MarkLogic Server

Java Application Developer's Guide

MarkLogic 6
September, 2012

Last Revised: 6.0-3, April, 2013

Copyright © 2013 MarkLogic Corporation. All rights reserved.

Java Application Developer's Guide

1.0 Introduction to the Java API 6
- 1.1 Java API or Java XCC? 6
- 1.2 Getting Started 7
- 1.3 Creating, Working With, And Releasing a Database Client 8
  - 1.3.1 Creating a Database Client 8
  - 1.3.2 Working With a Database Client 9
  - 1.3.3 Releasing a Database Client 9
- 1.4 A Basic "Hello World" Method 9
- 1.5 Document Managers 10
- 1.6 Streaming 11
- 1.7 Handles 11
- 1.8 Thread Safety of the Java API 14

2.0 Document Operations 16
- 2.1 Document Creation 16
  - 2.1.1 Writing an XML Document To The Database 16
  - 2.1.2 Creating a Text Document In the Database 17
- 2.2 Document Deletion 18
- 2.3 Reading Document Content 19
- 2.4 Format-Specific Operations 20
- 2.5 Writing A Binary Document 20
- 2.6 Reading Content From A Binary Document 20
- 2.7 Reading, Modifying, and Writing Metadata 21
  - 2.7.1 Document Metadata 21
  - 2.7.2 Reading Document Metadata 22
  - 2.7.3 Collections Metadata 23
  - 2.7.4 Properties Metadata 23
  - 2.7.5 Quality Metadata 24
  - 2.7.6 Permissions Metadata 24
  - 2.7.7 Manipulating Document Metadata In Your Application 25
  - 2.7.8 Writing Metadata 25
- 2.8 Conversion of Document Encoding 25

3.0 Searching 29
- 3.1 Overview of Search Using the Java API 29
- 3.2 Search Results in Different Formats 30
- 3.3 Search Using String Query Definition 31
- 3.4 Search Documents Using Key-Value Query Definition 32
  - 3.4.1 JSON Key-Value Searches 32
  - 3.4.2 XML Key-Value Searches 32
- 3.5 Search Documents Using Structured Query Definition 33
  - 3.5.1 Ways to Create a Structured Query 33

3.5.2 Basic Steps to Define a Structured Query Definition 34
3.5.3 Creating a Structured Query From Raw XML or JSON 34
3.5.4 Structured Query Examples 35
3.5.4.1 Example: Date Range Structured Query 37
3.5.4.2 Example: Element Index Structured Query 37
3.5.4.3 Example: Property Value Structured Query 37
3.5.4.4 Example: Property Value Structured Query 38
3.5.4.5 Example: Document Structured Query 39
3.5.4.6 Example: Collection Structured Query 39
3.6 Apply Dynamic Query Options to Document Searches 39
3.6.1 Creating a Combined Query 40
3.6.2 Interaction with Persistent Query Options 42
3.6.3 Performance Considerations 43
3.7 Search On Tuples (Tuples Query / Values Query) 43
3.7.1 Values Search 44
3.7.2 Tuples Search 44
3.8 Limiting A Search To Specific Collections And/Or A Directory 45

4.0 Query Options 46
4.1 Using Query Options 46
4.2 Default Query Options 47
4.3 Using QueryOptionsManager To Delete, Write, and Read Options 47
4.4 Using Query Options With Search 48
4.5 Using QueryOptionsBuilder 49
4.6 Creating Persistent Query Options From Raw JSON or XML 50
4.7 Validating Query Options With setQueryOptionValidation() 52
4.8 Query Options Examples 52
4.8.1 Example: Values and Tuples Query Options 52
4.8.2 Example: Field Constraint Query Options 54
4.8.3 Example: builder.configure Collection Constraint Query Options 54
4.8.4 Example: Path Range Index Constraint Query Options 55
4.8.5 Example: Element Attribute Range Constraint Query Options 56
4.8.6 Example: Geospatial Constraint Query Options 57

5.0 Transactions and Optimistic Locking 59
5.1 Multi-Statement Transactions 59
5.1.1 Transactions and the Java API 59
5.1.2 Transaction Class 60
5.1.3 Starting A Transaction 61
5.1.4 Operations Inside A Transaction 61
5.1.5 Rolling Back A Transaction 62
5.1.6 Committing A Transaction 62
5.1.7 Cookbook: Multistatement Transaction 62
5.2 Optimistic Locking 62
5.2.1 Activating Optimistic Locking 63

5.2.2 DocumentDescriptors 64
5.2.3 Using Optimistic Locking 64
5.2.4 Cookbook: Version Control and Optimistic Locking 65

6.0 Logging 66
6.1 Starting Logging 66
6.2 Suspending and Resuming Logging 66
6.3 Stopping Logging 67
6.4 Log Entry Format 67
6.5 Logging To The Server's Error Log 67

7.0 REST Server Configuration 68
7.1 Creating a Server Configuration Manager Object 68
7.2 Reading and Writing Server Configuration Properties 68
7.3 REST Server Properties 69
7.4 Creating New Server-Related Manager Objects 69
7.5 Namespaces 69
7.5.1 Namespaces Manager 70
7.5.2 Getting Server Defined Namespaces 71
7.5.3 Adding And Updating A Namespace Prefix 71
7.5.4 Reading Prefixes 71
7.5.5 Deleting Prefixes 72
7.6 Logging Namespace Operations 72

8.0 Content Transformations 73
8.1 Installing Transforms 73
8.2 Using Transforms 74
8.2.1 Transforming a Document When Reading It 74
8.2.2 Transforming a Document When Writing It 75
8.2.3 Overall Transform Administration 76
8.2.4 Reading Transforms 76
8.2.5 Logging 76
8.3 Writing Transformations 77

9.0 Resource Service Extensions 78
9.1 Extending the Java API 78
9.2 Creating a Resource Extension in XQuery 79
9.3 Installing Resource Extensions 79
9.4 Deleting Resource Extensions 81
9.5 Listing Resource Extensions 81
9.6 Using Resource Extensions 81

10.0 Troubleshooting 84
10.1 Error Detection 84

10.2 General Troubleshooting Techniques 84

11.0 Technical Support 86

Combined Product Notices 87
COPYRIGHT 87
TRADEMARK NOTICE 99

1.0 Introduction to the Java API

The MarkLogic Java Client API is built on top of a MarkLogic REST API. The REST API, in turn, is built using XQuery that is evaluated against an HTTP App Server. With the Java API, programmers can use MarkLogic without having to learn XQuery, and can easily take advantage of the advanced capabilities for persistence and search of unstructured documents that MarkLogic Server provides.

When working with the Java API, you first create a manager for the type of document or operation you want to perform on the database (for instance, a `JSONDocumentManager` to write and read JSON documents or a `QueryManager` to search the database). To write or read the content for a database operation, you use standard Java APIs such as InputStream, DOM, StAX, JAXB, and Transformer as well as Open Source APIs such as JDOM and Jackson.

The Java API provides a handle (a kind of adapter) as a uniform interface for content representation. As a result, you can use APIs as different as `InputStream` and DOM to provide content for one `read()` or `write()` method. In addition, you can extend the Java API so you can use the existing `read()` or `write()` methods with new APIs that provide useful representations for your content.

This chapter covers a number of basic architecture aspects of the Java API, including fundamental structures such as *database clients*, *managers*, and *handles* used in almost every program you will write with it. Before starting to code, you need to understand these structures and the concepts behind them.

This chapter includes the following sections:

- Java API or Java XCC?
- Getting Started
- Creating, Working With, And Releasing a Database Client
- A Basic “Hello World” Method
- Document Managers
- Streaming
- Handles
- Thread Safety of the Java API

1.1 Java API or Java XCC?

The Java API co-exists with the previously developed Java XCC, as they are intended for different use cases.

A Java developer can use the Java API to quickly become productive in their existing Java environment, using the Java interfaces for search, facets, and document management. It is also possible to use its extension mechanism to invoke XQuery, so as both to leverage development teams XQuery expertise and to enable MarkLogic server functionality not implemented by the Java API.

XCC provides a lower-level interface for running remote or ad hoc XQuery. While it provides significant flexibility, it also has a somewhat steeper learning curve for developers who are unfamiliar with XQuery. You may want to think of XCC as being similar to ODBC or JDBC; a low level API for sending query language directly to the server, while the Java Client API is a higher level API for working with database constructs in Java.

In terms of performance, the Java API is very similar to Java XCC for compatible queries. The Java API is a very thin wrapper over a REST API with negligible overhead. Because it is REST-based, minimize network distance for best performance.

For more information about Java XCC, see the *XCC Developer's Guide*.

1.2 Getting Started

To use the Java API, you must first do the following:

- Create a REST API instance of MarkLogic, which includes an app server which understands MarkLogic REST commands, for your database. For details, see Creating a REST API Instance in the *Information Studio Developer's Guide*, or Create a REST API Instance in the *REST Application Developer's Guide*.

  Note: Each application must make use of a separate modules database and REST API instance.

- From the Admin Interface, create users with the appropriate roles. Any user who reads data will need at least the `rest-reader` role and any user that writes data will need at least the `rest-writer` role. Note that MarkLogic Server Administration is *not* exposed in Java, so operations such as creating indices, creating users, creating databases, etc. must be done via the Admin Interface or other MarkLogic Server administration tool. While there is a server configuration component to the Java API, it is restricted to configuration operations on the REST instance. Any user who will use REST Server configuration operations must be `rest-admin`. For information on configuring the REST Server configuration, see "REST Server Configuration" on page 68. For details on the security requirements of the REST api, see Security Requirements in the *REST Application Developer's Guide*.
- Go to developer.marklogic.com/products/java-api to download the necessary jar and zip files to set up the Java Client API.

The Java Client API distribution includes several examples in the `examples/` directory. The examples include the following packages:

- `com.marklogic.client.example.cookbook`: A collection of small examples of using the core features of the API, such as document operations and search. Most of the example code in this guide is drawn from the Cookbook examples.
- `com.marklogic.client.example.handle`: IO handle subclasses and examples for creating new document handle types on top of open source document models, such as JDOM, DOM4J, and Jackson. You should be able to re-use these extensions as-is in your own applications.
- `com.marklogic.client.example.batch`: A collection of classes and examples for manipulating documents in batches.

1.3 Creating, Working With, And Releasing a Database Client

In order to access a database with the Java API, you must create a client for it, specifically an instance of the `DatabaseClient` class. The `DatabaseClient` instance represents a database connection sharable across threads. This connection is stateless, except that authentication is done the first time a client interacts with the database via a Document Manager, Query Manager, or other manager. This section describes how to create a client and includes the following parts:

- Creating a Database Client
- Working With a Database Client
- Releasing a Database Client

1.3.1 Creating a Database Client

This section includes instructions for creating a database client, which eventually connects to the database. The client's creation configures it for connection, but does not actually connect it to the database.

Before the connection happens, you must also create a manager object, and use the manager object for an operation that needs to connect to the database, such as a read, write, or delete operation. This causes the client to make the actual connection to the database, which lasts until either it is explicitly released or it times out. If it times out, it auto-connects back to the database the next time a manager object in the application does a read, write, or delete operation.

To create a database client, use the `com.marklogic.client.DatabaseClientFactory.newClient()` method. For example:

```
DatabaseClient client = DatabaseClientFactory.newClient("myhost",
                8006, "myuser", "mypassword", Authentication.DIGEST);
```

The `host` and `port` values are those of the database. The `user` and `password` values are the user's credentials for accessing the database. `type` takes one of two values; `Authentication.BASIC` or `Authentication.DIGEST`.

`BASIC` authentication sends the password in obfuscated, but not encrypted, mode. `DIGEST` authentication encrypts passwords sent over the network.

For more information about user authentication, see Authenticating Users in the *Understanding and Using Security Guide*.

If your server is using SSL (*Secure Socket Layer*) for authentication, you *must* provide an `SSLContext` object final argument to `connect()`. SSL provides greater security than `DIGEST`. `SSLContext` instances represent a secure socket protocol implementation which acts as a factory for secure socket factories. For information about creating and working with `SSLContext` objects, see Accessing SSL-Enabled XDBC App Servers in the *Understanding and Using Security Guide*.

For even more security, you can also include a `DatabaseClientFactory.SSLHostnameVerifier` object to check if a hostname is acceptable.

1.3.2 Working With a Database Client

In addition to representing the connection to the database, `DatabaseClient` objects also have many factory methods for creating managers that use the connection. In particular, its methods create *document managers*, *server configuration managers*, *query managers*, *loggers*, and *transactions*. These are covered in elsewhere in this guide.

1.3.3 Releasing a Database Client

When you finish and want to release connection resources, use the `DatabaseClient` object's `release()` method.

```
client.release()
```

IMPORTANT: When you are done with a database client, be sure to release it.

1.4 A Basic "Hello World" Method

The following code is a basic method that creates a new document in the database.

```
public static void run(String host, int port, String user, String
                       password, Authentication authType) {

// Create the database client
DatabaseClient client = DatabaseClientFactory.newClient(host, port,
                                          user, password, authType);

// Make a document manager to work with text files.
TextDocumentManager docMgr = client.newTextDocumentManager();

// Define a URI value for a document.
String docId = "/example/text.txt";

// Create a handle to hold string content.
StringHandle handle = new StringHandle();
```

```
// Give the handle some content
handle.set("A simple text document");

// Write the document to the database with URI from docId
// and content from handle
docMgr.write(docId, handle);

// release the client
client.release();
}
```

The above code is a slightly modified version of the `run` method from the `com.marklogic.client.example.cookbook.ClientCreator` cookbook example. It, along with a number of other basic example applications for the Java API, is located in `example/com/marklogic/client/example/cookbook` directory found in the zip file containing the Java API.

1.5 Document Managers

Different document formats are handled by different *document manager* objects, which serve as an interface between documents and the database connection. The package `com.marklogic.client.document` includes document managers for binary, XML, JSON, and text. If you don't know the document format, or need to work with documents of multiple formats, use a generic document manager. `DatabaseClient` instances have factory methods to create a new `com.marklogic.client.document.DocumentManager` of any subtype.

```
BinaryDocumentManager binDocMgr = client.newBinaryDocumentManager();
XMLDocumentManager XMLdocMgr = client.newXMLDocumentManager();
JSONDocumentManager JSONDocMgr = client.newJSONDocumentManager();
TextDocumentManager TextDocMgr = client.newTextDocumentManager();
GenericDocumentManager genericDocMgr =
client.newGenericDocumentManager();
```

Your application only needs to create one document manager for any given type of document, no matter how many of that type of document it works with. So, even if you expect to work with, say, 1,000,000,000 JSON documents, you only need to create one `JSONDocumentManager` object.

Document managers are thread safe once initially configured; no matter how many threads you have, you only need one document manager per document type.

If you make a mistake and try to use the wrong type of document with a document manager, the result depends on the combination of types. For example, a `BinaryDocumentManager` will try to interpret the document content as binary. `JSONDocumentManager` and `XMLDocumentManager` are the most particular, since if a document is not in their format, it will not parse. Most of the time, you will get an exception error, with `FailedRequestException` the default if the manager cannot determine the document type.

1.6 Streaming

To stream, you supply an `InputStream` or `Reader` for the data source, not only when reading from the database but also when writing to the database. This approach allows for efficient write operations that do not buffer the data in memory. You can also use an `OutputWriter` to generate data as the API is writing the data to the database.

When reading from the database using a stream, be sure to close the stream explicitly if you do not read all of the data. Otherwise, the resources that support reading continue to exist.

1.7 Handles

To access document content and metadata, create a *handle object* that supports the appropriate representation. For example, `com.marklogic.client.io.DOMHandle` is a handle for DOM data. There are many different types of handles, all defined in the `com.marklogic.client.io` package, as content handles are key to working with the Java API. They make use of the Adapter design pattern so the API can read and write a diverse and extensible set of content representations.

Some handles can support both read and write operations.

> **Note:** Handles are *not* thread safe. Whenever you create a new thread, you will have to also create new handle objects to use while in that thread.

