocument

```
#include <DbXml.hpp>

virtual bool XmlResolver::resolveDocument(
    XmlTransaction *txn, XmlManager &mgr,
    const std::string &uri, XmlValue &result) const
```

When implemented, should resolve a URI to an XmlValue (page 416). If the URI cannot be resolved by this resolver, this method should return false. Otherwise, it should return true.

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

uri

The URI to resolve.

result

The `XmlValue` object in which the results of the resolution are to be placed.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

XmlResolver::resolveEntity

```
#include <DbXml.hpp>

virtual XmlInputStream *XmlResolver::resolveEntity(
    XmlTransaction *txn, XmlManager &mgr, const std::string &systemId,
    const std::string &publicId, std::string &result) const
```

When implemented, should resolve a System ID and Public ID to a new XmlInputStream (page 270). If the IDs cannot be resolved by this resolver, this method should return NULL. The `XmlInputStream` object will be deleted by the caller.

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

systemId

The System ID to resolve.

publicId

The Public ID to resolve.

result

The string in which the results of the resolution are to be placed.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

XmlResolver::resolveExternalFunction

```
#include <DbXml.hpp>

virtual XmlExternalFunction *XmlResolver::resolveExternalFunction(
    XmlTransaction *txn, XmlManager &mgr, const std::string &uri,
    const std::string &name, size_t numberOfArgs) const
```

When implemented this method resolves a unigue combination of function URI, function name and number of arguments into an instance of XmlExternalFunction (page 217) used to implement an XQuery extension function. If the URI, name and arguments cannot be resolved NULL is returned. It may return a singleton or a new instance. Memory management of the returned object depends on the implementation of the `XmlResolver` and the XmlExternalFunction (page 217) instance. Singletons are usually owned by `XmlResolver` while new instances must be deleted by XmlExternalFunction::close() (page 220).

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

uri

The URI of the XQuery function being resolved.

name

The string name of the XQuery function being resolved.

numberOfArgs

The number of arguments passed to the XQuery function being resolved.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

XmlResolver::resolveModule

```
#include <DbXml.hpp>

virtual XmlInputStream *XmlResolver::resolveModule(
    XmlTransaction *txn, XmlManager &mgr,
    const std::string &moduleLocation,
    const std::string &nameSpace) const
```

When implemented, should resolve a module location (URI) and namespace to a new XmlInputStream (page 270). If the location and namespace cannot be resolved by this resolver, this method should return NULL. The `XmlInputStream` object will be deleted by the caller.

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

moduleLocation

The URI for the module resolve.

nameSpace

The namespace of the module to resolve.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

XmlResolver::resolveModuleLocation

```
#include <DbXml.hpp>

virtual bool XmlResolver::resolveModuleLocation(
    XmlTransaction *txn, XmlManager &mgr,
    const std::string &nameSpace, XmlResults &result) const
```

When implemented, should resolve a module namespace to a list of strings that are locations for the files that comprise the module. The strings are returned as XmlValue (page 416) objects in the XmlResults (page 380). If the module cannot be resolved by this resolver, this method should return false. Otherwise, it should return true.

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

nameSpace

The nameSpace of the module to resolve.

result

The `XmlResult` object in which the results of the resolution are to be placed.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

XmlResolver::resolveSchema

```
#include <DbXml.hpp>

virtual XmlInputStream *XmlResolver::resolveSchema(
    XmlTransaction *txn, XmlManager &mgr,
    const std::string &schemaLocation,
    const std::string &nameSpace, std::string &result) const
```

When implemented, should resolve schema location and namespace information to a new XmlInputStream (page 270). If this information cannot be resolved by this resolver, this method should return NULL. The `XmlInputStream` object will be deleted by the caller.

Parameters

txn

If a transaction is in force, a pointer to the XmlTransaction (page 407) object; otherwise, NULL.

mgr

The XmlManager (page 274) object associated with the operation.

schemaLocation

The location where the schema resides.

nameSpace

The namespace used by the schema.

result

The string in which the results of the resolution are to be placed.

Class

XmlResolver (page 371)

See Also

XmlResolver Methods (page 372)

Chapter 23. XmlResults

```
#include <DbXml.hpp>

class DbXml::XmlResults {
public:
 XmlResults();
 XmlResults(const XmlResults &);
 ~XmlResults();
 XmlResults &operator = (const XmlResults &)
 ...
};
```

The `XmlResults` class encapsulates the results of a query or other lookup operation. `XmlResults` is a collection of XmlValue (page 416) objects, which may represent any one of the supported types.

An `XmlResults` object is created by executing a query, calling XmlIndexLookup::execute (page 223) or calling XmlManager::createResults (page 294). There are several ways that a query is performed. One is to call XmlManager::query (page 316) directly. This mechanism is appropriate for one-off queries that will not be repeated.

A second approach is to create an XmlQueryExpression (page 362) using XmlManager::prepare (page 315). You then execute the query expression using XmlQueryExpression::execute (page 364). This approach is appropriate for queries that will be performed more than once as it means that the expense of compiling the query can be amortized across multiple queries.

Note that when you perform a query, you must provide an XmlQueryContext (page 341) object. Using this object, you can indicate whether you want the query to be performed eagerly or lazily. If eager evaluation is specified (the default), then the resultant values are stored within the `XmlResults` object. If lazy evaluation is selected, then the resultant values will be computed as needed. In this case the `XmlResults` object will maintain a reference to the affected containers (XmlContainer (page 16)), query context (XmlQueryContext (page 341)), and expression (XmlQueryExpression (page 362)).

The `XmlResults class provides an iteration interface through the XmlResults::next (page 390) method. XmlResults::next (page 390) method returns false and the null value when no more results are available ( XmlValue (page 416)::isNull` returns true). XmlResults::reset (page 393) method can be called to reset the iterator, and the subsequent call to the XmlResults::next (page 390) method will return the first value of the result set.

The copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body.

An object returned from a query or other container-based operation may contains XmlValue (page 416) objects that refer to persistent data that cannot be safely addressed once a transaction commits. It is possible to construct an entirely transient copy of a result set using

XmlResults::copyResults (page 386). Such a copy can be used as long as the object remains in scope but its values will no longer refer to data in any container. This object is not thread-safe, and can only be safely used by one thread at a time in an application.

XmlResults Methods

XmlResults Methods	Description
XmlResults::add	Adds an XmlValue to the end of the result set.
XmlResults::asEventWriter	Returns an XmlEventWriter for the result set.
XmlResults::concatResults	Concatenates one result set into another creating a transient copy.
XmlResults::copyResults	Creates a transient copy of the result set.
XmlResults::getEvaluationType	Returns the evaluation type of the result set.
XmlResults::hasNext	Is there another value in the results set.
XmlResults::hasPrevious	Is there a previous value in the results set.
XmlResults::next	Retrieve the next element in the results set.
XmlResults::peek	Retrieve current value with no iterator movement.
XmlResults::previous	Retrieve the previous element in the result set.
XmlResults::reset	The iterator is placed at the beginning of the result set.
XmlResults::size	Returns the number of elements in the result set.

XmlResults::add

```
#include <DbXml.hpp>

void XmlResults::add(const XmlValue &value)
```

Adds the specified XmlValue (page 416) to the end of the results set. Note that if the XmlResults (page 380) object was created as the result of a lazy evaluation, this method throws an exception. This method is used primarily for application resolution of collections in queries (see XmlResolver (page 371) and XmlManager::createResults (page 294)).

Errors

The `XmlResults::add` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

LAZY_EVALUATION

The method can only be called on eagerly evaluated result sets, or those created from scratch using XmlManager::createResults (page 294).

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::asEventWriter

```
#include <DbXml.hpp>

XmlEventWriter &XmlResults::asEventWriter()
```

Returns an instance of XmlEventWriter (page 192) that can be used to write events into the `XmlResults` object. Please note that only one active XmlEventWriter (page 192) is allowed for an `XmlResults` object.

The `XmlResults` object must be freshly created by XmlManager::createResults (page 294) and empty or an exception is thrown. Multiple node and atomic value events can be written into the returned object. When the event writing is complete the XmlEventWriter::close (page 194) method must be called. The `XmlResults` object can then be bound to a variable using XmlQueryContext::setVariableValue (page 361) and used in queries.

Errors

The `XmlResults::asEventWriter` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

INVALID_VALUE

The `XmlResults` object is not empty

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::concatResults

```
#include <DbXml.hpp>

void XmlResults::concatResults(XmlResults &from);
```

Concatenates transient copies of all XmlValue (page 416) objects in the from argument into the current result set. Copied values no longer reference database objects and can be safely used outside of transactions. Modifications of the values will not affect containers.

The XmlResults (page 380) object in the from argument may be lazily or eagerly evaluated. If it is eager, it is reset before and after use by calling XmlResults::reset (page 393).

Errors

The `XmlResults::concatResults` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

LAZY_EVALUATION

The method can only be called on eagerly evaluated result sets, or those created from scratch using XmlManager::createResults (page 294).

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::copyResults

```
#include <DbXml.hpp>

XmlResults XmlResults::copyResults();
```

Creates a new XmlResults (page 380) object and adds transient copies of all values in the result set. Copied values no longer reference database objects and can be safely used outside of transactions. Modifications of the values will not affect containers. The returned object is eagerly evaluated.

The XmlResults (page 380) object may be lazily or eagerly evaluated. If it is eager, it is reset before and after use by calling XmlResults::reset (page 393).

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::getEvaluationType

```
#include <DbXml.hpp>

XmlQueryContext::EvaluationType XmlResults::getEvaluationType() const
```

Returns the evaluation type of the XmlResults (page 380) object. This type is either `XmlQueryContext::Eager or XmlQueryContext::Lazy`.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::hasNext

```
#include <DbXml.hpp>

bool XmlResults::hasNext()
```

Returns true if there is another element in the results set.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::hasPrevious

```
#include <DbXml.hpp>

bool XmlResults::hasPrevious()
```

Returns true if there is a previous element in the results set.

Errors

The `XmlResults::hasPrevious` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

LAZY_EVALUATION

The method can only be called on eagerly evaluated result sets.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::next

```
#include <DbXml.hpp>

bool XmlResults::next(XmlValue &value);
bool XmlResults::next(XmlDocument &document);
```

Retrieves the next value in the result set. When no more values remain in the result set, the `XmlResults::next` method returns `false`. In an eager results set the iterator logically points "between" adjacent entries which means that alternating calls to `next()` and `previous()` will return the same value.

Two forms of this method exist: one that places the next value in an XmlValue (page 416) object, and another that places the next value in an XmlDocument (page 135) object.

Parameters

value

The XmlValue (page 416) into which the next value in the result set is to be placed.

document

The XmlDocument (page 135) into which the next value in the result set is to be placed.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::peek

```
#include <DbXml.hpp>

bool XmlResults::peek(XmlValue &value)
bool XmlResults::peek(XmlDocument &document)
```

Returns the current element in the results set without moving the internal iterator. If the provided object is successfully populated, this method returns `true`; otherwise, `false` is returned.

Two forms of this method exist: one that places the next value in an XmlValue (page 416) object, and another that places the next value in an XmlDocument (page 135) object.

Parameters

value

The XmlValue (page 416) into which the current value in the result set is to be placed.

document

The XmlDocument (page 135) into which the current value in the result set is to be placed.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::previous

```
#include <DbXml.hpp>

bool XmlResults::previous(XmlValue &value);
bool XmlResults::previous(XmlDocument &document);
```

Retrieves the previous value in the result set. When the first value in the results set has been reached, the `XmlResults::previous` method returns `false`. For an eager results set, the iterator logically points "between" adjacent entries which means that alternating calls to `previous()` and `next()` will return the same value.

This method can only be used on eagerly evaulated result sets.

Two forms of this method exist: one that places the previous value in an XmlValue (page 416) object, and another that places the previous value in an XmlDocument (page 135) object.

Parameters

value

The XmlValue (page 416) into which the previous value in the result set is to be placed.

document

The XmlDocument (page 135) into which the previous value in the result set is to be placed.