Some handles, unlike document managers, can be used with a variety of document formats. For example, an input stream can provide content in any format so `InputStreamHandle` can be used for any document format. Where the manager does not specify the format, you can call `setFormat()` on the handle. For example, you can call the following to write a JSON document with `GenericDocumentManager` or read search results as JSON with `QueryManager`.:

```
InputStreamHandle.setFormat(Format.JSON);
```

The following table lists as columns each of the four content types; XML, Text, JSON, and Binary. Each row is a different handle name. The cells tell whether the associated handle can be used for reading (R) or writing (W) the associated content type. If no letter is present, the handle cannot do anything with that format. If you use a handle on a format it does not accept, it throws an exception.

Handle Name	XML	Text	JSON	Binary
`BytesHandle`	RW	RW	RW	RW
`DOMHandle`	RW			
`FileHandle`	RW	RW	RW	RW
`InputSourceHandle`	RW			

Handle Name	XML	Text	JSON	Binary
`InputStreamHandle`	RW	RW	RW	RW
`JAXBHandle`	RW			
`OutputStreamHandle`	W	W	W	W
`ReaderHandle`	RW	RW	RW	
`SourceHandle`	RW			
`StringHandle`	RW	RW	RW	
`XMLEventReaderHandle`	RW			
`XMLStreamReaderHandle`	RW			

For example, consider a binary JPEG file. To write it to the database, you can create any of `BytesHandle, FileHandle, InputStreamHandle, OutputStreamHandle.`

Or, to read an XML document from the server into a DOM model document object in Java memory, do the following:

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
DOMHandle handle = new DOMHandle();
docMgr.read(docId, handle); //docId is the document's URI
org.w3c.dom.Document document = handle.get();
```

To write content to a database document, set a handle to contain the content, then pass that handle to a document manager's `write()` method. For example:

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
DOMHandle handle = new DOMHandle();
handle.set(document);
docMgr.write(docId, handle);
```

In addition to the handles above, which you use for reading to and writing from the database, other handles do things like contain search results. The following table lists all handles and their intended usage. These handles are all in the package `com.marklogic.client.document`.

.

Handle Name	Primary Use
`BytesHandle`	Represents document content as a byte array. Use for bytes buffered in memory. Use for JSON, text, or XML content only if UTF-8 encoded. Otherwise, use a `StringHandle` and specify the correct character encoding for the bytes when created.

Handle Name	Primary Use
`DocumentMetadataHandle`	Represents a database document's metadata as a POJO (Plain Old Java Object).
`DOMHandle`	Represents XML content as a DOM document for reading or writing. Use when working with parsed XML that is all in memory.
`FileHandle`	Represents document content as a file. Use for JSON, text, or XML content only if UTF-8 encoded, otherwise use `ReaderHandle` and specify encoding when created.
`InputSourceHandle`	Represents XML content as an input source for reading or writing. Use for streaming XML. When reading, the XML may be processed by a SAX content handler. When finished, close to release resources.
`InputStreamHandle`	Represents a resource as an input stream for reading or writing. When finished with the input stream, close it to release the resources. When writing JSON, text, or XML content, use an `InputStream` only if it is encoded in UTF-8. If the characters have a different encoding, use a `ReaderHandle` and specify the correct character encoding for the stream when creating the Reader.
`JAXBHandle`	Roundtrips a POJO (Plain Old Java Object) to and from a database document. The POJO class must have JAXB annotations or must be generated by xjc from an XML Schema. The JAXB Handle must be initialized with a JAXB Context with which the root POJO classes have been registered.
`OutputStreamHandle`	Generates output during writing. Define a class that implements the `OutputStreamSender` interface. When the client is ready to write to the server, it calls the `write()` method of the class with an `OutputStream`. That is, the `write()` method generates the output to be written to the server. The `OutputStreamSender` implementer is typically an anonymous class. Initialize the `OutputStreamHandle` with the `OutputStreamSender` implementer before passing the `OutputStreamHandle` to a document manager's `write()` method.
`QueryOptionsHandle`	Used to configure query options as a POJO. Use it if you want to use Java to configure and manage MarkLogic query options for search, value lookups, and facets.

Handle Name	Primary Use
`ReaderHandle`	Represents JSON, text, or XML content as a character reader for reading to or writing from the database. When finished with it, close it to release the resources.
`SearchHandle`	Represents a set of search results returned by the server as a POJO. Their search results context depends on the search's query options and on this handle configuration. Snippets, in particular, are returned in various ways. In the default case, snippets are returned as Java objects. For custom or raw snippets, DOM documents are returned. You can set the `forceDOM` flag to always return DOM documents, even in the default case.
`SourceHandle`	Represents XML content as a transform source for reading or transforms a source into a result for writing. Use when you want to transform XML documents using Java JAXP on the client.
`StringHandle`	Represents JSON, text, or XML content as a string for reading or writing. Use for character data buffered in memory
`TuplesHandle`	Represents a set of tuples returned by a query on the server as a POJO.
`ValuesHandle`	Represents a set of values recieved from the server.
`XMLEventReaderHandle`	Represents XML content as an XML event reader for reading as a series of StAX events. Use for streaming XML. When finished with the event reader, close it to release the response.
`XMLStreamReaderHandle`	Represents XML content as an XML stream reader for reading as a StAX pull stream. Use for streaming XML. When finished with the stream reader, close it to release the response.

1.8 Thread Safety of the Java API

You should be aware of the following API characteristics with respect to thread safety:

- `DatabaseClient` is thread safe after initialization.
- The various manager classes are thread safe after initial configuration. Examples: `DocumentManager`, `QueryManager`, `ResourceManager`.
- Handles are not thread safe. Examples: `StringHandle`, `FileHandle`, `SearchHandle`.
- Builders are not thread safe. Examples: `DocumentPatchBuilder`, `StructuredQueryBuilder`.

For example, you can create a `DocumentManager` for manipulating XML documents and share it across multiple threads. Similarly, you can create a `QueryManager`, set the page length, and then share it between multiple threads.

Handles can be used across multiple requests within the same thread, but cannot be used across threads, so whenever you create a new thread, you must create new Handle objects to use in that thread.

2.0 Document Operations

This chapter describes how to create, delete, write to, and read document content and metadata using the Java API. It describes core methods such as `read()`, `write()`, `delete()`, and so on. These methods have many different signatures for use with more advanced operations such as transactions, transforms, and others. Specific method signatures for calling `read()`, etc. in these advanced contexts are discussed when a relevant operation is covered.

When working with documents, it is important to keep in mind the difference between a document on your client and a document in the database. In particular, any changes you make to a document's content and metadata on the client do not persist between sessions. Only if you write the document out to the database do your changes persist.

This chapter includes the following sections:

- Document Creation
- Document Deletion
- Reading Document Content
- Writing A Binary Document
- Reading Content From A Binary Document
- Reading, Modifying, and Writing Metadata
- Conversion of Document Encoding

2.1 Document Creation

Document creation is not done via a document creation method. When you first write content via a Manager object to a document in the database as identified by its URI, MarkLogic Server creates a document in the database with that URI and content.

> **Note:** To call `write()`, an application must authenticate as a user with at least one of the `rest-writer` or `rest-admin` roles (or as a user with the `admin` role).

This section describes the following about document creation operations:

- Writing an XML Document To The Database
- Creating a Text Document In the Database

2.1.1 Writing an XML Document To The Database

Note that no changes you make to a document or its metadata persist until you write the document out to the database. Within your application, you are only manipulating it within system memory, and those changes will vanish when the application ends. The database content is constant until and unless a write or delete operation changes it.

The basic steps needed to write a document are:

1. If you have not already done so, connect to the database, storing the connection in a `com.marklogic.client.DatabaseClient` object.

```
DatabaseClient client = DatabaseClientFactory.newClient(
  host, port, user, password, authType);
```

2. If you have not already done so, use the `DatabaseClient` object to create a `com.marklogic.client.document.DocumentManager` object of the appropriate subclass for the document content you want to access (XML, text, JSON, binary, generic). In this example code, an `XMLDocumentManager`.

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
```

3. Get the document's content. For example, by using an `InputStream`. b

```
FileInputStream docStream = new FileInputStream(
                                "data"+File.separator+filename);
```

4. Create a handle associated with the input stream to receive the document's content. How you get content determines which handle you use. Use the handle's `set()` method to associate it with the desired stream.

```
InputStreamHandle handle = new InputStreamHandle(docStream);
```

5. Write the document's content by calling a `write()` method on the `DocumentManager`, with arguments of the document's URI and the handle.

```
docMgr.write(docId, handle);
```

6. When finished with the database, release the connection resources by calling the `DatabaseClient` object's `release()` method.

```
client.release();
```

2.1.2 Creating a Text Document In the Database

This procedure outlines a very basic creation operation for a simple text document is as follows:

1. Create a `com.marklogic.client.DatabaseClient` for the database.

```
DatabaseClient client = DatabaseClientFactory.newClient(
    host, port, user, password, authType);
```

2. Create a `com.marklogic.client.document.DocumentManager` object of the appropriate format for your document; text, binary, JSON, XML, or generic if you are not sure.

```
TextDocumentManager docMgr = client.newTextDocumentManager();
```

3. For convenience's sake, set a variable to your new document's URI. This is not required; the raw string could be used wherever `docId` is used.

```
String docId = "/example/text.txt";
```

4. As discussed previously in "Handles" on page 11, within MarkLogic Java applications you use handle objects to contain a document's content and metadata. Since this is a text document, we will use a `com.marklogic.client.io.StringHandle` to contain the text content. After creation, set the handle's value to the document's initial content.

```
StringHandle handle = new StringHandle();
handle.set("A simple text document");
```

5. Write the document content out to the database. This creates the document in the database if it is not already there (if it is already there, it updates the content to whatever is in the `handle` argument). The identifier for the document is the value of the `docId` argument.

```
docMgr.write(docId, handle);
```

6. When finished with the database, release the connection resources by calling the `DatabaseClient` object's `release()` method.

```
client.release();
```

2.2 Document Deletion

To delete a document, call the `com.marklogic.client.document.DocumentManager` object's `delete()` method with an argument of the URI of the document you want to delete.

> **Note:** To call `delete()`, an application must authenticate as a user with at least one of the `rest-writer` or `rest-admin` roles (or as a user with the `admin` role).

The following example shows how to delete an XML document from the database.

1. Create a `com.marklogic.client.DatabaseClient` for connecting to the database.

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);
```

2. If you have not already done so, use the `DatabaseClient` object to create a `com.marklogic.client.document.DocumentManager` object of the appropriate subclass for the document format (XML, text, JSON, or binary).

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
```

3. Delete the document.

```
String docId = "/example/text.xml";
docMgr.delete(docId);
```

4. When finished with the database, release the connection resources by calling the `DatabaseClient` object's `release()` method.

```
client.release();
```

2.3 Reading Document Content

Reading requires a handle to access document content.

Note that no changes you make to a document or its metadata persist until you write the document out to the database. Within your application, you are only manipulating it on the client, and those changes will vanish when the application ends. The database content is persistent until and unless a write or delete operation changes it.

If you read content with a stream, you must close the stream when done. If you do not close the stream, HTTP clients do not know that you are finished and there are fewer connections available in the connection pool.

The basic steps to read a document from the database are:

1. Create a `com.marklogic.client.DatabaseClient` for connecting to the database.

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);
```

2. If you have not already done so, use the `DatabaseClient` object to create a `com.marklogic.client.document.DocumentManager` object of the appropriate subclass for the document format (XML, text, JSON, or binary).

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
```

3. Create a handle to receive the document's content. For information on handles and the wide variety of handle types, see "Handles" on page 11. This example uses a `com.marklogic.client.io.DOMhandle` object.

```
DOMHandle handle = new DOMHandle();
```

4. Read the document's content by calling a `read()` method on the `DocumentManager`, with arguments of the document's URI and the handle. Here, assume `docId` contains the document's URI.

```
docMgr.read(docId, handle);
```

5. Access the content by calling a `get()` method on the handle. For example, `DomHandle.get` returns a W3C `Document` object. There are many alternatives.

```
Document document = handle.get();
```

6. When finished with the database, release the connection resources by calling the `DatabaseClient` object's `release()` method.

```
client.release();
```

2.4 Format-Specific Operations

Different document formats have different things you can do. The following table highlights some of the format-specific operations you can perform on documents:

Format	Operation
binary	On write, you can extract metadata. On read, you can extract a range of bytes, only reading that range (instead of the whole document).
XML	You can to document repair.
JSON	You can specify language.

2.5 Writing A Binary Document

To insert or update a binary document, use a handle containing your binary content with `com.marklogic.client.document.BinaryDocumentManager`. You can use any handle that implements `BinaryWriteHandle`, such as `BytesHandle` or `FileHandle`.

No metadata extraction is performed by default. You can request metadata extraction and specify how it is saved by calling `BinaryDocumentManager.setMetadataExtraction()`.

The following example reads a JPEG image from a file named `my.png` and inserts it into the database as a binary document with URI `/images/my.png`. During insertion, metadata is extracted from the binary content and saved as document properties.

```
String docId = "/example/my.png";
String mimetype = "image/png";

BinaryDocumentManager docMgr = client.newBinaryDocumentManager();
docMgr.setMetadataExtraction(MetadataExtraction.PROPERTIES);

docMgr.write(
    docId,
    new FileHandle().with(new File("my.png")).withMimetype(mimetype)
  );
```

2.6 Reading Content From A Binary Document

There are several ways to read content from a binary document.

To stream binary content, use InputStream as follows:

```
InputStream byteStream = docMgr.read(docID,
                                     new InputStreamHandle()).get();
```

To buffer the binary content, use `com.marklogic.client.io.BytesHandle` object as follows:

```
byte[] buf = docMgr.read(docID, new BytesHandle()).get();
```

Or you can read only part of the content:

```
BytesHandle handle = new BytesHandle();
buf = docMgr.read(docId, handle, 9, 10).get();
```

2.7 Reading, Modifying, and Writing Metadata

Reading and writing document metadata from and to the database are very similar operations to reading and writing document content. Each requires calling methods on `DocumentManager`. The handle for metadata can be a `com.marklogic.client.document.DocumentMetadataHandle` to modify metadate in a POJO, or it can be raw XML or JSON.

You can perform operations on the metadata associated with documents such as collections, permissions, properties, and quality. This section describes those metadata operations and includes the following parts:

- Document Metadata
- Reading Document Metadata
- Collections Metadata
- Properties Metadata
- Quality Metadata
- Permissions Metadata
- Manipulating Document Metadata In Your Application
- Writing Metadata

2.7.1 Document Metadata

The following are the metadata types in the Java API:

- `COLLECTIONS`: Document collections, a non-hierarchical way of organizing documents in the database. For details, see “Collections Metadata” on page 23
- `PERMISSIONS`: Document permissions. For details, see “Permissions Metadata” on page 24.
- `PROPERTIES`: Document properties. Property-value pairs associated with the document. For details, see “Properties Metadata” on page 23.

- `QUALITY`: Document search quality. Helps determine which documents are of the best quality. For details, see "Quality Metadata" on page 24.

The enum `DocumentManager.Metadata` enumerates the metadata categories (including `ALL`). They are described in detail later in this chapter.

2.7.2 Reading Document Metadata

The basic steps needed to read a document's metadata are:

1. Create a `com.marklogic.client.DatabaseClient` for connecting to the database.

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);
```

2. If you have not already done so, use the `DatabaseClient` object to create a `com.marklogic.client.document.DocumentManager` object of the appropriate subclass for the document format (XML, text, JSON, or binary).

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
```

3. Create a `com.marklogic.client.io.DocumentMetadataHandle` object, which will receive the document's metadata. Alternately, you can create raw XML or JSON.

```
DocumentMetadataHandle metadataHandle = new DocumentMetadataHandle();
```

4. If you also want to get the document's content, create a handle to receive it. Note that you need separate handles for a document's content and metadata.

```
DOMHandle docHandle = new DOMHandle();
```

5. Read the document's metadata by calling a `readMetadata()` method on the `DocumentManager`, with an argument of the metadata handle. Note that you can also call `read()` with an additional argument of a content handle so that it will read the metadata into the metadata handle and the content into the content handle in a single operation. To call `read()`, an application must authenticate as `rest-reader`, `rest-writer`, or `rest-admin`. Below, `docId` is a variable containing a document URI.