Errors

The `XmlResults::previous` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

LAZY_EVALUATION

This method can only be called on eagerly evaluated result sets.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::reset

```
#include <DbXml.hpp>

void XmlResults::reset()
```

If a query was processed with eager evaluation, a call to the `XmlResults::reset` method resets the result set iterator, so that a subsequent call to XmlResults::next (page 390) method will return the first value in the result set. If the query was processed with lazy evaluation then a call to `XmlResults::reset` method throws an exception.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

XmlResults::size

```
#include <DbXml.hpp>

size_t XmlResults::size();
```

If a query was processed with eager evaluation, a call to the `XmlResults::size method returns the number of values in the result set. If the query was processed with lazy evaluation, a call to XmlResults::size` throws an exception.

Errors

The `XmlResults::size` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

LAZY_EVALUATION

The method can only be called on eagerly evaluated result sets.

Class

XmlResults (page 380)

See Also

XmlResults Methods (page 382)

Chapter 24. XmlStackFrame

```
#include <DbXml.hpp>

class DbXml::XmlStackFrame {
public:
    const char *getQueryFile() const;
    int getQueryLine() const;
    int getQueryColumn() const;
    XmlResults query(const std::string &queryString) const;
    std::string getQueryPlan() const;
    const XmlStackFrame *getPreviousStackFrame() const;
};
```

The `XmlStackFrame` class provides stack trace information about an executing query to the methods of XmlDebugListener (page 128). `XmlStackFrame` objects describe a stack frame in the stack trace and includes a pointer to the previous stack frame.

XmlStackFrame Methods

XmlStackFrame Methods	Description
XmlStackFrame::getQueryColumn()	Return the column number of the position in the query
XmlStackFrame::getQueryFile()	Return the URI of the query
XmlStackFrame::getQueryLine()	Return the line number of the position the query
XmlStackFrame::getQueryPlan()	Return the query plan for the sub-expression
XmlStackFrame::getPreviousStackFrame()	Return a pointer to the previous stack frame
XmlStackFrame::query()	Return the query plan for the sub-expression

XmlStackFrame::getQueryColumn()

```
#include <DbXml.hpp>

int XmlStackFrame::getQueryColumn() const
```

Returns the column number of the position in the query represented by this stack frame.

Class

XmlStackFrame (page 395)

See Also

XmlStackFrame Methods (page 396)

XmlStackFrame::getQueryFile()

```
#include <DbXml.hpp>

const char * XmlStackFrame::getQueryFile() const
```

Returns the URI of the query represented by this stack frame.

Class

XmlStackFrame (page 395)

See Also

XmlStackFrame Methods (page 396)

XmlStackFrame::getQueryLine()

```
#include <DbXml.hpp>

int XmlStackFrame::getQueryLine() const
```

Returns the line number of the position in the query represented by this stack frame.

Class

XmlStackFrame (page 395)

See Also

XmlStackFrame Methods (page 396)

XmlStackFrame::getQueryPlan()

```
#include <DbXml.hpp>

std::string XmlStackFrame::getQueryPlan() const
```

Returns the query plan of the sub-expression represented by this stack frame.

Class

XmlStackFrame (page 395)

See Also

XmlStackFrame Methods (page 396)

XmlStackFrame::getPreviousStackFrame()

```
#include <DbXml.hpp>

const XmlStackFrame * XmlStackFrame::getPreviousStackFrame() const
```

Returns a pointer to the previous stack frame or 0 if this is the first stack frame.

Class

XmlStackFrame (page 395)

See Also

XmlStackFrame Methods (page 396)

XmlStackFrame::query()

```
#include <DbXml.hpp>

XmlResults XmlStackFrame::query(const std::string &queryString) const
```

Prepares and executes the query given in the dynamic context of the stack frame. This can be used to examine the value of the context item (".") or variables ("$var") for a given stack frame as well as other parts of the dynamic context. Users may find that the context item and variables present in their original query do not exist during query execution due to optimization performed by BDB XML.

Parameters

queryString

The XQuery query to be executed within the context of this stack frame.

Class

XmlStackFrame (page 395)

See Also

XmlStackFrame Methods (page 396)

Chapter 25. XmlStatistics

```
#include <DbXml.hpp>

XmlStatistics::XmlStatistics()
XmlStatistics::XmlStatistics(const XmlStatistics&)
XmlStatistics &operator = (const XmlStatistics &)
virtual XmlStatistics::~XmlStatistics()
```

The `XmlStatistics` class encapsulates statistical information about the number of keys in existence for a given index. Statistics are available for the total number of keys currently in use by the index, as well as the total number of unique keys in use by the index. Use XmlStatistics::getNumberOfIndexedKeys (page 405) and XmlStatistics::getNumberOfUniqueKeys (page 406), respectively, to retrieve these values.

Be aware that the number the number of keys maintained for an index is a function of the number and size of documents stored in the container, and of the type of index being examined.

`XmlStatistics` objects are instantiated by XmlContainer::lookupStatistics (page 54). This object is free threaded, and can be safely shared among threads in an application.

XmlStatistics Methods

XmlStatistics Methods	Description
XmlStatistics::getNumberOfIndexedKeys	The total number of index keys.
XmlStatistics::getNumberOfUniqueKeys	The number of unique index keys.

XmlStatistics::getNumberOfIndexedKeys

```
#include <DbXml.hpp>

double XmlStatistics::getNumberOfIndexedKeys() const
```

Returns the total number of keys contained by the index for which statistics are being reported.

Class

XmlStatistics (page 403)

See Also

XmlStatistics Methods (page 404)

XmlStatistics::getNumberOfUniqueKeys

```
#include <DbXml.hpp>

double XmlStatistics::getNumberOfUniqueKeys() const
```

Returns the number of unique keys contained in the index for which statistics are being reported. There are likely to be many more keys than unique keys in the index because a given key can appear multiple times, once for each document feature on each document that it is referencing.