```
//read only the metadata into a handle
docMgr.readMetadata(docId, metadataHandle);
//read metadata and content
docMgr.read(docId, metadataHandle, docHandle);
```

6. Access the metadata by calling `get()` methods on the metadata handle. Later sections in this chapter show how to access the other types of metadata.

```
DocumentCollections collections = metadataHandle.getCollections();
Document document = contentHandle.get();
```

7. When finished with the database, release the connection resources by calling the `DatabaseClient` object's `release()` method.

```
client.release();
```

By default, `DocumentManager` reads and writes all categories of metadata. To read or write a subset of the metadata categories, configure `DocumentManager` by calling `setMetadataCatefories()`.

2.7.3 Collections Metadata

Collections are a way to organize documents in a database. A collection defines a set of documents in the database. You can set documents to be in any number of collections either at the time the document is created or by updating a document. Searches against collections are both efficient and convenient. For more details on collections, see Collections in the *Search Developer's Guide*.

The Java API allows you to read and manipulate collections metadata using the `com.marklogic.client.io.DocumentMetadataHandle.DocumentCollections`. Collections are named by specifying a URI. A collection URI serves as an identifier, and it can be any valid URI.

The code in this section assumes a `DocumentManager` object of an appropriate type for the document, `docMgr`, and a string containing a document URI, `docId`, have been created.

To get all collections for a document and put them in an array, do the following:

```
//Get the set of collections the document belongs to and put in array.
DocumentCollections collections = metadataHandle.getCollections();
```

To check if a collection URI exists in a document's set of collections, do the following:

```
collections.contains("/collection_name/collection_name2");
```

To add a document to one or more collections, do the following:

```
collections.addAll("/shakespeare/sonnets", "/shakespeare/plays");
```

To remove a document from a collection, do the following:

```
collections.remove("/shakespeare/sonnets");
```

To remove a document from all its collections, do the following:

```
collections.clear();
```

2.7.4 Properties Metadata

Manipulate properties metadata using the `com.marklogic.client.io.DocumentMetadataHandle.DocumentProperties` class.

The code in this section assumes a `DocumentManager` object, `docMgr`, and a string containing a document's URI, `docId`, have been created.

To get all of a document's properties metadata, do the following:

```
DocumentProperties properties = metadataHandle.getProperties();
```

`DocumentProperties` objects represent a document's properties as a map.

To check if a document's properties contain a specific property name, do the following:

```
exists = properties.containsKey("name");
```

To get a specific property's value do the following:

```
value = metadataHandle.getProperties("name");
```

You can add any new property names and values to a document that you want. To add a new property or change the value of an existing property in a document's metadata do the following:

```
metadataHandle.getProperties().put("name", "value");
```

2.7.5 Quality Metadata

The code in this section assumes a `com.marklogic.client.io.DocumentManager` object, `docMgr`, and a string containing a document's URI, `docId`, have been created.

The quality metadata affects the ranking of documents for use in searches my creating a multiplier for calculating the score for that document, and the default value for quality in the Java API is `0`.

To get a document's search quality metadata value do the following:

```
int quality = metadataHandle.getQuality();
```

To set a document's search quality value do the following:

```
metadataHandle.setQuality(3);
```

2.7.6 Permissions Metadata

Permissions on documents control who can access a document for the capabilities of read, update, insert, and execute. To perform one of these operations on a document, a user must have a role corresponding to the permission for each capability needed. For details on permissions and on the security model in MarkLogic Server, see the *Understanding and Using Security Guide*.

The code in this section assumes a `DocumentManager` object, `docMgr`, and a string containing a document's URI, `docId`, have been created. Manipulate document properties using the class `com.marklogic.client.io.DocumentMetadataHandle.DocumentPermissions`.

MarkLogic Server defines permissions using roles and capabilities.

The allowed values for capabilities are those in the enum `com.marklogic.client.io.DocumentMetadataHandle.Capability`:

- `EXECUTE` - Permission to execute the document.
- `INSERT` - Permission to create but not modify or delete the document.
- `READ` - Permission to read the document but not modify it..
- `UPDATE` - Permission to create, modify, or delete the document, but not to read it.

Roles are assigned to users via the Admin Interface or through other administrative tools, and cannot be assigned via the Java API. You can, however, control permissions on documents as part of their metadata.

To get permissions metadata for a document, do the following:

```
DocumentPermissions permissions = metadataHandle.getPermissions()

metadataHandle.getPermissions().add("app-user",
   Capability.UPDATE, Capability.READ);
```

2.7.7 Manipulating Document Metadata In Your Application

A `DocumentMetadataHandle` represents metadata as a POJO. A `DocumentMetadataHandle` has several methods for manipulating a document's metadata. That may not be how you want to work with the metadata, however. If you would prefer to work with it as XML, then read it with an XML handle. If you would prefer to work with it as JSON, read it with a JSON handle. A `StringHandle` can use either XML or JSON, defaulting to XML.

To specify the format for reading content, use `setFormat()`, as in the following example:

```
StringHandle metadataHandle = new StringHandle();
metadataHandle.setFormat(Format.JSON);
```

2.7.8 Writing Metadata

When you are finished modifying metadata categories, you must write it to the database to persist it. Note that the above operations all only change the document's metadata stored on the client, and do not change the metadata for document in the database. To write the metadata changes to the database, as well as the document content, do the following:

```
InputStreamHandle handle = new InputStreamHandle(docStream);
docMgr.write(docId, metadataHandle, handle);
```

2.8 Conversion of Document Encoding

The Java API handles encoding conversions for you, but you have to:

- know the encoding
- use the appropriate handle

If you specify the encoding and it turns out to be the wrong encoding, then the conversion will likely not turn out as you expect.

MarkLogic Server stores text, XML, and JSON as UTF-8. In Java, characters in memory and input streams are UTF-16. The Java API converts characters to and from UTF-8 automatically.

When writing documents to the server, you need to know if they are already UTF-8 encoded. If a document is not UTF-8, you must specify its encoding or you are likely to end up with data that has incorrect characters due to the incorrect encoding. If you specify a non-UTF-8 encoding, the Java API will automatically convert the encoding to UTF-8 when writing to MarkLogic.

When writing characters to or reading characters from a file, Java defaults to the platform's standard encoding. For example, there is different platform encoding on Linux than Windows.

XML supports multiple encodings as defined by the header (called an XML declaration):

```
<?xml version="1.0" encoding ="utf-8">
```

The XML declaration declares a file's encoding. XML parsing tools, including handles, can determine encoding from this and do the conversion for you.

When writing character data to the database, you need to pick an appropriate handle type, depending on your intent and circumstances.

Depending on your application, you may need to be aware that MarkLogic Server normalizes text to precomposed Unicode characters for efficiency. Unicode abstract characters can either be precomposed (one character) or decomposed (two characters). If you write a decomposed Unicode document to MarkLogic Server and then read it back, you will get back precomposed Unicode. Usually, you do not need to care if characters are precomposed or decomposed. This Unicode issue only affects some characters, and many APIs abstract away the difference. For instance, the Java collator treats the precomposed and decomposed forms of a character as the same character. If your application needs to compensate for this difference, you can use `java.text.Normalizer`; for details, see:

http://docs.oracle.com/javase/6/docs/api/java/text/Normalizer.html

The following table describes possible cases for reading character data with recommended handles to use in each case.

Read Condition	Recommended Handle(s)
If reading binary data:	Use `BytesHandle`, `FileHandle`, or `InputStreamHandle`.
If reading character data from the database:	`BytesHandle`, `FileHandle`, `InputStreamHandle`, and the XML handles are encoded as UTF-8. `StringHandle` and `ReaderHandle` convert to UTF-16.

The following table describes possible cases for writing character data with recommended handles to use in each case.

Write Condition	Recommended Handle(s)
If the data you are writing is a Java string:	Use `StringHandle`; it converts on write from UTF-16 to UTF-8.
If writing binary data:	Use `BytesHandle`, `FileHandle`, or `InputStreamHandle`.
If the data you are writing is encoded as UTF-8 and you do not need to modify the data:	Use `BytesHandle`, `FileHandle`, or `InputStreamHandle`.
If it is XML that declares an encoding other than UTF-8 in the XML declaration and you do not need to modify the data:	Use `InputSourceHandle`, `XMLEventReaderHandle`, or `XMLStreamReaderHandle`; these convert to UTF-8.
If the character data to write is XML that declares the encoding in a prolog and you need to modify the data:	Use `DOMHandle`, `SourceHandle`, or create a handle class on an open source DOM. For examples of the latter, see `JDOMHandle`, `XOMHandle`, or `DOM4JHandle` in the package `com.marklogic.client.example.handle`. All these classes convert to UTF-8.
If the character data to write has a known encoding other than UTF-8 and you don't need to modify the data:	Use `ReaderHandle` and specify the encoding when creating the `Reader` (as usual in Java); these convert to UTF-8.

Write Condition	Recommended Handle(s)

If the character data to write is XML with a known but undeclared encoding and you need to modify the data:	Use `DOMHandle` with a `DocumentBuilder` parsing an `InputSource` with a specified encoding as in:
	
	`DOMHandle handle = new DOMHandle();`
	`handle.set(`
	`handle.getFactory().newDocumentBuilder()`
	`parse(newInputSource(...reader specifying charset ...)));`
	
	or Use `SourceHandle` with a `StreamReader` on a `Reader` with a specified encoding as in:
	
	`SourceHandle handle = new SourceHandle();`
	`handle.set(new StreamSource(... reader specifying charset ...));`

If the character data to write is JSON and you need to modify the data:	Consider using a JSON library such as Jackson or GSON. See `com.marklogic.client.example.handle.JacksonHandle` for an example.
If the character data to write is text other than JSON or XML and you need to modify the data:	Consider using a `StreamTokenizer` with a `Reader`, or `Pattern` with a `String`

3.0 Searching

This chapter describes how to submit searches using the Java API, and includes the following sections:

- Overview of Search Using the Java API
- Search Results in Different Formats
- Search Using String Query Definition
- Search Documents Using Key-Value Query Definition
- Search Documents Using Structured Query Definition
- Apply Dynamic Query Options to Document Searches
- Search On Tuples (Tuples Query / Values Query)
- Limiting A Search To Specific Collections And/Or A Directory

3.1 Overview of Search Using the Java API

The MarkLogic Java API provides two fundamental ways of querying the database:

- Searches on documents, which return search results, snippets, and facets.
- Value or Tuple (co-occurrences) searches, which return data from range indexes and the results of aggregate functions (including user-defined aggregate functions) applied to range indexes.

When you search documents you can express search criteria using one of the following kinds of query:

- String query: Use a Google-style query string to search documents and metadata.
- Key-Value query: Use key-value pair queries for ad hoc queries on XML elements, XML attributes, or JSON keys.
- Structured query: A simple and easy way to construct queries as a Java, XML, or JSON structure, allowing you to manipulate complex queries (such as geospatial polygons) in the Java client.
- Combined query: Combine a string or structured query with dynamic query options.

When you query or aggregate range indexes, you express your search criteria using a values query.

Each search method can also use persistent query options. *Persistent query options* are stored on the REST Server and referenced by name in future queries. Once created and persisted, you can apply query options to multiple searches, or even set to be the default options for all searches. Note that in XQuery, query option configurations are called *options nodes*.

A combined query allows you to bundle a string and/or structured query with dynamic query options to further customize a search on a per search basis. You can also specify persistent query options with a combined query search. The search automatically merges the persistent (or default) query options and the dynamic query options together. For details, see “Apply Dynamic Query Options to Document Searches” on page 39.

Query options can be very simple or very complex. If you accept the defaults, for example, there is no need to specify explicit query options. You can also make them as complex as is needed.

For details on how to create and work with query option configurations, see “Query Options” on page 46. For details on individual query options and their values, see Appendix A: JSON Query Options Reference in the *REST Application Developer’s Guide* and Appendix B: XML Query Options Reference in the *REST Application Developer’s Guide*. For more information on search concepts, see the *Search Developer’s Guide*.

In the examples in this chapter, assume a `DatabaseClient` called `client` has already been defined.

3.2 Search Results in Different Formats

Frequently you will use a `SearchHandle` object to contain your query results. The exact nature of results varies, depending on both the handle’s configuration and what query options and values were used for the search operation.

You can specify snippets to return in various ways. By default, they return as Java objects. But for custom or raw snippets, they are returned as DOM documents by using the `forceDOM` flag.

There are several ways to access different parts of the search result or control search results from a `SearchHandle`.

- The `getMatchResults()` method returns an array of `MatchDocumentSummary` objects of the matched documents, from which you can further extract for each result its match locations, path, metadata, an array of smippets, fitness, confidence measure, and URI. For details, see the `MatchDocumentSummary` entry in Java API JavaDoc.
- `getMetrics()` returns a `SearchMetrics` object containing various timing metrics about the search.
- `getFacetNames()`, `getFacetResult(name)`, `getFacetResults()` return, respectively, a list of returned facet names, the specified named facet result, and an array of facet results for this search.
- `getTotalResults()` returns an estimate of the number of results from the search.
- `setForceDOM(boolean)` sets the force DOM flag, which if `true` causes snippets to always be returned as DOM documents.

See the Java API JavaDoc for SearchHandle for its remaining methods.

The following is a typical programming technique for accessing search results using a search handle:

```
// iterate over MatchDOcumentSummary array locations, getting
// the snippet text for each location (you would then do something
// with the snippet text)
MatchDocumentSummary[] summaries = results.getMatchResults();
for (MatchDocumentSummary summary : summaries ) {
    MatchLocation[] locations = summary.getMatchLocations();
    for (MatchLocation location : locations) {
        location.getAllSnippetText();
      // do something with the snippet text
    }
}
```

3.3 Search Using String Query Definition

The MarkLogic Server Search API lets you do searches on string arguments, including the usual search operators such as AND and OR. For example, you could search on “Batman”, “Batman AND Robin”, “Batman OR Robin”, etc. For details, see Search Grammar in the *Search Developer’s Guide*.

1. As always with searches, start by defining a `QueryManager`. Like documents, the manager deals with interaction between the client and the database.

   ```
   QueryManager queryMgr = client.newQueryManager();
   ```

2. For a string search, define your query in a `StringQueryDefinition` object. After creating a `StringQueryDefinition`, use `setCriteria()` to specify your search string.

   ```
   StringQueryDefinition querydef =
                         queryMgr.newStringDefinition();

   querydef.setCriteria("Batman AND Robin");
   ```

3. Run a search with the `StringQueryDefinition` object as an argument, returning a `SearchHandle` object or an XML or JSON handle to get the search results in either of those formats:

   ```
   SearchHandle results = queryMgr.search(querydef, new SearchHandle());
   DomHandle results = queryMgr.search(querydef, new DomHandle());
   StringHandle results = querymgr.search(querydef,
                     newStringHandle().withFormat(Format.JSON)
   ```

4. Process and/or display the results using the handle.

3.4 Search Documents Using Key-Value Query Definition

A key-value search uses key-value pair queries to search XML and JSON content and metadata using an XML element, an XML attribute, or a JSON key. Using XML elements or attributes may require binding a namespace on the server; for details on namespaces, see “Namespaces” on page 69.

Key-value searches use the `exact` value semantics for the value stored in the database (eqivalent to the `cts:element-value-query` exact option). An exact match uses the query options `"case-sensitive"`, `"diacritic-sensitive"`, `"punctuation-sensitive"`, `"whitespace-sensitive"`, `"unstemmed"`, and `"unwildcarded"`, and by default is performed as an unfiltered search.

This section shows the following ways of doing key-value searches:

- JSON Key-Value Searches
- XML Key-Value Searches

3.4.1 JSON Key-Value Searches

The basic steps to query key/value pairs for JSON keys are as follows:

1. As always with searches, start by defining a `QueryManager`:

```
QueryManager queryMgr = client.newQueryManager();
```

2. For a key value search, define your query in a `KeyValueQueryDefinition` object:

```
KeyValueQueryDefinition kvqdef = queryMgr.newKeyValueDefinition();
```

3. Next, specify a JSON key/value pair to match JSON documents having that pair.