Class

XmlStatistics (page 403)

See Also

XmlStatistics Methods (page 404)

Chapter 26. XmlTransaction

```
#include <DbXml.hpp>

       XmlTransaction()
       XmlTransaction(const XmlTransaction &)
       XmlTransaction &operator = (const XmlTransaction &)
       ~XmlTransaction()
```

The `XmlTransaction` class is the handle for a transaction. It encapsulates a `DB_TXN` handle. Methods of the `XmlTransaction` class are used to abort and commit the transaction. `XmlTransaction` handles are provided to XmlContainer (page 16), XmlManager (page 274), and other objects that query and modify documents and containers in order to transactionally protect those database operations.

`XmlTransaction` objects are instantiated using XmlManager::createTransaction (page 296).

`XmlTransaction` handles are not free-threaded; transactions handles may be used by multiple threads, but only serially, that is, the application must serialize access to the `XmlTransaction` handle. Once the XmlTransaction::abort (page 409) or XmlTransaction::commit (page 410) methods are called, the handle may not be accessed again, regardless of the method's return. In addition, parent transactions may not issue any operations while they have active child transactions (child transactions that have not yet been committed or aborted) except for XmlTransaction::abort (page 409) and XmlTransaction::commit (page 410).

If the object is used after a commit or abort, an exception is thrown with the exception code, TRANSACTION_ERROR.

XmlTransaction Methods

XmlTransaction Methods	Description
XmlTransaction::abort	Abort a transaction.
XmlTransaction::commit	Commit a transaction.
XmlTransaction::createChild	Create a child transaction.
XmlTransaction::getDB_TXN	Get the underlying DB_TXN object.

XmlTransaction::abort

```
#include <DbXml.hpp>

void XmlTransaction::abort()
```

The `XmlTransaction::abort` method causes an abnormal termination of the transaction. All write operations previously performed within the scope of the transaction are undone. Before this method returns, any locks held by the transaction will have been released.

In the case of nested transactions, aborting a parent transaction causes all children (unresolved or not) of the parent transaction to be aborted.

After `XmlTransaction::abort` method has been called, regardless of its return, the XmlTransaction (page 407) method handle may not be accessed again.

Errors

The `XmlTransaction::abort` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

TRANSACTION_ERROR

Cannot use `XmlTransaction` once committed or aborted

Class

XmlTransaction (page 407)

See Also

XmlTransaction Methods (page 408)

XmlTransaction::commit

```
#include <DbXml.hpp>

void XmlTransaction::commit(u_int32_t flags = 0)
```

The `XmlTransaction::commit` method ends the transaction. Container and document modifications made within the scope of the transaction are by default written to stable storage.

After `XmlTransaction::commit` has been called, regardless of its return, the XmlTransaction (page 407) handle may not be accessed again. If `XmlTransaction::commit` encounters an error, the transaction and all child transactions of the transaction are aborted.

The `XmlTransaction::commit` method throws an exception that encapsulates a non-zero error value on failure.

Parameters

flags

This parameter must be set to 0 or one of the following values:

- DB_TXN_NOSYNC

  Do not synchronously flush the log. This means the transaction will exhibit the ACI (atomicity, consistency, and isolation) properties, but not D (durability); that is, database integrity will be maintained, but it is possible that this transaction may be undone during recovery.

  This behavior may also be set for a Berkeley DB environment using the `DbEnv::set_flags()` method or for a single transaction using the XmlManager::createTransaction (page 296) method. Any value specified to this method overrides both of those settings.

- DB_TXN_SYNC

  Synchronously flush the log. This means the transaction will exhibit all of the ACID (atomicity, consistency, isolation, and durability) properties.

  This behavior is the default for Berkeley DB environments unless the `DB_TXN_NOSYNC` flag was specified to the `DbEnv::set_flags()` method. This behavior may also be set for a single transaction using the XmlManager::createTransaction (page 296) method. Any value specified to this method overrides both of those settings.

Errors

The `XmlTransaction::commit` method may fail and throw XmlException (page 206), encapsulating one of the following non-zero errors:

TRANSACTION_ERROR

Cannot use `XmlTransaction` once committed or aborted.

DB_REP_LEASE_EXPIRED

You are using a replicated enviroment, the master lease is expired, and DBXML_IGNORE_LEASE is not enabled.

DB_RUNRECOVERY

You are using a replicated enviroment, the master lease is expired, and DBXML_IGNORE_LEASE is not enabled.

Class

XmlTransaction (page 407)

See Also

XmlTransaction Methods (page 408)

XmlTransaction::createChild

```
#include <DbXml.hpp>

XmlTransaction XmlTransaction::createChild(u_int32_t flags = 0)
```

The `XmlTransaction::createChild` method creates a child transaction of this transaction. While this child transaction is active (has been neither committed nor aborted), the parent transaction may not issue any operations except for XmlTransaction::commit (page 410) or XmlTransaction::abort (page 409).

Parameters

flags

This parameter must be set to 0 or by bitwise inclusively **OR**'ing together one or more of the following values:

- `DB_READ_COMMITTED`

  This operation will have degree 2 isolation. This provides for cursor stability but not repeatable reads. Data items which have been previously read by this transaction may be deleted or modified by other transactions before this transaction completes.

- `DB_READ_UNCOMMITTED`

  This operation will support degree 1 isolation; that is, read operations may return data that has been modified by other transactions but which has not yet been committed. Silently ignored if the `DB_READ_UNCOMMITTED` flag was not specified when the underlying container was opened.

- `DB_TXN_NOSYNC`

  Do not synchronously flush the log when this transaction commits or prepares. This means the transaction will exhibit the ACI (atomic, consistent, and isolated) properties, but not D (durable); that is, database integrity will be maintained but it is possible that this transaction may be undone during recovery.

  This behavior may be set for a Berkeley DB environment using the `DbEnv::set_flags()` method. Any value specified to this method overrides that setting.

- `DBXML_IGNORE_LEASE`

  This flag is relevant only when using a replicated environment.

  Perform transactional operations irrespective of the state of master leases. The transactional operations will perform under all conditions: if master leases are not configured, if the request is made to a client, if the request is made to a master with a valid lease, or if the request is made to a master without a valid lease.

  Refer to Master Leases in the *Berkeley DB Programmer's Reference Guide* for more information.