```
kvqdef.put(queryMgr.newKeyLocator("myKey"), "my value");
```

4. Run a search with the query definition object as an argument, returning a `SearchHandle` object:

```
SearchHandle results = queryMgr.search(kvqdef, new SearchHandle());
```

3.4.2 XML Key-Value Searches

The basic steps to query key/value pairs in XML are essentially identical to JSON key-value searches, except for the third step, where instead of using a `KeyLocator` in the query, you use an `ElementLocator`.

1. As always with searches, start by defining a `QueryManager`:

```
QueryManager queryMgr = client.newQueryManager();
```

2. For a key value search, you need to define your query in a `KeyValueQueryDefinition` object:

```
KeyValueQueryDefinition kvqdef = queryMgr.newKeyValueDefinition();
```

3. Next, use `put()` to associate a key-value pair with the query definition as a map. Since this is for a query on XML, it obtains the key value by passing a string of an XML element's name and an XML attribute's name to `newElementLocator()`, which creates and returns an `ElementLocator` used by the `KeyValueQuery`.

```
kvqdef.put(queryMgr.newElementLocator(new QName("myName"), "value");
```

4. Run a search with the query definition object as an argument, returning a handle on the results:

```
SearchHandle results = queryMgr.search(kvqdef, new SearchHandle());
```

The previous example specifies an XML element for the key and matches the value against the exact text content of the element, according to `exact` value semantics and unfiltered search. If you specify an XML attribute for the key, then the key-value search matches the text value of the attribute.

3.5 Search Documents Using Structured Query Definition

Structured queries let you construct and modify complex queries in Java, XML, or JSON. For details, see Searching Using Structured Queries in the *Search Developer's Guide*. This section includes the following parts:

- Ways to Create a Structured Query
- Basic Steps to Define a Structured Query Definition
- Creating a Structured Query From Raw XML or JSON
- Structured Query Examples

3.5.1 Ways to Create a Structured Query

You can create a structured query in XML, in JSON, or using the `StructuredQueryBuilder` object in the Java API.

To specify a structured query directly in XML or JSON, use `RawStructuredQueryDefinition`; for details, see "Creating a Structured Query From Raw XML or JSON" on page 34. If you construct a structured query directly, it is up to you to make sure the query is constructed correctly. Incorrectly constructed queries can result in syntax errors, a query that does not do what you expect, or other exceptions. For syntax details, see Searching Using Structured Queries in the *Search Developer's Guide*.

The `StructuredQueryBuilder` object in the Java API lets you build out a structured query one piece at a time in Java.

3.5.2 Basic Steps to Define a Structured Query Definition

The following are the basic steps needed to define a structured query definition in the Java API:

1. As always with searches, start by defining a `QueryManager`. Like documents, the manager deals with interaction between queries and the database.

```
QueryManager queryMgr = client.newQueryManager();
```

2. For a structured search, define your query in a `StructuredQueryDefinition` object. You can use a `StructuredQueryBuilder` to create the definition.

```
// create a query builder for the query options
StructuredQueryBuilder qb = new StructuredQueryBuilder(OPTIONS_NAME);

// build a search definition
StructuredQueryDefinition querydef =
    qb.and(qb.term("neighborhood"),
           qb.valueConstraint("industry", "Real Estate"));
```

3. Run a search with the `StringQueryDefinition` object as an argument, returning a result handle:

```
SearchHandle results = queryMgr.search(querydef, new SearchHandle());
```

3.5.3 Creating a Structured Query From Raw XML or JSON

To create a structured query from a raw XML or JSON representation, use any handle class that implements `com.marklogic.client.io.marker.StructureWriteHandle`.

The Java API includes `StructureWriteHandle` implementations that support creating a structure in XML or JSON from a string (`StringHandle`), a file (`FileHandle`), stream (`InputStreamHandle`), and popular abstractions (`DOMHandle`, `DOM4JHandle`, `JDOMHandle`). For a complete list of implementations, see the Java API JavaDoc.

Follow this procedure to create a structured query using a handle:

1. Create a `QueryManager` to execute queries on the database.

```
QueryManager queryMgr = client.newQueryManager();
```

2. Create a JSON or XML representation of the query, using a text editor or other tool or library. Use the syntax detailed in Searching Using Structured Queries in the *Search Developer's Guide*. The following example uses `String` for the raw representation:

```
String rawXMLQuery =
    "<search:query "+
          "xmlns:search='http://marklogic.com/appservices/search'>"+
      "<search:term-query>"+
          "<search:text>neighborhoods</search:text>"+
      "</search:term-query>"+
      "<search:value-constraint-query>"+
          "<search:constraint-name>industry</search:constraint-name>"+
          "<search:text>Real Estate</search:text>"+
      "</search:value-constraint-query>"+
    "</search:query>";
```

3. If you express your query in XML, create a handle on your raw query using a class that that implements `StructureWriteHandle`. For example:

```
StringHandle rawHandle = new StringHandle(rawXMLQuery);
```

4. If you express your query in JSON, create a handle using a class that implements `StructureWriteHandle`, set the handle content format to JSON, and associate your query with the handle.

```
StringHandle rawHandle = new StringHandle();
rawHandle.withFormat(Format.JSON).set(rawJSONQuery);
```

5. Create a `RawStructuredQueryDefinition` from the handle. Optionally, include the name of persistent query options. For example:

```
// Use the default persistent query options
RawStructuredQueryDefinition querydef =
    queryMgr.newRawStructuredQueryDefinition(rawHandle);

// Use the persistent options previously saved as "myoptions"
RawStructuredQueryDefinition querydef =
    queryMgr.newRawStructuredQueryDefinition(rawHandle, "myoptions");
```

6. Perform a search using the `RawStructuredQueryDefinition` and a results handle.

```
SearchHandle resultsHandle =
    queryMgr.search(querydef, new SearchHandle());
```

3.5.4 Structured Query Examples

This section shows some structured query examples, showing the XML for a structured query and the corresponding Java code using `StructuredQueryBuilder`. You can put each of these examples in context by inserting the `StructuredQueryDefinition` line in the following code:

```
QueryManager queryMgr = dbClient.newQueryManager();
StructuredQueryBuilder sb =
   queryMgr.newStructuredQueryBuilder("myopt");

// put code from examples here
```

```
StructuredQueryDefinition criteria =
   ... example of building query definition ...
// end code from examples

StringHandle searchHandle =
  queryMgr.search(
    criteria, new StringHandle()).get();
```

Additionally, these examples use query options from the following code:

```
QueryOptionsManager optionsMgr =
  dbClient.newServerConfigManager().newQueryOptionsManager();
QueryOptionsBuilder ob = new QueryOptionsBuilder();

QueryOptionsHandle queryOptions = new QueryOptionsHandle();
  queryOptions.withConstraints(
    ob.constraint("date",
      ob.range(
        ob.elementRangeIndex(
          new QName("http://purl.org/dc/elements/1.1/","date"),
            ob.rangeType("xs:date")
        )
      )
    ),
    ob.constraint("popularity",
      ob.range(
        ob.elementRangeIndex(
          new QName("popularity"),
          ob.rangeType("xs:int")
        )
      )
    ),
  ob.constraint("title",
    ob.word(
      ob.elementTermIndex(
        new QName("title")
      )
    )
  )
);

queryOptions.setTransformResults(ob.rawResults());
// queryOptions.setReturnQuery(true);
queryOptions.setReturnResults(true);
// queryOptions.setDebug(true);

optionsMgr.writeOptions("myopt", queryOptions);
```

This section contains the following examples:

- Example: Date Range Structured Query
- Example: Element Index Structured Query

- Example: Property Value Structured Query
- Example: Property Value Structured Query
- Example: Document Structured Query
- Example: Collection Structured Query

3.5.4.1 Example: Date Range Structured Query

For the boilerplate code environment in which this example runs, see the code snippet in “Structured Query Examples” on page 35.

The following example searches for the `"2005-01-01"` value in the date range index.

```
StructuredQueryDefinition criteria =
   sb.rangeConstraint("date", Operator.EQ, "2005-01-01");

/* XML equivalent
<search:query xmlns:search=
   "http://marklogic.com/appservices/search">
  <search:range-constraint-query>
    <search:constraint-name>date</search:constraint-name>
    <search:value>2005-01-01</search:value>
  </search:range-constraint-query>
</search:query>
*/
```

3.5.4.2 Example: Element Index Structured Query

For the boilerplate code environment in which this example runs, see the code snippet in “Structured Query Examples” on page 35.

The following example searches for the `"Bush"` value within an element range index on `title`.

```
StructuredQueryDefinition criteria =
   sb.wordConstraint("title", "Bush");

/* XML equivalent
<search:query xmlns:search=
   "http://marklogic.com/appservices/search">
  <search:word-constraint-query>
    <search:constraint-name>title</search:constraint-name>
      <search:text>Bush</search:text>
  </search:word-constraint-query>
</search:query>
*/
```

3.5.4.3 Example: Property Value Structured Query

For the boilerplate code environment in which this example runs, see the code snippet in “Structured Query Examples” on page 35.

The following example searches for the `"hello"` term in the value of any property.

```
StructuredQueryDefinition criteria =
   sb.properties(sb.term("hello"));

/* XML equivalent
<search:query xmlns:search=
   "http://marklogic.com/appservices/search">
  <search:properties-query>
    <search:term-query>
      <search:text>hello</search:text>
    </search:term-query>
  </search:properties-query>
</search:query>
*/
```

3.5.4.4 Example: Property Value Structured Query

For the boilerplate code environment in which this example runs, see the code snippet in “Structured Query Examples” on page 35.

The following example searches for documents in the `"http://testdoc/doc6/"` directory.

```
StructuredQueryDefinition criteria =
   sb.directory(true, "http://testdoc/doc6/");

/* XML equivalent
<search:query xmlns:search=
   "http://marklogic.com/appservices/search">
  <search:directory-query>
    <search:uri>
      <search:text>http://testdoc/doc6/</search:text>
    </search:uri>
  </search:directory-query>
</search:query>
*/
```

3.5.4.5 Example: Document Structured Query

For the boilerplate code environment in which this example runs, see the code snippet in “Structured Query Examples” on page 35.

The following example searches for the `"http://testdoc/doc6/"` document.

```
StructuredQueryDefinition criteria =
   sb.document("http://testdoc/doc2");

/* XML equivalent
<search:query xmlns:search=
   "http://marklogic.com/appservices/search">
  <search:document-query>
    <search:uri>
      <search:text>http://testdoc/doc2</search:text>
    </search:uri>
  </search:document-query>
</search:query>
*/
```

3.5.4.6 Example: Collection Structured Query

For the boilerplate code environment in which this example runs, see the code snippet in “Structured Query Examples” on page 35.

The following example searches documents belonging to the `"http://test.com/set3/set3-1"` collection.

```
StructuredQueryDefinition criteria =
   sb.collection("http://test.com/set3/set3-1");

/* XML equivalent
<search:query xmlns:search=
   "http://marklogic.com/appservices/search">
   <search:collection-query>
    <search:uri>
      <search:text>http://test.com/set3/set3-1</search:text>
      </search:uri>
  </search:collection-query>
</search:query>
*/
```

3.6 Apply Dynamic Query Options to Document Searches

You can use a combined query to specify query options on the fly, without first persisting them as named options. A *combined query* is an XML or JSON wrapper around a string query and/or structured query, plus query options.

The following example is a combined query that corresponds to the string query "cat AND dog" plus dynamically setting the `return-query` query option to true. For syntax details, see Syntax in the *REST Application Developer's Guide*.

```
<search xmlns="http://marklogic.com/appservices/search">
  <qtext>cat AND dog</qtext>
  <options>
    <return-query>true</return-query>
  </options>
</search>
```

Combined queries are useful for rapid prototying during development, and for applications that need to modify query options on a per query basis. The `RawCombinedQueryDefinition` class represents a combined query in the Java API.

This section covers the following topics:

- Creating a Combined Query
- Interaction with Persistent Query Options
- Performance Considerations

3.6.1 Creating a Combined Query

You can only create a combined query from raw XML or JSON; there is no builder class. Follow the procedure below to bind a handle on the raw representation to a `RawCombinedQueryDefinition` object usable for searching.

To create a combined query, use any handle class that implements `com.marklogic.client.io.marker.StructureWriteHandle`. The Java API includes `StructureWriteHandle` implementations that support creating a structure in XML or JSON from input sources such as a string (`StringHandle`), a file (`FileHandle`), a stream (`InputStreamHandle`), and popular abstractions (`DOMHandle`, `DOM4JHandle`, `JDOMHandle`). For a complete list of implementations, see the Java API JavaDoc.

Follow this procedure to create a combined query using a handle:

1. Create a `QueryManager` to execute queries on the database.

```
QueryManager queryMgr = client.newQueryManager();
```

2. Create a JSON or XML representation of the query, using a text editor or other tool or library. For syntax details, see Syntax in the *REST Application Developer's Guide*. The following example uses `String` for the raw representation:

```
String rawXMLQuery =
    "<search:search "+
        "xmlns:search='http://marklogic.com/appservices/search'>"+
```

```
    "<search:query>"+
      "<search:term-query>"+
        "<search:text>neighborhoods</search:text>"+
      "</search:term-query>"+
      "<search:value-constraint-query>"+
        "<search:constraint-name>industry</search:constraint-name>"+
        "<search:text>Real Estate</search:text>"+
      "</search:value-constraint-query>"+
    "</search:query>"+
    "<search:options>"+
      "<search:constraint name='industry'>"+
        "<search:value>"+
          "<search:element name='industry' ns=''/>"+
        "</search:value>"+
      "</search:constraint>"+
    "</search:options>"+
  "</search:search>";
```

3. If you express your query in XML, create a handle on your raw query, using a class that implements `StructureWriteHandle`. For example:

```
StringHandle rawHandle = new StringHandle(rawXMLQuery);
```

4. If you express your query in JSON, create a handle using a class that implements `StructureWriteHandle`, set the handle content format to JSON, and associate your raw query with the handle.

```
StringHandle rawHandle = new StringHandle();
rawHandle.withFormat(Format.JSON).set(rawJSONQuery);
```

5. Create a `RawCombinedQueryDefinition` from the handle. Optionally, include the name of persistent query options. For example:

```
// Use the default persistent query options
RawCombinedQueryDefinition querydef =
    queryMgr.newRawCombinedQueryDefinition(rawHandle);

// Use persistent options previously saved as "myoptions"
RawCombinedQueryDefinition querydef =
    queryMgr.newRawCombinedQueryDefinition(rawHandle, "myoptions");
```

6. Perform a search using the `RawCombinedQueryDefinition` and a results handle.

```
SearchHandle resultsHandle =
    queryMgr.search(querydef, new SearchHandle());
```

For a complete example of searching with a combined query, see `com.marklogic.client.example.cookbook.RawCombinedSearch` in the `example/` directory of your Java API installation.

3.6.2 Interaction with Persistent Query Options

Dynamic query options supplied in a combined query are merged with persistent and default options that are in effect for the search. If the same non-constraint option is specified in both the combined query and persistent options, the setting in the combined query takes precedence.

Constraints are overridden by name. That is, if the dynamic and persistent options contain a `<constraint/>` element with the same `@name`, the definition in the dynamic query options is the one that applies to the query. Two constraints with different name are both merged into the final options.

For example, suppose the following query options are installed under the name `my-options`:

```
<options xmlns="http://marklogic.com/appservices/search">
  <fragment-scope>properties</fragment-scope>
  <return-metrics>false</return-metrics>
  <constraint name="same">
    <collection prefix="http://server.com/persistent/"/>
  </constraint>
  <constraint name="not-same">
    <element-query name="title" ns="http://my/namespace" />
  </constraint>
</options>
```

Further, suppose you use the following raw XML combined query to define dynamic query options:

```
<search xmlns="http://marklogic.com/appservices/search">
  <options>
    <return-metrics>true</return-metrics>
    <debug>true</debug>
  <constraint name="same">
    <collection prefix="http://server.com/dynamic/"/>
  </constraint>
    <constraint name="different">
      <element-query name="scene" ns="http://my/namespace" />
    </constraint>
  </options>
</search>
```

You can create a `RawQueryDefinition` that encapsulates the combined query and the persistent options:

```
StringHandle rawQueryHandle =
    new StringHandle(...);
RawCombinedQueryDefinition querydef =
    queryMgr.newRawCombinedQueryDefinition(
        rawQueryHandle, "my-options");
```

The query is evaluated with the following merged options. The persistent options contribute the `fragment-scope` option and the constraint named `not-same`. The dynamic options in the combined query contribute the `return-metrics` and `debug` options and the constraints named `same` and `different`. The `return-metrics` setting and the constraint named `same` from `my-options` are discarded.

```
<options xmlns="http://marklogic.com/appservices/search">
  <fragment-scope>properties</fragment-scope>
  <return-metrics>true</return-metrics>
  <debug>true</debug>
  <constraint name="same">
    <collection prefix="http://server.com/dynamic/"/>
  </constraint>
    <constraint name="different">
      <element-query name="scene" ns="http://my/namespace" />
    </constraint>
  </options>
  <constraint name="not-same">
    <element-query name="title" ns="http://my/namespace" />
  </constraint>
</options>
```

3.6.3 Performance Considerations

Using persistent query options usually performs better than using dynamic query options. In most cases, the performance difference between the two methods is slight.

When MarkLogic Server processes a combined query, the per request query options must be parsed and merged with named and default options on every search. When you only use persistent named or default query options, you reduce this overhead.

If your application does not require dynamic per-request query options, you should use a `QueryOptionsManager` to persist your options under a name and associate the options with a simple `StringQueryDefinition` or `StructuredQueryDefinition`.

3.7 Search On Tuples (Tuples Query / Values Query)

You can return values and tuples (co-occurrences) through the Java API. Value and tuple searches require the appropriate range indexes are configured on your MarkLogic Server database. For background on values and co-occurrences, see Browsing With Lexicons in the *Search Developer's Guide*.

This section includes the following parts:

- Values Search
- Tuples Search

3.7.1 Values Search

The following returns values through the Java API:

The following are the basic steps to search on values:

1. As always with searches, start by defining a `QueryManager`.

```
QueryManager queryMgr = client.newQueryManager();
```

2. For a values search, define your query in a `ValuesDefinition` object. As mentioned, just assume the options argument, "`valueoptions`" has defined the `"myvalue"` value.

```
// build a search definition
ValuesDefinition vdef = queryMgr.newValuesDefinition("myvalue",
                                                     "valuesoptions");
```

3. `ValuesDefinition` has a number of associated properties and values relating to a tuples or values search. In particular, `setAggregate()` sets the name of the aggregate function to be applied as part of the query.

```
vdef.setAggregate("correlation", "covariance");
```

4. Run a search with the `ValuesDefinition` object as an argument, returning a `ValuesHandle` object. Note that the tuples search method is called `values()`, not `search()`.

```
ValuesHandle results = queryMgr.values(vdef, new ValuesHandle());
```

For more information on tuples search concepts, see Returning Lexicon Values With search:values and Browsing With Lexicons in the *Search Developer's Guide*.

3.7.2 Tuples Search

The following returns tuples (co-occurrences) through the Java API:

1. As always with searches, start by defining a `QueryManager`.

```
QueryManager queryMgr = client.newQueryManager();
```

2. For a tuples search, define your query in a `ValuesDefinition` object. As mentioned, just assume the options argument, "`tupleoptions`" has defined the `"co"` tuple.

```
// build a search definition
ValuesDefinition vdef = queryMgr.newValuesDefinition("co",
                                                     "tupleoptions");
```

3. Run a search with the `ValuesDefinition` object as an argument, returning a `TuplesHandle` object. Note that the tuples search method is called `tuples()`, not `search()`.

```
TuplesHandle results = queryMgr.tuples(vdef, new TuplesHandle());
```

For more information on tuples search concepts, see Returning Lexicon Values With search:values and Browsing With Lexicons in the *Search Developer's Guide*.

3.8 Limiting A Search To Specific Collections And/Or A Directory

All query definitions have `setCollections()` and `setDirectory()` methods. By calling `setDirectory(`*`directory_URI_string`*`)` on your query definition, you limit your search to that directory. By calling `setCollections(`*`list_of_collection_name_strings`*`)` on your query definition, you limit your search to those collections. Of course, you can call both and limit your search to collections and a single directory.

4.0 Query Options

This chapter describes how to use, write, read, and delete *query options*. In the MarkLogic XQuery Search API, a query options object is called an *options node*.

This chapter contains the following sections:

- Using Query Options
- Default Query Options
- Using QueryOptionsManager To Delete, Write, and Read Options
- Using Query Options With Search
- Using QueryOptionsBuilder
- Creating Persistent Query Options From Raw JSON or XML
- Validating Query Options With setQueryOptionValidation()
- Query Options Examples

For details on each of the query options, see Appendix A: JSON Query Options Reference in the *REST Application Developer's Guide* for their JSON format version and Appendix B: XML Query Options Reference in the *REST Application Developer's Guide* for their XML format version. While there are a large number of options, in order to configure your searches properly and build persistent query options, you will need to familiarize yourself with them.

4.1 Using Query Options

Query options let you specify a set of options for search and apply them repeatedly to multiple searches. The individual options can specify the following:

- Define constraints that do not require indexes, such as word, value and element constraints.
- Define constraints that do require indexes, such as collection, field-value, and other range constraints.
- Control search characteristics such as case sensitivity and ordering.
- Extend the search grammar.
- Customize query results including pagination, snippeting, and filtering.

Query options can be persistent or dynamic. Persistent query options are stored on the REST Server and referenced by name in future queries. Dynamic query options are options created on a per-request basis. Choosing between the two is a trade off between flexibility and performance: Dynamic query options are the more flexible, but persistent query options usually provide better performance. You can use both persistent and dynamic query options in the same query. Dynamic query options are only available for operations that accept a `RawCombinedQueryDefinition`. For details, see "Apply Dynamic Query Options to Document Searches" on page 39.

Use a `QueryOptionsManager` object to manage persistent query options and store them on the REST Server. To see individual option values, use the appropriate `get()` command on a `QueryOptionsHandle`.

The persistent query options are the static part of the query, and are generally used for many different queries. For example, you might have a range constraint specified in saved query options that specifies a date range index that contains date values that you will facet on.

Additionally, most queries have a component that is constructed dynamically by your Java code. For example, you might change the result page, the query criteria (terms, facet values, and so on), or other dynamic parts of the query. The static and dynamic parts of the query are merged together during a search.

For details on specific query options, see Appendix A: JSON Query Options Reference and Appendix B: XML Query Options Reference in the *REST Application Developer's Guide*. While there are a large number of options, in order to configure your searches properly, you will need to familiarize yourself with them.

4.2 Default Query Options

The MarkLogic Java API comes with preexisting query options called `default`. It acts just like any other options and is used if options are not specified elsewhere. You can read it into a handle, change values, and write it back out, where it will still be used as the default query options. While changing its values should not be done casually, this can be very useful if your site needs different default behaviors.

The default options are selected for optimal performance; searches run unfiltered, and document quality is not taken into consideration during scoring. If you install different default options, consider including the following options unless your application requires filtering or the use of document quality.

```
<options xmlns="http://marklogic.com/appservices/search">
  <search_option>unfiltered</search-option>
  <quality-weight>0</quality-weight>
</options>
```

If you delete `default`, the server will fall back to its own defaults.

4.3 Using QueryOptionsManager To Delete, Write, and Read Options

Use `com.marklogic.client.admin.QueryOptionsManager` to manage persistent query options that are stored on the REST server.

As always in the Java API, interactions with the database are done via a manager object, in this case `QueryOptionsManager`. Since query options are associated with the REST server configuration, to create a `QueryOptionsManager` you call `newQueryOptionsManager()` on a `ServerConfigManager` object.

As with all operations on `ServerConfigManager`, an application must authenticate as `rest-admin`. Note that any application that authenticates as `rest-reader` and `rest-writer` can use query options, but to write or delete them from the server requires `rest-admin`.

```
// create a manager for writing, reading, and deleting query options
QueryOptionsManager qoManager=
            client.newServerConfigManager().newQueryOptionsManager();
```

The simplest `QueryOptions` operation is deleting a stored one:

```
qoManager.deleteOptions("myqueryoptions");
```

To either read query options from the database or write query options to the database, you need a handle object that implements the `QueryOptionsHandle` interface. The API includes several handle classes that implement `QueryOptionsHandle`, including `StringHandle`, `BytesHandle`, `DOM4JHandle`, and `DOMHandle`. These interfaces allow you to work with query options as raw strings, XML, and JSON.

The following example reads in the options configuration called `myqueryoptions` from the server, then writes it out again.

```
qoHandle = new QueryOptionsReadHandle();

// read a query option configuration from the database
// qoHandle now contains the query option
// "myqueryoptions"
qoManager.readOptions("myqueryoptions", qoHandle);

// write the query option to the database
qoManager.writeOptions("myqueryoptions", qoHandle);
```

You can get a list of all named `QueryOptions` from the server via the `QueryOptionsListHandle` object:

```
QueryManager queryMgr = dbclient.newQueryManager();
QueryOptionsHandle qolHandle = new QueryOptionsListHandle();
queryMgr.optionsList(qolHandle);
Set<String> results = qolHandle.getValuesMap().keySet();
```

4.4 Using Query Options With Search

You can customize a query with query options in the following ways:

1. Create persistent query options, save them to the REST server with an associated name, and then reference them by name when you construct a query. To use the default query options, omit an options name when you construct the query. The following example creates a string query that uses the options stored as “myoptions”:

   ```
   // Create a string query that uses persistent query options
   QueryManager qMgr = new QueryManager();
   ```

```
StringQueryDefinition qDef = qMgr.newStringDefinition("myoptions");
...
qMgr.search(qDef, resultsHandle);
```

2. Embed dynamic query options in a combined query definition.

You can use both persistent and dynamic query options in the same search by including a query options name when constructing a combined query (`RawCombinedQueryDefinition`).

Persistent query options must be stored on the REST server before you can use them in a search. For details, see "Using QueryOptionsManager To Delete, Write, and Read Options" on page 47.

To construct persistent query options, use a handle class that implements `QueryOptionsHandle`, such as `StringHandle` or `DOMHandle`, or use the `QueryOptionsBuilder` class. Using a handle, you can create query options directly in XML or JSON; for details, see "Creating Persistent Query Options From Raw JSON or XML" on page 50. When you use `QueryOptionsBuilder`, the internal XML representation is created for you.

Warning The use of `QueryOptionsBuilder` is deprecated. This class will be removed in a future release.

To construct dynamic query options, use a handle that implements `StructureWriteHandle`, such as `StringHandle` or `DOMHandle` to create a combined query that includes an options component, then associate the handle with a `RawCombinedQueryDefinition`. For details, see "Apply Dynamic Query Options to Document Searches" on page 39.

4.5 Using QueryOptionsBuilder

Warning The use of `QueryOptionsBuilder` is deprecated. This class will be removed in a future release. For the preferred method, see "Creating Persistent Query Options From Raw JSON or XML" on page 50.

The `QueryOptionsBuilder` class simplifies constructing a query options handle. To start using the `QueryOptionsBuilder` class, define a `QueryOptionsHandle` and a `QueryOptionsBuilder` object.

```
builder = new QueryOptionsBuilder();
options = new QueryOptionsHandle();
```

Once you have your builder and handle objects, start with the handle. To populate the handle with options and values, you will use the either the `set` methods or the `with` methods. The set methods set values on the object, and the `with` methods `set` values and return the object.

The `QueryOptionsBuilder` object provides parameters for these methods. Consider a simple example of defining a word constraint:

```
options.setConstraints(builder.constraint("word",
        builder.word(builder.elementTermIndex(
                     new QName("Summary")) )));
```

Inside the `setConstraints()`, you want to add a constraint option and its value. That means you need to build a constraint, so you use the `builder.constraint()` method. A constraint needs a name and a source, and `builder.constraint()`'s signature is a word index specification. Looking at the JavaDoc or your Java tools for `QueryOptionsBuilder`, its `word` method builds an option and value to specify a word index.

To build up the Java expression, start with the following:

```
options.setConstraints(
  builder.constraint(..., ...));
```

The first argument is easy; it is a string for a constraint name, which results in the following:

```
options.withConstraints(
  builder.constraint("word", ...));
```

For the second argument, we need a term word index, and looking at the available methods and their JavaDoc, you use `builder.word()` to build the following:

```
options.withConstraints(
  builder.constraint("word",
                     builder.word(...)));
```

As described in the JavaDoc, `word()` takes a `TermSpec` for the index backing the word options. And `QueryOptionsBuilder` has an `elementTermIndex()` method that will build one for you:

```
options.withConstraints(
  builder.constraint("word",
                    builder.word(builder.elementTermIndex(...))));
```

The `elementTermIndex()` takes one argument, a QName of the name of the indexed element. That is something simple enough that it is not necessary to build it, so you can just use the name as the argument as follows:

```
options.withConstraints(
  builder.constraint("word", builder.word(
                builder.elementTermIndex(new QName("Summary")))));
```

The result define query options consisting of a constraint.

4.6 Creating Persistent Query Options From Raw JSON or XML

To create persistent query options from a raw XML or JSON representation, use any handle class that implements `com.marklogic.client.io.marker.QueryOptionsWriteHandle`. Follow this procedure to create persistent query options using a handle:

1. Create a JSON or XML representation of the query options, using the tools of your choice. The following example uses a String representation:

```
String xmlOptions =
  "<search:options "+
        "xmlns:search='http://marklogic.com/appservices/search'>"+
      "<search:constraint name='industry'>"+
        "<search:value>"+
          "<search:element name='industry' ns=''/>"+
        "</search:value>"+
      "</search:constraint>"+
  "</search:options>";
```

2. If you are using XML query options, create a handle that implements `QueryOptionsWriteHandle` and associate your options with the handle. For example:

```
StringHandle writeHandle = new StringHandle(xmlOptions);
```

3. If you are using JSON query options, create a handle that implements `QueryOptionsWriteHandle`, set the handle content format to JSON, and associate your options with the handle. For example:

```
StringHandle writeHandle = new StringHandle();
writeHandle.withFormat(Format.JSON).set(jsonOptions);
```

4. Save the options to the REST server using `QueryOptionsManager.writeOptions`. For example:

```
optionsMgr.writeOptions(optionsName, writeHandle);
```

For a complete example, see `com.marklogic.client.example.cookbook.QueryOptions` in the following directory of the Java API distribution:

```
example/com/marklogic/client/example/cookbook
```

The Java API includes `QueryOptionsWriteHandle` implementations that support constructing query options as XML or JSON using several alternatives to `String`. These alternatives include reading from a file (`FileHandle`) or stream (`InputStreamHandle`), and popular abstractions, such as DOM, DOM4J, and JDOM. For details, see the Java API JavaDoc.

You can use any handle that implements `QueryOptionsReadHandle` to fetch previously persisted query options from the REST server. The following example fetches the JSON representation of query options into a `String` object:

```
StringHandle jsonStringHandle = new StringHandle();
jsonStringHandle.setFormat(Format.JSON);

qoManager.readOptions("jsonoptions", jsonStringHandle);
```

4.7 Validating Query Options With setQueryOptionValidation()

Query options can be complex. By default, the server validates query options before writing them out to a database. This takes a small amount of time, but because the query options are usually created once and then persisted, it does not really make a difference.

If you do try to write out an invalid query options and validation is enabled (which is the default), you get a `400` error from the server and a `FailedRequestException` thrown.

If you want to turn validation off, you can do so by calling the following right after you create your `ServerConfigurationManager` object:

```
ServerConfigurationManager.setQueryOptionValidation(false)
```

Note that if validation is disabled and you have query options that turn out to be invalid, your searches will still run, but any invalid options will be ignored. For example, if you define an invalid constraint and then try to use it in a search, the search will run, but the constraint will not be used. The search results will contain a warning in cases where a constraint is not used. You can access those warnings via `SearchHandle.getWarnings()`.

4.8 Query Options Examples

This section shows some examples of setting query options in the Java API, showing the corresponding XML query options for reference along with the Java code. You can also specify query options as a JSON string. For a list of the XML and JSON query options, see Appendix A: JSON Query Options Reference in the *REST Application Developer's Guide* for their JSON format version and Appendix B: XML Query Options Reference in the *REST Application Developer's Guide* for their XML format version.

This section includes the following examples:

- Example: Values and Tuples Query Options
- Example: Field Constraint Query Options
- Example: builder.configure Collection Constraint Query Options
- Example: Path Range Index Constraint Query Options
- Example: Element Attribute Range Constraint Query Options
- Example: Geospatial Constraint Query Options

4.8.1 Example: Values and Tuples Query Options

This example sets up configurations:

- A values tuple `/v1/values/pop`, which gets the values from the element range index from `xs:QName("popularity")` (in no namespace), and is typed as an `xs:int`.

- A values tuple /`v1/values/score`, which gets the values from the element range index from a QName with namespace `"http://test.aggr.com"` and localname `"score"`, typed as an `xs:decimal`.
- A tuple `/v1/values/pop-rate-tups`, which combines the range index from from `xs:QName("popularity")` (in no namespace), and typed as an `xs:int`, and the range index from a QName with namespace `"http://test.aggr.com"` and localname `"score"`, typed as an `xs:decimal`.

The following is the Java code for the query options, along with a sample of what the XML node would look like for the corresponding query options.