- DB_TXN_NOWAIT

  If a lock is unavailable for any Berkeley DB operation performed in the context of this transaction, cause the operation to return DB_LOCK_DEADLOCK or throw an XmlException (page 206) with DB error code DB_LOCK_DEADLOCK immediately instead of blocking on the lock.

- DB_TXN_SNAPSHOT

  This transaction will execute with snapshot isolation. For containers with the DB_MULTIVERSION flag set, data values will be read as they are when the transaction begins, without taking read locks. Silently ignored for operations on databases with DB_MULTIVERSION not set on the underlying container (read locks are acquired).

  The DB_LOCK_DEADLOCK error is returned from update operations if a snapshot transaction attempts to update data which was modified after the snapshot transaction read it.

- DB_TXN_SYNC

  Synchronously flush the log when this transaction commits or prepares. This means the transaction will exhibit all of the ACID (atomic, consistent, isolated, and durable) properties.

  This behavior is the default for Berkeley DB environments unless the DB_TXN_NOSYNC flag was specified to the DbEnv::set_flags() method. Any value specified to this method overrides that setting.

Class

XmlTransaction (page 407)

See Also

XmlTransaction Methods (page 408)

XmlTransaction::getDB_TXN

```
#include <DbXml.hpp>

DB_TXN *XmlTransaction::getDB_TXN()
```

The `XmlTransaction::getDB_TXN` method returns a pointer to the `DB_TXN` handle encapsulated by this XmlTransaction (page 407) object.

Class

XmlTransaction (page 407)

See Also

XmlTransaction Methods (page 408)

Chapter 27. XmlUpdateContext

```
#include <DbXml.hpp>

class DbXml::XmlUpdateContext {
public:
 XmlUpdateContext();
 XmlUpdateContext(const XmlUpdateContext &);
 ~XmlUpdateContext();
 XmlUpdateContext &operator = (const XmlUpdateContext &)
 ...
};
```

The `XmlUpdateContext` class encapsulates the context within which update operations are performed against an XmlContainer (page 16).

A copy constructor and assignment operator are provided for this class. The class is implemented using a handle-body idiom. When a handle is copied both handles maintain a reference to the same body.

`XmlUpdateContext` objects are instantiated using XmlManager::createUpdateContext (page 299). This object is not thread-safe, and can only be safely used by one thread at a time in an application.

Chapter 28. XmlValue

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 XmlValue();
 XmlValue(const std::string &value);
 XmlValue(const char *value);
 XmlValue(double value);
 XmlValue(bool value);
 XmlValue(const XmlDocument &document);
 XmlValue(Type type, const std::string &value);
    XmlValue(const std::string &typeURI,
        const std::string &typeName,
        const std::string &value);
 XmlValue(Type type, const XmlData &data);

 virtual ~XmlValue();
 XmlValue(const XmlValue &);
 XmlValue &operator=(const XmlValue &);
 bool operator==(const XmlValue &v) const
 bool equals(const XmlValue &v) const;
 bool isNull() const;
 ...
};
```

The `XmlValue` class encapsulates the value of a node in an XML document. The type of the value may be any one of the enumerated types in the `XmlValue::Type` enumeration. There are convenience operators for specially handling the types of STRING, DOUBLE, and BOOLEAN, as well as a constructor to make an `XmlValue` from XmlDocument (page 135).

The `XmlValue` class provides several constructors, each of which maps a C++ type or Berkeley DB XML class onto an appropriate `XmlValue` type. The following table lists the constructor parameter mappings.

C++ Type	XmlValueType
No Parameter	NONE
std::string or const char *	STRING
double	DOUBLE
bool	BOOLEAN
XmlDocument (page 135)	NODE
Type and std::string	String is converted to the specified type.
Type and XmlData (page 118)	`XmlData` is converted to the specified type.

The Type may be any of of the following:

XmlValue::NONE
XmlValue::NODE
XmlValue::ANY_SIMPLE_TYPE
XmlValue::ANY_URI
XmlValue::BASE_64_BINARY
XmlValue::BOOLEAN
XmlValue::DATE
XmlValue::DATE_TIME
XmlValue::DATE_TIME_DURATION
XmlValue::DECIMAL
XmlValue::DOUBLE
XmlValue::DURATION
XmlValue::FLOAT
XmlValue::G_DAY
XmlValue::G_MONTH
XmlValue::G_MONTH_DAY
XmlValue::G_YEAR
XmlValue::G_YEAR_MONTH
XmlValue::HEX_BINARY
XmlValue::NOTATION
XmlValue::QNAME
XmlValue::STRING
XmlValue::TIME
XmlValue::YEAR_MONTH_DURATION
XmlValue::UNTYPE_ATOMIC
XmlValue::BINARY

The `XmlValue` class implements a set of methods that test if the `XmlValue` is of a named type. The `XmlValue` class also implements a set of methods that return the `XmlValue` as a value of a specified type. If the `XmlValue` is of type variable and no query context is provided when calling the test or cast methods, or no binding can be found for the variable, an exception is thrown.

In addition to type conversion, the `XmlValue` class also provides DOM-like navigation functions. Using these, you can retrieve features from the underlying document, such as the parent, an attribute, or a child of the current node.

This object is not thread-safe, and can only be safely used by one thread at a time in an application.

The following constructors are available for this class:

-
  ```
  XmlValue(std:string &value)
  ```

  or

  ```
  XmlValue(const char *value)
  ```

  Construct an `XmlValue` object of type `STRING`.

-
  ```
  XmlValue(double value)
  ```

  Construct an `XmlValue` object of type `DOUBLE`.

-
  ```
  XmlValue(bool value)
  ```

  Construct an `XmlValue` object of type `BOOLEAN`.

-
  ```
  XmlValue(Type, std:string &value)
  ```

  or

  ```
  XmlValue(TYPE, const XmlData &value)
  ```

  Construct an `XmlValue` object of the value `Type`, converting the string value to the specified type.

-
  ```
  XmlValue(const std::string &typeURI, const std::string &typeName,
      const std::string &value)
  ```

  Construct an `XmlValue` object of the URI and type name provided, converting the string value to the specified type.