```
// set up for the query options
QueryOptionsHandle handle = new QueryOptionsHandle();
QueryOptiondBuilder builder = new QueryOptionsBuilder();
// construct the options for the example
handle.withValues(
builder.values("pop",
builder.range(builder.elementRangeIndex(
  new QName("popularity"),
   builder.rangeType("xs:int")))),
   builder.values("score",
   builder.range(builder.elementRangeIndex(
      new QName("http://test.aggr.com", "score"),
    builder.rangeType("xs:decimal")))))
  .withTuples(
    builder.tuples("pop-rate-tups",
    builder.tupleSources(
    builder.range(builder.elementRangeIndex(
      new QName("popularity"), builder.rangeType("xs:int"))),
    builder.range(builder.elementRangeIndex(
      new QName("http://test.tups.com", "rate"),
    builder.rangeType("xs:int"))))));

/* Corresponding <search:option> node
<search:options
  xmlns:search="http://marklogic.com/appservices/search">
  <search:values name="pop-aggr">
    <search:range type="xs:int">
      <search:element ns="" name="popularity"/>
    </search:range>
  </search:values>
  <search:values name="score-aggr">
    <search:range type="xs:decimal">
      <search:element ns="http://test.aggr.com" name="score"/>
    </search:range>
  </search:values>
  <search:tuples name="pop-rate-tups">
    <search:range type="xs:int">
      <search:element ns="" name="popularity"/>
    </search:range>
    <search:range type="xs:int">
      <search:element ns="http://test.tups.com" name="rate"/>
```

```
    </search:range>
  </search:tuples>
</search:options>
*/
```

4.8.2 Example: Field Constraint Query Options

This example constructs a simple options node that sets up just one constraint, based on the field named `"bbqtext"`. It creates a word constraint, therefore when you search for

```
summary:"hot dog"
```

It constrains the search to documents that have the phrase `"hot dog"` in the field called `"bbqtext"`.

The following is the Java code for the query options, along with a sample of what the XML node would look like for the corresponding query options.

```
// set up for the query options
QueryOptionsHandle handle = new QueryOptionsHandle();
QueryOptiondBuilder builder = new QueryOptionsBuilder();
// construct the options for the example
handle.withConstraints(
  builder.constraint("summary",
    builder.word(
      builder.fieldTermIndex("bbqtext"))));

/* Corresponding <search:option> node
<search:options
    xmlns:search="http://marklogic.com/appservices/search">
    <search:constraint name="summary">
      <search:word>
        <search:field name="bbqtext"/>
      </search:word>
    </search:constraint>
</search:options>
*/
```

4.8.3 Example: builder.configure Collection Constraint Query Options

This example shows three things:

- It includes a `builder.configure()` argument, which is how you set flags and atomic values.
- It uses a stock builder function to set return results to raw documents.
- It includes a collection constraint named `"coll"` that uses the collection URIs with `"http://test.com"` stripped off.

The following is the Java code for the query options, along with a sample of what the XML node would look like for the corresponding query options.

```
// set up for the query options
QueryOptionsHandle handle = new QueryOptionsHandle();
QueryOptiondBuilder builder = new QueryOptionsBuilder();
// construct the options for the example
handle.withConfiguration(
  builder.configure()
    .returnMetrics(false)
    .debug(true)
    .returnQtext(false))
    .withTransformResults(builder.rawResults())
    .withConstraints(
      builder.constraint("coll",
        builder.collection("http://test.com")));
/* Corresponding <search:option> node
<search:options
    xmlns:search="http://marklogic.com/appservices/search">
  <search:debug>true</search:debug>
  <search:constraint name="coll">
    <search:collection prefix="http://test.com"/>
  </search:constraint>
  <search:return-metrics>false</search:return-metrics>
  <search:return-qtext>false</search:return-qtext>
  <search:transform-results apply="raw">
    <search:preferred-elements/>
  </search:transform-results>
</search:options>
*/
```

4.8.4 Example: Path Range Index Constraint Query Options

This example shows how to configure a constraint with a path range index. With this in place, searching for:

```
pindex:low
```

Searches for values less than 5 from the nodes at `/Employee/fn` (in no namespace). It is a string range index, faceted, scoped to documents, with a nice looking unicode label. The facet values are returned with any search.

The following is the Java code for the query options, along with a sample of what the XML node would look like for the corresponding query options.

```
// set up for the query options
QueryOptionsHandle handle = new QueryOptionsHandle();
QueryOptiondBuilder builder = new QueryOptionsBuilder();
// construct the options for the example
handle.withConstraints(

builder.constraint("pindex",
  builder.range(
    builder.pathIndex("/Employee/fn", null,
      builder.stringRangeType("http://marklogic.com/collation/")),
```

```
        Facets.FACETED,
        FragmentScope.DOCUMENTS,
          builder.buckets(
            builder.bucket("low", "0 to 5", "5", null),
            builder.bucket("medium", "5 to 10", "5", "10"),
            builder.bucket("high", "10 to 15", "10", "15"))
)));
/* Corresponding <search:option> node
<search:options
    xmlns:search="http://marklogic.com/appservices/search">
  <search:constraint name="pindex">
    <search:range type="xs:string" facet="true"
       collation="http://marklogic.com/collation/">
      <search:path-index>/Employee/fn</search:path-index>
      <search:fragment-scope>documents</search:fragment-scope>
      <search:bucket name="low" ge="5">0 to 5</search:bucket>
      <search:bucket name="medium" lt="10" ge="5"
         >5 to 10</search:bucket>
      <search:bucket name="high" lt="15" ge="10"
         >10 to 15</search:bucket>
    </search:range>
  </search:constraint>
</search:options>
*/
```

4.8.5 Example: Element Attribute Range Constraint Query Options

This example shows an element attribute range index, with computed buckets. When you search, the facets will be filled out depending on the values of `{http://example.com}entry/@date`

The following is the Java code for the query options, along with a sample of what the XML node would look like for the corresponding query options.

```
// set up for the query options
QueryOptionsHandle handle = new QueryOptionsHandle();
QueryOptiondBuilder builder = new QueryOptionsBuilder();
// construct the options for the example
handle.withConstraints(

builder.constraint("date",
  builder.range(
    builder.elementAttributeRangeIndex(
      new QName("http://example.com","entry"),
      new QName("date"),
      builder.rangeType("xs:dateTime")),
  Facets.FACETED,
  FragmentScope.DOCUMENTS,
  builder.buckets(
    builder.computedBucket("older", "Older", null, "-P1Y",
      AnchorValue.START_OF_YEAR),
    builder.computedBucket("year", "This Year", "P0Y", "P1Y",
      AnchorValue.START_OF_YEAR), builder.computedBucket("month",
        "This Month", "P1M", "P0M",
```

```
      AnchorValue.START_OF_MONTH),
        builder.computedBucket("today", "Today", "P1D", "P0D",
        AnchorValue.START_OF_DAY),
        builder.computedBucket("future", "Future", "P0D", null,
        AnchorValue.NOW)))));
/* Corresponding <search:option> node
<search:options
    xmlns:search="http://marklogic.com/appservices/search">
  <search:constraint name="date">
    <search:range type="xs:dateTime" facet="true">
      <search:attribute ns="" name="date"/>
      <search:element ns="http://example.com" name="entry"/>
      <search:fragment-scope>documents</search:fragment-scope>
      <search:computed-bucket name="older" lt="-P1Y"
        anchor="start-of-year">Older</search:computed-bucket>
      <search:computed-bucket name="year" lt="P1Y" ge="P0Y"
        anchor="start-of-year">This Year</search:computed-bucket>
      <search:computed-bucket name="month" lt="P0M" ge="P1M"
        anchor="start-of-month">This Month</search:computed-bucket>
      <search:computed-bucket name="today" lt="P0D" ge="P1D"
        anchor="start-of-day">Today</search:computed-bucket>
      <search:computed-bucket name="future" ge="P0D"
        anchor="now">Future</search:computed-bucket>
    </search:range>
  </search:constraint>
</search:options>
*/
```

4.8.6 Example: Geospatial Constraint Query Options

This example shows a geospatial constraint query options. In addition to having a configuration setting (with pagelength), this sets up three geospatial constraints and three element constraints.

The following is the Java code for the query options, along with a sample of what the XML node would look like for the corresponding query options.

```
// set up for the query options
QueryOptionsHandle handle = new QueryOptionsHandle();
QueryOptiondBuilder builder = new QueryOptionsBuilder();
// construct the options for the example
handle.withConfiguration(
  builder.configure().
  returnMetrics(false).
  debug(true).
  pageLength(25L))
  .withConstraints(
     builder.constraint("geo-elem",
       builder.geospatial(builder.elementGeospatialIndex(new
         QName("g-elem-point")))),
       builder.constraint("geo-elem-pair",
         builder.geospatial(
           builder.elementPairGeospatialIndex(
             new QName("g-elem-pair"),
```

```
              new QName("lat"),
              new QName("long")))),
        builder.constraint("geo-attr-pair",
          builder.geospatial(
            builder.attributePairGeospatialIndex(
              new QName("g-attr-pair"),
              new QName("lat"),
              new QName("long")))),
        builder.constraint("eq-name",
          builder.elementQuery(new QName("name"))),
        builder.constraint("eq-person",
          builder.elementQuery(new QName("person"))),
        builder.constraint("eq-desc",
          builder.elementQuery(new QName("description"))));
/* Corresponding <search:option> node
<search:options
    xmlns:search="http://marklogic.com/appservices/search">
  <search:debug>true</search:debug>
  <search:return-metrics>false</search:return-metrics>
  <search:page-length>25</search:page-length>
  <search:constraint name="geo-elem">
    <search:geo-elem>
      <search:element ns="" name="g-elem-point"/>
    </search:geo-elem>
    </search:constraint>
    <search:constraint name="geo-elem-pair">
      <search:geo-elem-pair>
        <search:lat ns="" name="lat"/>
        <search:lon ns="" name="long"/>
        <search:parent ns="" name="g-elem-pair"/>
    </search:geo-elem-pair>
  </search:constraint>
  <search:constraint name="geo-attr-pair">
    <search:geo-attr-pair>
      <search:lat ns="" name="lat"/>
      <search:lon ns="" name="long"/>
      <search:parent ns="" name="g-attr-pair"/>
    </search:geo-attr-pair>
  </search:constraint>
</search:options>
*/
```

5.0 Transactions and Optimistic Locking

This chapter covers two different ways for locking documents during MarkLogic Server operations, *multi-statement transactions* and *optimistic locking.*

This chapter includes the following sections:

- Multi-Statement Transactions
- Optimistic Locking

5.1 Multi-Statement Transactions

The following sections cover how to put multiple MarkLogic Server operations in a single *multi-statement transaction.* Specifically, you *open* a transaction, perform multiple operations in it, and then either *rollback* or *commit* the transaction. This section includes the following parts:

- Transactions and the Java API
- Transaction Class
- Starting A Transaction
- Operations Inside A Transaction
- Rolling Back A Transaction
- Committing A Transaction
- Cookbook: Multistatement Transaction

For detailed information about transactions in MarkLogic Server, see Understanding Transactions in MarkLogic Server in the *Application Developer's Guide*.

5.1.1 Transactions and the Java API

A multi-statement transaction lets you ensure that all operations finish successfully or roll back the transaction such that the system's state is the same as it was before you opened the transaction. For example, you open a transaction, successfully create a document, and then try to perform a metadata operation on a different document that fails. Responding to the failure, you can rollback the document creation.

Since it was in a rolled back transaction, the metadata operation is not the only one that fails; all operations in the transaction are rolled back, resetting the program's and database's states such that the document was never created. However, if you commit the transaction, then the new document will exist.

A key point about MarkLogic Server multi-statement transactions is that rollbacks do not take place automatically on operation failure. It is up to the Java API developer to write code that checks if operations in a transaction succeed or fail, and, in the event of failure, have the rollback method called. Failure could be detected either by tests of your devising, or detecting that an exception has been thrown.

While you can read and search documents during transactions, most transaction operations are writes and deletes, which alter the database's state. A transaction's purpose is to ensure that either all or none of multiple changes to the database are made.

You can also specify a time limit, so if the transaction does not commit before time expires, it is automatically rolled back. However, you should never write code with the expectation that a timeout will roll back a transaction. The time limit serves as a failsafe, not as a programming tool.

Finally, you should be aware that ordinary Java API MarkLogic operations are automatically in a single operation transaction, and whenever an operation touches a document, it locks the document until that operation succeeds or fails. If the MarkLogic detects a deadlock, then the transaction is automatically restarted until either it completes or an exception is thrown (for example, by reaching a time limit for the update transaction). This all happens automatically, and you normally do not need to worry about it. The material in the following sections all deals with multi-statement transactions.

5.1.2 Transaction Class

Transactions are straightforward, once you understand they amount to a wrapper around a series of actions to guarantee either all of those actions are successful, or none of them ever happened. There are only three main operations, opening, committing, and rolling back, each of which will be covered in the next few sections.

`Transaction` is defined in the `com.marklogic.client` package.

- Start a transaction:

  ```
  DatabaseClient.openTransaction()
  ```

- Commit a transaction when it successfully finishes:

  ```
  Transaction.commit()
  ```

- Rollback a multi-statement transaction to reset any actions that have already occured in that transaction; for example, delete any created items, restore any deleted items, revert back any edits, etc.

  ```
  Transaction.rollback()
  ```

Finally, there is the `readStatus()` method, which lets you check if the transaction is still open (in other words, it has been opened, but you have not committed it or rolled it back yet).

5.1.3 Starting A Transaction

To start a transaction and obtain a `Transaction` object, call the `openTransaction()` method on a `DatabaseClient` object (since the transaction controls if database changes are made). To call `openTransaction()`, an application must authenticate as `rest-writer` or `rest-admin`. For example:

```
Transaction transaction = client.openTransaction();
```

You can also include a transaction name, which is rarely used, and time limit arguments. The `timeLimit` value is the number of seconds the transaction has to finish and commit before it is automatically rolled back. As previously noted, you should not depend on the time limit rolling back your transaction; it is only meant as a failsafe to end the transaction if all else fails.

```
Transaction transaction1 = client.openTransaction("MyTrans", 10);
```

5.1.4 Operations Inside A Transaction

Once created and opened, you pass the `Transaction` object to a document manager's `read()`, `write()`, or `delete()` methods, or a query manager's `search()` method to perform operations within the multi-statement transaction. For example:

```
// read a document inside a transaction
docMgr.read(myDocId1, handle, myTransaction);

// write a document inside a transaction
docMgr.write(myDocId1, handle, myTransaction);

// delete a document inside a transaction
docMgr.delete(myDocId2, myTransaction);
```

Of course, you could have several different transactions happening at once, and/or other users could also be running transactions on or sending requests to the same database. To prevent conflicts, whenever the server does something to a document while in a transaction, the database locks the document until that transaction either commits or rolls back. Because of this, you should commit or roll back your transactions as soon as possible to avoid slowing down your and possibly others' applications.

You can intermix commands which are not part of a transaction with transaction commands. Any command without a `Transaction` object argument is not part of a multi-statement transaction. However, you almost always want to group all commands for a given transaction together without interruption so you can commit or roll it back as fast as possible.

5.1.5 Rolling Back A Transaction

In case of an error or exception, call a transaction's `rollback()` method. The `rollback()` method cancels the remainder of the transaction, and reverts the database to its state prior to the transaction's start. With respect to server load, this is better than timing out the transaction. To roll back a transaction, your application must authenticate as `rest-writer` or `rest-admin`. Then just call:

```
transaction.rollback()
```

5.1.6 Committing A Transaction

Once all of a transaction's actions have successfully completed, you need to *commit* the transaction so that the database is actually changed by those actions. To commit, your application must authenticate as `rest-writer` or `rest-admin`. Then just call:

```
transaction.commit();
```

Once a transaction has been committed, it cannot be rolled back and the `transaction` object is no longer available for use. To perform another transaction, you must create a new `transaction` object.

5.1.7 Cookbook: Multistatement Transaction

See `com.marklogic.client.example.cookbook.MultiStatementTransaction` for a full example of how to use transactions. The Cookbook examples are in the Java API distribution in the following directory:

```
example/com/marklogic/client/example/cookbook
```

5.2 Optimistic Locking

An application under *optimistic locking* creates a document only when the document does not exist and updates or deletes a document only when the document has not changed since this application last changed it. However, optimistic locking does not actually involve placing a lock on an object.

Optimistic locking is useful in environments where integrity is important, but contention is rare enough that it is useful to minimize server load by avoiding unnecessary multi-statement transactions.

This section includes the following sub-sections:

- Activating Optimistic Locking
- DocumentDescriptors
- Using Optimistic Locking

5.2.1 Activating Optimistic Locking

Optimistic locking relies on an opaque numeric identifier that is associated with the state of the document's content at a point of time. By default, the REST Server to which the Java API connects does not keep track of this identifier, but you can enable it for use by setting a property, and make it optional or required.

To expand, there is a number associated with every document. Whenever a document's content changes, the value of its number changes. By comparing the stored value of that number at a point in time with the current value, the REST Server can determine if a document's content has changed since the time the stored value was stored.

Note: While this numeric identifier lets you compare state, and uses a numeric value to do so, this is *not* document versioning. The numeric identifier only indicates that a document has been changed, nothing more. It does not store multiple versions of the document, nor does it keep track of what the changes are to a document, only that it has been changed at some point. You cannot use this for change-tracking or archiving previous versions of a document.

Since this App Server configuration parameter applies either to all documents or none, it is implemented in the REST Server. This means it is part of the overall server configuration, and must be turned on and off via a `ServerConfigurationManager` object and thus requires `rest-admin` privileges. For more about server configuration management, see "REST Server Configuration" on page 68.

To activate optimistic locking, do the following:

```
// if not already done, create a database client
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);

// create server configuration manager
ServerConfigurationManager configMgr =
                           client.newServerConfigManager();

// read the server configuration from the database
configMgr.readConfiguration();

// require content versions for updates and deletes
// use Policy.OPTIONAL to allow but not require identifier use
// use Policy.NONE to deactive identifier use
configMgr.setContentVersionRequests(Policy.REQUIRED);

// write the server configuration to the database
configMgr.writeConfiguration();

// release the client
client.release();
```

Allowed values for `ContentVersionRequests` are in the Enum `ServerConfigurationManager.Policy`

5.2.2 DocumentDescriptors

To work with a document's change identifier, you must create a `DocumentDescriptor` for the document. A `DocumentDescriptor` describes exactly one document and is created via use of an appropriately typed method for the document. For more information on document managers, see "Document Managers" on page 10.

```
// create a descriptor for versions of the document
DocumentDescriptor desc = docMgr.newDescriptor(docId);
```

You can also get a document's `DocumentDescriptor` by checking to see if the document exists. This code returns the specified document's `DocumentDesciptor` or, if the document does not exist, `null`:

```
DocumentDescriptor desc = docMgr.exists(docId);
```

5.2.3 Using Optimistic Locking

Each `read()`, `write()`, and `delete()` method for `DocumentManager` has both a version that uses a URI string parameter to identify the document to be read, written, or deleted, and an identical version except you use the `DocumentDescriptor` object instead. The descriptor is only populated with state when you read a document or when you check for a document's existence. When you write, the state changes, but is not reflected in the descriptor.

When `ContentVersionRequests` is required, you must use the `DocumentDescriptor` versions of the `write()` request when modifying the document and `delete()` methods. If the change identifier has not changed, the write or delete operation succeeds. If someone else has changed the document so that a new version has been created, the operation fails by throwing an exception.

> **Note:** Should someone decide to require `ContentVersionRequests`, there is no general notification that this has happened. If someone else is using the same server and, in all innocence, uses the URI string version of `read()`, etc., their applications will now fail and throw exceptions.

If you are creating a document under required use, you either must not supply a descriptor, or if you do pass in a descriptor it must not have state. A descriptor is stateless if it is created through a DocumentManager and has not yet been populated with state by a `read()` or `exists()` method. If the document does not exist, the operation succeeds. If the document exists, the operation fails and throws an exception.

If optimistic locking is optional, if you do not supply an identifier value via the descriptor and use the `docId` versions of `write()` and `delete()`, the operation always succeeds. If you do supply an identifier value by using the `DocumentDescriptor` versions of `write()` and `delete()`, the same rules apply as above when the version flag is mandatory.

The identifier value always changes on the server when a document's content changes there.

The "optimistic" part of optimistic locking comes from this not being an actual lock, but rather a means of checking if another application has changed a document since you last accessed it. If another application does try to modify the document, the Server does not even try to stop it from doing so. It just changes the document's identifier value.

So, the next time your application accesses the document, it compares the number it stored for that document with its current number. If they are different, your application knows the document has been changed since it last accessed the document. It could have been changed once, twice, a hundred times; it does not matter. All that matters is that it has been changed. If the numbers are the same, the document has not been changed since you last accessed it.

5.2.4 Cookbook: Version Control and Optimistic Locking

See `com.marklogic.client.example.OptimisticLocking` in the Cookbook for a full example of how to use and optimistic locking. The Cookbook examples are in the Java API distribution in the following directory:

```
example/com/marklogic/client/example/cookbook
```

6.0 Logging

`RequestLogger` objects are supplied to individual manager objects, most commonly document and query managers. You can choose to log content sent to the server as well as any requests. It is located in `com.marklogic.client.util`.

This chapter includes the following sections:

- Starting Logging
- Suspending and Resuming Logging
- Stopping Logging
- Log Entry Format
- Logging To The Server's Error Log

6.1 Starting Logging

First, you must obtain a `RequestLogger` object via `DatabaseClient`'s `newLogger()` method, which takes an argument of an output stream to send the log messages to. This output stream can be shared with other loggers outside of the MarkLogic Server Java API. You are responsible for flushing the output stream.

```
out = new ByteArrayOutputStream();
RequestLogger logger = client.newLogger(out);
```

To start logging, call the `startLogging()` method on a manager object with an argument of a `RequestLogger` object. For example:

```
MyDocumentManager.startLogging(logger)
```

There is only one logger for any given object. However, you can share a `RequestLogger` object among multiple manager objects, just by specifying the same `RequestLogger` object in multiple `startLogging()` method calls.

6.2 Suspending and Resuming Logging

By using `RequestLogger`'s `setEnabled()` method, you can pause and resume logging on any logger object. For example, to suspend logging:

```
logger.setEnabled(false)
```

To reenable logging:

```
logger.setEnabled(true)
```

To check if logging is enabled or not:

```
logger.isEnabled(); //returns a boolean
```

When you change a logger's enable status, it applies to all manager objects for which that `RequestLogger` object was used as an argument to `StartLogging()`.

6.3 Stopping Logging

To stop logging on a manager, call the `stopLogging()` method. If called on a manager not currently logging, nothing happens, not even an error or exception. The `RequestLogger` object associated with the manager is not destroyed by this method and you can reuse and restart it.

```
MyDocumentManager.stopLogging()
```

6.4 Log Entry Format

Two types of things can be logged once logging is turned on and enabled. Requests to the server are always logged. These include search requests, configuration requests, and all database requests. By default, only requests are logged.

You can use `RequestLogger`'s `setContentMax()` method to control how much content is logged. By giving it the constant `ALL_CONTENT` value, all content is logged. To revert to no content being logged, use the constant `NO_CONTENT`. If you use a numeric value, such as `1000`, the first that many content bytes are logged. Note that if the request is for a deletion, no content is logged.

`FileHandle` is an exception to the ability to log content. Only the name of the file is logged.

You can also retrieve a request logger's underlying print stream by calling `getPrintStream()` on the `RequestLogger` object. Once you access the log's print stream, writing to it adds your own messages to the log.

6.5 Logging To The Server's Error Log

You can also use `ServerConfigurationManager.setServerRequestLogging()` to turn logging requests to the server's error log on or off, based on the boolean argument you provide. This log's location is platform dependent. For details about log files in MarkLogic Server, see Log Files in the *Administrator's Guide*.

7.0 REST Server Configuration

REST Server configuration is done through a `ServerConfigurationManager` object located in package `com.marklogic.client.admin`. REST Server configuration deals with the underlying REST instance running in MarkLogic. You can configure REST Server properties, namespace bindings, query options, and transform and resource extensions.

Note that you can only configure aspects of the underlying REST instance with the Java API. MarkLogic Server administration is not exposed in Java, so things such as creating indexes, creating users, creating databases, assigning roles to users, and so on must be done via the MarkLogic Admin Interface or other means (for example the Admin API or Information Studio). For more information about administering MarkLogic Server, see the *Administrator's Guide*.

This chapter includes the following sections:

- Creating a Server Configuration Manager Object
- Reading and Writing Server Configuration Properties
- REST Server Properties
- Creating New Server-Related Manager Objects
- Namespaces
- Logging Namespace Operations

7.1 Creating a Server Configuration Manager Object

Using a `com.marklogic.client.DatabaseClient` object, call `newServerConfigManager()`

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);

// create a manager for server configuration
ServerConfigurationManager configMgr =
                           client.newServerConfigManager();
```

Your application should only need one active `ServerConfigurationManager` at any time.

7.2 Reading and Writing Server Configuration Properties

TUse `com.marklogic.client.admin.ServerConfigurationManager` to manage server configuration properties. To read the current server configuration values into the `ServerConfigurationManager` object, do:

```
configMgr.readConfiguration();
```

If your application changes these values, they will not persist unless written out to the server. To write the REST Server Configuration values to the server, do:

```
configMgr.writeConfiguration();
```

7.3 REST Server Properties

`com.marklogic.client.admin.ServerConfigurationManager` objects have `get` and `set` methods for the following server properties:

- `ContentVersionRequests`: Value from the `ServerConfigurationManager.Policy` enum specifying whether the system tries to detect if a document is "fresh" or not via use of an opaque numeric identifier. For more information, see "Optimistic Locking" on page 62.
- `DefaultDocumentReadTransform`: Name of the default transform applied to documents as they are read from the server. For information about document transforms, see "Content Transformations" on page 73.
- `QueryOptionsValidation`: Boolean specifying whether the server validates query options before storing them in configurations. For information about query options, see "Query Options" on page 46.
- `ServerRequestLogging`: Boolean specifying whether the REST Server logs requests to the MarkLogic Server error log (`ErrorLog.txt`). For performance reasons, you should only enable this when debugging your application. For information about logging, see "Logging" on page 66.

7.4 Creating New Server-Related Manager Objects

Most manager objects described so far handle access to the database and its content, and accordingly are created via a method on a `DatabaseClient` object. The following managers handle listing, reading, writing, and deleting REST Server data and settings, rather than those of the database. Therefore, these managers are created by factory methods on a `ServerConfigurationManager` instead of a `DatabaseClient`.

The `ServerConfigurationManager` associated managers are:

- `NamespaceManager`: Namespace bindings. For details about namespaces, see "Namespaces" on page 69.
- `QueryOptionsManager`: Query options. For details, about query options, see "Query Options" on page 46.
- `ResourceExtensionsManager`: Resource service extensions. For details about resource service extensions, see "Resource Service Extensions" on page 78.
- `TransformExtensionManager`: Transform extensions. For details, about transform extensions, see "Content Transformations" on page 73.

7.5 Namespaces

Namespaces are similar to Java packages in that they differentiate between potentially ambiguous XML elements. With the Java API, you can define namespace bindings on the REST Server.

In XML and XQuery, element and attribute nodes are always in a namespace, even if it is the empty namespace (sometimes called no namespace) which has the name of the empty string (""). Each non-empty namespace has an associated URI, which is essentially a unique string that identifies the namespace. That string can be bound to a namespace prefix, which is a shorthand string used as an alias for the namespace in path expressions, element qnames, and variable declarations. Namespace operations in the Java Client API are used to define namespace prefixes on the REST Server so the client and server can share identical namespace bindings on XML elements and attributes for use in queries.

Specifically, for key-value searchs, if an element argument is in a namespace, you need to know which namespace it is in. This is also true for the attribute value, but is less commonly used. For example, if you have element as follows:

```
<f:foo xmlns:f="example.com">---</f:foo>
```

Now if you want to be able to pass `/f:foo`. So you want to configure the server so that the prefix `f` binds to namespace URI `example.com`.

Another usage example is in query configuration; when you are setting up a range index configuration, or specifying an XML element or element attribute, you need identical prefix bindings to the server in order to correctly pass the arguments to the server.

Note that a namespace URI can be bound to multiple prefixes, but a prefix can only be bound to one URI.

For more information about namespaces, see Understanding XML Namespaces in XQuery in the *XQuery and XSLT Reference Guide*, which provides a detailed description of XML namespaces and their use. This section includes the following parts:

- Namespaces Manager
- Getting Server Defined Namespaces
- Adding And Updating A Namespace Prefix
- Reading Prefixes
- Deleting Prefixes

7.5.1 Namespaces Manager

The `com.marklogic.admin.NamespacesManager` class provides editing for namespaces defined on the REST Server. To use `NamespacesManager`, the application must authenticate as `rest-admin`. Since namespaces are based on the REST Server, a new `NamespacesManager` is defined via `com.marklogic.client.admin.ServerConfigManager`.

```
NamespacesManager nsManager =
    client.newServerConfigManager().newNamespacesManager();
```

7.5.2 Getting Server Defined Namespaces

Use `com.marklogic.client.admin.NamespacesManager` to get all of the namespaces defined on the REST Server. For example:

```
nsManager.readAll();
```

This returns a `javax.xml.namespace.NamespaceContext` interface that includes all of the REST Server defined namespaces. You can run the following on the `NamespaceContext` object.

```
nsContext.getNamespaceURI(prefix-string);
nsContext.getPrefix(URI-string);
nsContext.getPrefixes(URI-string);
```

`getNamespaceURI()` returns the URI associated with the given prefix. `getPrefix()` returns one of the prefixes associated with the given URI. `getPrefixes()` returns an iterator of all the prefixes associated with the given URI.

In addition, by casting the `NamespaceContext` to `EditableNamespaceContext`, you can iterate over the complete set of prefixes and URIs:

```
EditableNamespaceContext c =(EditableNamespaceContext)nsMgr.readAll();
for (Entry e:c.entrySet()){
  prefix = e.getKey();
  nsURI = e.getValue();
...
}
```

7.5.3 Adding And Updating A Namespace Prefix

Use `com.marklogic.client.admin.NamespacesManager` to add a new namespace prefix. For example:

```
nsManager.addPrefix("ml", "http://marklogic.com/exercises");
```

The first argument is the prefix, and the second argument is the URI being associated with the prefix.

To update the value of an existing prefix, do the following:

```
nsManager.updatePrefix("ml", "http://marklogic.com/new_exercises");
```

Where the first argument is the prefix, and the second argument is the new URI bound to it.

7.5.4 Reading Prefixes

Use `com.marklogic.client.admin.NamespacesManager` to read, or get, the associated URI value, of a single prefix. For example:

```
nsManager.readPrefix("ml");
```

It returns the prefix's associated URI as a string.