-
  ```
  XmlValue(XmlDocument &document)
  ```

  Construct an `XmlValue` object of type `NODE`.

XmlValue Methods

XmlValue Methods	Description
XmlValue::asBoolean	Return value as a boolean.
XmlValue::asDocument	Return value as XmlDocument.
XmlValue::asEventReader	Return value as XmlEventReader.
XmlValue::asNumber	Return value as a double.
XmlValue::asString	Return value as a string.
XmlValue::equals	Compare two XmlValue objects.
XmlValue::getAttributes	Get the node's attributes.
XmlValue::getFirstChild	Get the node's first child.
XmlValue::getLastChild	Get the node's last child.
XmlValue::getLocalName	Get the node's local name.
XmlValue::getNamespaceURI	Get the node's namespace URI.
XmlValue::getNextSibling	Get the node's next sibling node.
XmlValue::getNodeHandle	Return a string node handle for the value.
XmlValue::getNodeName	Get the name of the node value.
XmlValue::getNodeType	Get the node type of the node.
XmlValue::getNodeValue	Get the value of the node.
XmlValue::getOwnerElement	Get the node's owner element.
XmlValue::getParentNode	Get the node's parent node.
XmlValue::getPrefix	Get the node's namespace prefix.
XmlValue::getPreviousSibling	Get the node's previous sibling node.
XmlValue::getType	Get the type enumeration of the value.
XmlValue::getTypeName	Get the name of the type.
XmlValue::getTypeURI	Get the URI for the type.
XmlValue::isBinary	Check if value is binary.
XmlValue::isBoolean	Check if value is a boolean.
XmlValue::isNode	Check if value is node.
XmlValue::isNull	Check if the value is initialized.
XmlValue::isNumber	Check if value is a number .
XmlValue::isString	Check if value is a string .
XmlValue::isType	Check type of value.
XmlValue::operator==	Compare two XmlValue objects.

XmlValue::asBoolean

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool asBoolean() const;
 ...
};
```

Returns the value as a BOOLEAN.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::asDocument

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 const XmlDocument &asDocument() const;
 ...
};
```

Returns the value as an XmlDocument (page 135).

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::asEventReader

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlEventReader &asEventReader() const;
 ...
};
```

Returns the value as an XmlEventReader (page 152). Only valid for objects of type `XmlValue::NODE`.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::asNumber

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 double asNumber() const;
 ...
};
```

Returns the value as a DOUBLE.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::asString

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string asString() const;
 ...
};
```

Returns the value as a STRING.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::equals

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 bool equals(const XmlValue &v) const;
 bool operator==(const XmlValue &v) const
 ...
};
```

Determines if two XmlValue (page 416) objects represent the same value. It returns `true` if the two objects represent the same value.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getAttributes

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlResults getAttributes() const
 ...
};
```

Returns XmlResults (page 380) objects that contain all of the attributes appearing on this node.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getFirstChild

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlValue getFirstChild() const
 ...
};
```

Returns current node's parent. If the node has no parent, a null node is returned.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getLastChild

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlValue getLastChild() const
 ...
};
```

Returns current node's last child node. If the node has no children, a null node is returned.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getLocalName

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getLocalName() const
 ...
};
```

Returns the node's local name. For example, if a node has the namespace prefix, "prefix," and its qualified name is prefix:name, then 'name' is the local name.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getNamespaceURI

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getNamespaceURI() const
 ...
};
```

Returns the URI used for the node's namespace.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getNextSibling

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlValue getNextSibling() const
 ...
};
```

Returns the sibling node immediately following this node in the document. If the current node had no siblings following it in the document, a null node is returned.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getNodeHandle

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getNodeHandle() const
 ...
};
```

Returns a string handle representing an XmlValue of type `XmlValue::NODE`, which can be used to construct a new XmlValue (page 416) object representing the same node, at a later time, using `XmlContainer::getNode`. The handle returned encodes its document ID; however, it does not include its container. Matching a node handle to container is the responsibility of the caller.

Node handles are guaranteed to remain stable in the absence of modifications to a document. If a document is modified, a handle may cease to exist, or may belong to a different node.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getNodeName

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getNodeName() const
 ...
};
```

Returns the name of the node contained in this XmlValue (page 416).

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getNodeType

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 short getNodeType() const
 ...
};
```

Returns the short value for this node's type (`XmlValue::NodeType`).

Valid values for XmlValue::NodeType are:

ELEMENT_NODE	PROCESSING_INSTRUCTION_NODE
ATTRIBUTE_NODE	COMMENT_NODE
TEXT_NODE	DOCUMENT_NODE
CDATA_SECTION_NODE	DOCUMENT_TYPE_NODE
ENTITY_REFERENCE_NODE	DOCUMENT_FRAGMENT_NODE
ENTITY_NODE	NOTATION_NODE

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getNodeValue

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getNodeValue() const
 ...
};
```

Returns the node's value.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getOwnerElement

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlValue getOwnerElement() const
 ...
};
```

If the current node is an attribute node, returns the document element node that contains this attribute node.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getParentNode

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlValue getParentNode() const
 ...
};
```

Returns current node's parent. If the node has no parent, a null node is returned.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getPrefix

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getPrefix() const
 ...
};
```

Returns the prefix set for the node's namespace.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getPreviousSibling

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 XmlValue getPreviousSibling() const
 ...
};
```

Returns the sibling node immediately preceding this node in the document. If the current node had no siblings preceding it in the document, a null node is returned.

If the node type is not `XmlValue::NODE`, `XmlException::INVALID_VALUE` is thrown.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getType

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 Type getType() const;
 ...
};
```

Returns the type of the XmlValue from the enumeration XmlValue::Type. Valid values are:

XmlValue::NONE
XmlValue::NODE
XmlValue::ANY_SIMPLE_TYPE
XmlValue::ANY_URI
XmlValue::BASE_64_BINARY
XmlValue::BOOLEAN
XmlValue::DATE
XmlValue::DATE_TIME
XmlValue::DATE_TIME_DURATION
XmlValue::DECIMAL
XmlValue::DOUBLE
XmlValue::DURATION
XmlValue::FLOAT
XmlValue::G_DAY
XmlValue::G_MONTH
XmlValue::G_MONTH_DAY
XmlValue::G_YEAR
XmlValue::G_YEAR_MONTH
XmlValue::HEX_BINARY
XmlValue::NOTATION
XmlValue::QNAME
XmlValue::STRING
XmlValue::TIME
XmlValue::YEAR_MONTH_DURATION
XmlValue::UNTYPE_ATOMIC
XmlValue::BINARY

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getTypeName

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getTypeName() const;

 ...
};
```

Returns the string name of the type of the XmlValue.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::getTypeURI

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 std::string getTypeURI() const;
 ...
};
```

Returns the URI associated with the type of the XmlValue.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isBinary

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool isBinary() const;
 ...
};
```

Returns `true` if the XmlValue is one of type BINARY.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isBoolean

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool isBoolean() const;
 ...
};
```

Returns `true` if the XmlValue is one of type BOOLEAN.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isNode

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool isNode() const;
 ...
};
```

Returns `true` if the XmlValue is one of type NODE.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isNull

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 bool isNull() const;
 ...
};
```

Returns `true` if the XmlValue has no value (type NONE).

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isNumber

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool isNumber() const;
 ...
};
```

Returns `true` if the XmlValue is one of the numeric types, such as DOUBLE, FLOAT, etc.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isString

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool isString() const;
 ...
};
```

Returns `true` if the XmlValue is one of type STRING.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::isType

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 ...
 bool isType(XmlValue::Type type) const
 ...
};
```

Indicates whether the XmlValue is of the specified type.

Valid values for `XmlValue::Type` are:

XmlValue::NONE
XmlValue::NODE
XmlValue::ANY_SIMPLE_TYPE
XmlValue::ANY_URI
XmlValue::BASE_64_BINARY
XmlValue::BOOLEAN
XmlValue::DATE
XmlValue::DATE_TIME
XmlValue::DATE_TIME_DURATION
XmlValue::DECIMAL
XmlValue::DOUBLE
XmlValue::DURATION
XmlValue::FLOAT
XmlValue::G_DAY
XmlValue::G_MONTH
XmlValue::G_MONTH_DAY
XmlValue::G_YEAR
XmlValue::G_YEAR_MONTH
XmlValue::HEX_BINARY
XmlValue::NOTATION
XmlValue::QNAME
XmlValue::STRING
XmlValue::TIME
XmlValue::YEAR_MONTH_DURATION
XmlValue::UNTYPE_ATOMIC
XmlValue::BINARY

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

XmlValue::operator==

```
#include <DbXml.hpp>

class DbXml::XmlValue {
public:
 bool equals(const XmlValue &v) const;
 bool operator==(const XmlValue &v) const
 ...
};
```

Determines if two `XmlValue` objects represent the same value. It returns `true` if the two objects represent the same value.

Class

XmlValue (page 416)

See Also

XmlValue Methods (page 419)

Appendix A. Berkeley DB XML Command Line Utilities

The following describes the command line utilities that are available for Berkeley DB XML.

Utilities

Utility	Description
dbxml_dump	Container dump utility
dbxml_load	Container load utility
dbxml_load_container	Load XML files to a container
dbxml	Interactive shell

dbxml_dump

```
dbxml_dump [-NRrV] [-f output] [-h home] [-P password] xml_container
```

The **dbxml_dump** utility reads the XML container in file **xml_container** and writes it to the standard output using a portable flat-text format understood by the dbxml_load (page 455) utility. The **xml_container** argument must be a file produced using the Berkeley DB XML library functions.

The options are as follows:

- **-f**

  Write to the specified file instead of to the standard output.

- **-h**

  Specify a home directory for the database environment; by default, the current working directory is used.

- **-N**

  Do not acquire shared region mutexes while running. Other problems, such as potentially fatal errors in Berkeley DB, will be ignored as well. This option is intended only for debugging errors, and should not be used under any other circumstances.

- **-P**

  Specify an environment password. Although Berkeley DB utilities overwrite password strings as soon as possible, be aware there may be a window of vulnerability on systems where unprivileged users can see command-line arguments or where utilities are not able to overwrite the memory containing the command-line arguments.

- **-R**

  Aggressively salvage data from a possibly corrupt file. The **-R** flag differs from the **-r** option in that it will return all possible data from the file at the risk of also returning already deleted or otherwise nonsensical items. Data dumped in this fashion will almost certainly have to be edited by hand or other means before the data is ready for reload into another database

- **-r**

  Salvage data from a possibly corrupt file. When used on a uncorrupted database, this option should return equivalent data to a normal dump, but most likely in a different order.

- **-V**

  Write the library version number to the standard output, and exit.

The **dbxml_dump** utility may be used with a Berkeley DB environment (as described for the **-h** option, the environment variable **DB_HOME**, or because the utility was run in a directory containing a Berkeley DB environment). In order to avoid environment corruption when using a Berkeley DB environment, **dbxml_dump** should always be given the chance to detach from the environment and exit gracefully. To cause **dbxml_dump** to release all environment resources and exit cleanly, send it an interrupt signal (SIGINT).

Even when using a Berkeley DB database environment, the dbxml_dump utility does not use any kind of database locking if it is invoked with the **-R** or **-r** arguments. If used with one of these arguments, the **dbxml_dump** utility may only be safely run on XML containers that are not being modified by any other process; otherwise, the output may be corrupt.

The **dbxml_dump** utility exits 0 on success, and >0 if an error occurs.

Environment Variables

DB_HOME

If the **-h** option is not specified and the environment variable DB_HOME is set, it is used as the path of the database home, as described in the DB_ENV->open() method.

dbxml_load

```
dbxml_load [-V] [-f file] [-h home] [-P password] xml_container
```

The **dbxml_load** utility reads from the standard input and loads it into the XML container **xml_container**. The XML container **xml_container** is created if it does not already exist.

The input to **dbxml_load** must be in the output format specified by the dbxml_dump (page 453) utility.

The options are as follows:

- **-f**

  Read from the specified **input** file instead of from the standard input.

- **-h**

  Specify a home directory for the database environment.

  If a home directory is specified, the database environment is opened using the DB_INIT_LOCK, DB_INIT_LOG, DB_INIT_MPOOL, DB_INIT_TXN, and DB_USE_ENVIRON flags to the DB_ENV->open() method. (This means that **dbxml_load** can be used to load data into databases while they are in use by other processes.) If the DB_ENV->open() call fails, or if no home directory is specified, the database is still updated, but the environment is ignored; for example, no locking is done.

- **-P**

  Specify an environment password. Although Berkeley DB utilities overwrite password strings as soon as possible, be aware there may be a window of vulnerability on systems where unprivileged users can see command-line arguments or where utilities are not able to overwrite the memory containing the command-line arguments.

- **-V**

  Write the library version number to the standard output, and exit.

The **dbxml_load** utility may be used with a Berkeley DB environment (as described for the **-h** option, the environment variable **DB_HOME**, or because the utility was run in a directory containing a Berkeley DB environment). In order to avoid environment corruption when using a Berkeley DB environment, **dbxml_load** should always be given the chance to detach from the environment and exit gracefully. To cause **dbxml_load** to release all environment resources and exit cleanly, send it an interrupt signal (SIGINT).

The **dbxml_load** utility exits 0 on success, 1 if one or more key/data pairs were not loaded into the database because the key already existed, and >1 if an error occurs.

Environment Variables

DB_HOME

If the **-h** option is not specified and the environment variable DB_HOME is set, it is used as the path of the database home, as described in the DB_ENV->open() method.

dbxml_load_container

```
dbxml_load_container [-c container] [-h home]
    [-s node|wholedoc] [-f file_list]
    [-p file_list_path] [--v] [--V]
    [--P password] file1.xml file2.xml ...
```

The **dbxml_load_container** utility loads XML documents into the specified container. XML files can either be specified as arguments, or they can be specified in a file using the **-f** option.

This utility will attempt to join an environment if one is active. It is recommended, however, that this tool be used offline. If the joined environment is transactional, this utility will also be transactional, with a separate transaction for each document added.

The options are as follows:

- **-c**

  Specify the name of the container into which you want to load the identified documents. If the container does not currently exist, it is created for you. This is a required parameter.

- **-f**

  Specify a file that contains a list of XML files to load into the container.

- **-h**

  Specify a home directory for the database environment; by default, the current working directory is used.

- **-P**

  Specify an environment password. Although Berkeley DB utilities overwrite password strings as soon as possible, be aware there may be a window of vulnerability on systems where unprivileged users can see command-line arguments or where utilities are not able to overwrite the memory containing the command-line arguments.

- **-p**

  Specify a path prefix to prepend to every filename contained in the file list specified by the **-f** option.

- **-s**

  Specify the container type. Valid value are:

  - **node**

    Use node-level storage. XML documents are broken into their individual nodes, and the nodes are stored as individual records in the underlying database. This is the default.

  - **wholedoc**

Use whole-document storage. Entire XML documents are stored as individual records in the underlying database.

Note that if the container already exists, this option is ignored.

- **-V**

  Write the library version number to the standard output, and exit.

- **-v**

  Generate verbose output.

The **dbxml_load_container** utility exits 0 on success, and >0 if an error occurs.

Environment Variables

DB_HOME

If the **-h** option is not specified and the environment variable DB_HOME is set, it is used as the path of the database home, as described in the DB_ENV->open() method.

dbxml

```
dbxml [-ctVvx] [-h home] [-P password] [-s script] [-z size]
```

The **dbxml** utility provides an interactive shell that you can use to manipulate containers, documents and indices, and to perform XQuery queries against those containers.

dbxml uses an optional Berkeley DB home (environment) directory, which defaults to the current directory. An attempt is made to join an existing environment; if that fails, a private environment is created in the specified location. **dbxml** has a concept of a default open container; that is, the container upon which container operations such as adding and deleting indices are performed. The default container is set by use of the **createContainer** and **openContainer** commands. An in-memory container can be created using the command, **createContainer ""**. This is useful for using **dbxml** without file system side effects.

For a list of the commands available in the shell, use the **help** command. For help on a specific command, pass the command's name to the **help** command. For example:

```
dbxmlsh> help createContainer
```

The options are as follows:

- **-c**

  Create a new environment in the directory specified by the `-h` option. This option should only be used for debugging because it does not allow you to specify important environment configuration options.

- **-h**

  Specify a home directory for the database environment; by default, the current working directory is used.

- **-P**

  Specify an environment password. Although Berkeley DB utilities overwrite password strings as soon as possible, be aware there may be a window of vulnerability on systems where unprivileged users can see command-line arguments or where utilities are not able to overwrite the memory containing the command-line arguments.

- **-s**

  Execute the dbxml commands contained in the **script** file upon shell startup. The commands must be specified one to a line in the script file. If any of the commands contained in the script file fail, the shell will not start.

  For example, the following is the contents of a script that creates a container, loads several files into it, performs a query, and then prints the results:

  ```
  createContainer myContainer.dbxml
  putDocument a {<a><b name="doc1">doc1 n1</b><c>doc1 n2</c></a>}
  putDocument a {<a><b name="doc2">doc2 n1</b><c>doc2 n2</c></a>}
  putDocument a {<a><b name="doc3">doc3 n1</b><c>doc3 n2</c></a>}
  ```

```
query collection("myContainer.dbxml")/a/b
print
```

- **-t**

  Run in transaction mode. Transactions can be used, and they are required for writes.

  Note that if you are running this utility against an existing environment, and that environment is transactional, this option will be automatically enabled.

- **-v**

  Provide verbose output. Specify this option a second time to increase the shell's verbosity.

- **-V**

  Write the library version number to the standard output, and exit.

- **-x**

  Run in secure mode. XQuery queries cannot access the local filesystem or perform network access.

- **-z**

  If creating an environment, specify the cache size in MB (default 64) in **size**.

If you are using **dbxml** to manipulate containers that are managed by an existing database environment, you must specify the path to that existing database environment. **dbxml** cannot be used to create environment files that can be shared with other applications. It will either create a private environment, or join an existing, shareable environment created by another application.

Environment Variables

DB_HOME

If the **-h** option is not specified and the environment variable DB_HOME is set, it is used as the path of the database home, as described in the DB_ENV->open() method.