In order to read, or get, all of the prefixes associated with a Namespace Manager, do the following:

```
NamespaceContext context = nsManager.readAll();
```

`NamespaceContext` is a standard `javax.xml` Interface for storing a set of namespace declarations on the client. With a `NamespaceContext` object, you can:

- Get the prefix for any URI for which a prefix-URI binding has been created in this `NamespaceServer`. The below would return its prefix, say, `"ml"`.

  ```
  context.getPrefix("http://marklogic.com/new_exercises");
  ```

- Get the URI for any prefix for which a prefix-URI binding has been created in this `NamespaceServer`. The below returns the URI `"http://marklogic.com/new_exercises"`

  ```
  context.getNamespaceURI("ml");
  ```

- Get all of the prefixes for any URI for which prefix-URI bindings have been created in this `NamespaceServer`. The below returns all the associated prefixes in an Iterator.

  ```
  context.getPrefixes(“http://marklogic.com/new_exercises);
  ```

7.5.5 Deleting Prefixes

To delete a single prefix from the namespaces manager, do:

```
nsManager.deletePrefix("ml");
```

To delete all of the prefixes defined under a `NamespaceManager`, do:

```
nsManager.deleteAll();
```

7.6 Logging Namespace Operations

As with all manager objects, you can start and stop logging operations on a `NamespacesManager` via the `startLogging()` and `stopLogging()` methods. For details on how to use the logging facility, see “Logging” on page 66.

8.0 Content Transformations

The MarkLogic REST API allows you to create custom content transformations and apply them during operations such as document ingestion and retrieval. Transforms can be implemented as XQuery library modules or XSLT stylesheets, and can accept transform-specific parameters. Transforms can be applied on document reads and writes.

`ServerConfigurationManager` has the property `DefaultDocumentReadTransform` which can be set to the name of a content transformation. This transformation then will be automatically applied to every document when they are read from the database. By default, there is no default read transform; an admin has to set `DefaultDocumentReadTransform` to one.

As always, you must be `rest-admin` to do `ServerConfigurationManager` operations.

This chapter contains the following sections:

- Installing Transforms
- Using Transforms
- Writing Transformations

8.1 Installing Transforms

To install a transform on your server, do the following steps:

1. Create a `DatabaseClient` for connecting to the database.

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);
```

2. Create a manager for transform extensions. Since extensions are a configuration of the REST Server, use a `ServerConfigManager` to create a `TransformExtensionsManager`.

```
TransformExtensionsManager transMgr =
    client.newServerConfigManager().newTransformExtensionsManager();
```

3. Specify the new transform extension's metadata, using an `ExtensionMetadata` object to contain it.

```
ExtensionMetadata metadata = new ExtensionMetadata();
metadata.setTitle("XML-TO-HTML XSLT Transform");
metadata.setDescription("This plugin transforms an XML document with a
                         known vocabulary to HTML");
metadata.setProvider("MarkLogic");
metadata.setVersion("0.1");
```

4. Get the transform's source code.

```
FileInputStream transStream = new FileInputStream(
              "scripts"+File.separator+TRANSFORM_NAME+".xsl");
```

5. Wrap the source code in a text handle.

```
InputStreamHandle handle = new InputStreamHandle(transStream);
```

6. Write the transform, its source code, and its metadata to the REST server.

```
transMgr.writeXSLTransform(TRANSFORM_NAME, handle, metadata);
```

7. Release the client.

```
client.release();
```

8.2 Using Transforms

Once you have installed a transform in the database, it is simple to use it. You can apply a transform to a document either when reading the document from the database, or when writing the document to the database. This section describes how to use transforms and includes the following parts:

- Transforming a Document When Reading It
- Transforming a Document When Writing It
- Overall Transform Administration
- Reading Transforms
- Logging

8.2.1 Transforming a Document When Reading It

To transform a document when reading it, do the following:

1. Create a `DatabaseClient` for connecting to the database.

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);
docId = "/examples/mydoc.xml";
```

2. Create an appropriate Document Manager for the to be transformed document. In this case, we use a `TextDocumentManager`.

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
```

3. Create a handle for the document's content.

```
StringHandle readHandle = new StringHandle();
```

4. Optionally, specify the content's MIME type for the content. This is only needed if the transform supports content negotiation and changes the content from one MIME type to another.

```
readHandle.setMimetype("text/xml");
```

5. Specify the transform by declaring a new `ServerTransform` object using the transform's name as the argument.

```
ServerTransform transform = new ServerTransform(TRANSFORM_NAME);
// you can specify any parameters here too
```

6. Read the XML content file in, specifying that as it is read in, a transform is to be applied to it. The result is that the handle argument will contain the transformed content.

```
docMgr.read(docId, readHandle, transform);
```

7. Close the database client.

```
client.release();
```

8.2.2 Transforming a Document When Writing It

To transform a document when writing it, do the following:

1. Create a `DatabaseClient` for connecting to the database and assign the name of the document to be transformed to a variable.

```
DatabaseClient client = DatabaseClientFactory.newClient(
   host, port, user, password, authType);
docId = "/examples/mydoc.xml";
```

2. Create an appropriate Document Manager for the document. In this case, we use a `TextDocumentManager`.

```
TextDocumentManager writeMgr = client.newTextDocumentManager();
```

3. Specify a document on the client.

```
FileInputStream docStream = docId);
```

4. Create a handle for the document's content.

```
InputStreamHandle writeHandle = new InputStreamHandle(docStream);
```

5. Optionally, specify the content's MIME type for the content. This is only needed if the transform supports content negotiation and changes the content from one MIME type to another.

```
writeHandle.setMimetype("text/xml");
```

6. Specify the transform by declaring a new `ServerTransform` object using the transform's name as the argument and then specify any parameter values.

```
ServerTransform transform = new ServerTransform(TRANSFORM_NAME);
transform.put("drop-font-tags", "yes");
```

7. Write out the content to the database, so that the transform stored there changes it.

```
writeMgr.write(docId, writeHandle, transform);
```

8. Close the database client.

```
client.release();
```

8.2.3 Overall Transform Administration

You can list all currently installed transform extensions by doing the following:

```
String result = transMgr.listTransforms(
   new StringHandle().withFormat(Format.XML)).get();
// format can be JSON as well
```

To delete a transform, effectively uninstalling it from the server do the following:

```
TransMgr.deleteTransform(TRANSFORM_NAME);
```

8.2.4 Reading Transforms

To read the source code of an XQuery implemented transform into your application, do:

```
StringHandle textHandle = readXQueryTransform(TRANSFORM_NAME,
   new StringHandle()); // can be any text handle
```

To read the source code of an XSLT implemented transform into your application, do:

```
XMLReadHandle xHandle = readXSLTransform(TRANSFORM_NAME,
   new XMLReadhandle());
```

8.2.5 Logging

Since it is a manager, you can define a `RequestLogger` object and start and stop logging client requests to the `TransformExtensionsManager`. For more information, see "Logging" on page 66

```
RequestLogger logger = client.newLogger(stream);
transformsMgr.startLogging(logger);
transformsMgr.stopLogging();
```

8.3 Writing Transformations

You can create transformation using the REST API and then access them from the Java API. For details on writing transformations, see Writing Transformations in the *REST Application Developer's Guide*.

9.0 Resource Service Extensions

This chapter describes how to extend the Java API functionality with resource service extensions, and contains the following sections:

- Extending the Java API
- Creating a Resource Extension in XQuery
- Installing Resource Extensions
- Deleting Resource Extensions
- Listing Resource Extensions
- Using Resource Extensions

9.1 Extending the Java API

Resource service extensions extend the MarkLogic Java API by making XQuery library modules available for use within Java. A resource extension implements services for a server-side resource. For example, you can create a dictionary program resource extension that looks up words, checks spelling, and makes suggestions for not found words. The individual operations an application programmer may call, for example, `lookUpWords()`, `spellCheck()`, and so on, are the services that make up the resource extension. The following are the basic steps to making a resource extension available to the Java API:

1. Create an XQuery library module that implements the services for the resource.

2. Install the XQuery library module as an extension on the REST Server using `com.marklogic.client.admin.ResourceExtensionsManager`.

3. Make your resource extension available to Java applications by creating a wrapper class that is a subclass of `com.marklogic.client.extensions.ResourceManager`. Inside this class, access the resource extension methods using a `com.marklogic.client.extensions.ResourceServices` object obtained through the `ResourceManager.getServices()` method.

4. Use the methods of your `ResourceManager` subclass to access the services provided by the extension from the rest of your application.

The key objects for resource extentions in the Java API are:

- `ResourceExtensionsManager`, which manages creation, modification, and deletion of XQuery implementations of resource services on the REST Server.
- `ResourceManager`, the base class for classes that you write to provide client interfaces to resource services.
- `ResourceServices`, which supports calling the services for a resource. The resource services extension XQuery module must already be installed on the server via the `ResourceExtensionsManager` before `ResourceServices` can access it.

These objects are created via a `ServerConfigManager`, since resource services are associated with the server, not the database. You must connect to MarkLogic as a user with the `rest-admin` role to create and work with `ResourceExtensionsManager`.

For a complete example of implementing and using a resource service extension, see `com.marklogic.client.example.cookbook.ResourceExtension` in the `example/` directory of your Java API installation.

9.2 Creating a Resource Extension in XQuery

The Java API uses the same XQuery resource extensions as the REST API, and the guidelines for writing extensions are identical for use in the Java API as in the REST API. For details on extending the REST API with a XQuery resource extension, see Extending the REST API in the *REST Application Developer's Guide*.

9.3 Installing Resource Extensions

Before you can use a resource extension, you must install the XQuery implementation on MarkLogic Server as follows:

1. If your resource extension XQuery library module depends on additional library modules, install these dependent libraries on MarkLogic Server. For details, see Installing Dependent XQuery Library Modules in the *REST Application Developer's Guide*.

2. If you have not already done so, connect to the database, storing the connection in a `com.marklogic.client.DatabaseClient` object.

```
DatabaseClient client = DatabaseClientFactory.newClient(
     host, port, user, password, authType);
```

3. If you have not already done so, create a `com.marklogic.client.admin.ResourceExtensionsManager`. Note that the method for doing so is associated with a `ServerConfigManager`.

```
ResourceExtensionsManager resourceMgr =
    client.newServerConfigManager().newResourceExtensionsManager();
```

4. Define a `com.marklogic.client.admin.ExtensionMetadata` object and populate it with your resource extension's metadata. You can set title, description, provider name, and version.

```
ExtensionMetadata metadata = new ExtensionMetadata();
metadata.setTitle("Spelling Dictionary Resource Services");
metadata.setDescription("This plugin supports spelling dictionaries");
metadata.setProvider("MarkLogic");
metadata.setVersion("0.1");
```

5. Create a text handle (such as an input stream and a handle associated with it) to receive the extension's source code. Use the `set()` method on the handle to associate it with the desired stream, as in the following example.

```
FileInputStream myStream = new FileInputStream("sourcefile.xqy");
InputStreamHandle handle = new InputStreamHandle(myStream);
handle.set (myStream);
```

6. Install the extension by calling a `writeServices()` method on the `ResourceExtensionsManager`, with arguments of the extension's name, the handle containing the source code, the `ExtensionMetadata` object containing the metadata, and a declaration of the parameters for the services for one or more of the four methods, in this case `GET`.

```
resourceMgr.writeServices(DictionaryManager.NAME, handle, metadata,
                          new MethodParameters(MethodType.GET));
```

7. When finished with the database, release the resources for the database connection by calling the `release()` method on the `DatabaseClient` object.

```
client.release();
```

The following code sample demonstrates the above steps. For a complete example, see `com.marklogic.client.example.cookbook.ResourceExtension` in the `example/` directory of your Java API distribution.

```
// create a manager for resource extensions
ResourceExtensionsManager resourceMgr =
  client.newServerConfigManager().newResourceExtensionsManager();

// specify metadata about the resource extension
ExtensionMetadata metadata = new ExtensionMetadata();
metadata.setTitle("Spelling Dictionary Resource Services");
metadata.setDescription("This plugin supports spelling dictionaries");
metadata.setProvider("MarkLogic");
metadata.setVersion("0.1");

// acquire the resource extension source code
InputStream sourceStream = new FileInputStream("dictionary.xqy");

// create a handle on the extension source code
```

```
InputStreamHandle handle = new InputStreamHandle();
handle.set(sourceStream);

// write the resource extension to the database
resourceMgr.writeServices(DictionaryManager.NAME, handle, metadata,
new MethodParameters(MethodType.GET));
```

9.4 Deleting Resource Extensions

To delete a resource extension, call the `deleteServices()` method of `com.marklogic.client.admin.ResourceExtensionManager`. For example, assuming you have already obtained a `ResourceExtensionsManager` object, do the following:

```
resourceMgr.deleteServices(resourceName);
```

9.5 Listing Resource Extensions

To list all the installed extensions, use a handle as in the following example, which gets the extensions list in XML format:

```
String result = resourceMgr.listServices(
   new StringHandle().withFormat(Format.XML)).get();
```

9.6 Using Resource Extensions

After you install the extension as described in “Installing Resource Extensions” on page 79, create a wrapper class that exposes the functionality of the extension to your application. The wrapper class should be a subclass of `com.marklogic.client.extensions.ResourceManager`. In the implementation of your wrapper class, use `com.marklogic.client.extensions.ResourceServices` to invoke the GET, PUT, POST and/or DELETE methods of the resource extension.

Use these guidelines in implementing your wrapper subclass:

1. Before using any services, initialize your `ResourceManager` subclass by passing it to `com.marklogic.client.DatabaseClient.init()`. For example:

```
public class DictionaryManager extends ResourceManager {
    static final public String NAME = "dictionary";
    ...

    public DictionaryManager(DatabaseClient client) {
         super();

        // Initialize the Resource Manager via the Database Client
       client.init(NAME, this);
    }
    ...
}
```

2. To pass parameters to a resource extension method, create a `com.marklogic.client.util.RequestParameters` object and add parameters to it. Each parameter is represented by a parameter name and value. Use the parameter names defined by the resource extension. For example:

```
//Build up the set of parameters for the service call
RequestParameters params = new RequestParameters();
params.add("service", "dictionary");
params.add("uris", uris);
```

3. Obtain a `com.marklogic.com.extensions.ResourceServices` object through the inherited protected method `getServices()`. For example:

```
public class DictionaryManager extends ResourceManager {
    ...
    public Document[] checkDictionaries(String. . . uris) {

        ...
        // get the initialized service object from the base class
        ResourceServices services = getServices();
        ...
    }
}
```

4. Use the `get()`, `put()`, `post()`, and `delete()` methods of `ResourceServices` to invoke methods of the resource extension on the server. For example:

```
ResourceServices services = getServices();
ServiceResultIterator resultItr = services.get(params, mimetypes);
```

The results from calling a resource extension method are returned as either a `com.marklogic.client.extensions.ResourceServices.ServiceResultIterator` or a handle on the appropriate content type. Use a `ServiceResultIterator` when a method can return multiple items; use a handle when it returns only one. Resources associated with the results are not released until the associated handle is discarded or the iterator is closed or discarded.

The code below combines all the guidelines together in a sample application that exposes dictionary operations. For the complete example, see the Cookbook example `com.marklogic.client.example.cookbook.ResourceExtension` in the `example/` directory of your Java API distribution.

```
public class DictionaryManager extends ResourceManager {
    static final public String NAME = "dictionary";
    private XMLDocumentManager docMgr;

    public DictionaryManager(DatabaseClient client) {
         super();

        // Initialize the Resource Manager via the Database Client
       client.init(NAME, this);
```

```
    }

    // Our first Java implementation of a specific service from
    // the extension
    public Document[] checkDictionaries(String. . . uris) {
        //Build up the set of parameters for the service call
        RequestParameters params = new RequestParameters();
        // Add the dictionary service parameter
        params.add("service", "dictionary");
        params.add("uris", uris);

        String[] mimetypes = new String[uris.length];
        for (int i=0; i < uris.length; i++) {
            mimetypes[i] = "application/xml";
        }

        // get the initialized service object from the base class
        ResourceServices services = getServices();

        // call the service implementation on the REST Server,
        // returning a ResourceServices object
        Service Result Iterator resultItr =
            services.get(params, mimetypes);

        //iterate over results, get content
        ...
        // release resources
        resultItr.close();

    }
    ...
}
```

10.0 Troubleshooting

This chapter describes how to troubleshoot errors while programming in the Java API, and contains the following sections:

- Error Detection
- General Troubleshooting Techniques

10.1 Error Detection

As you would expect, the Java API client indicates errors by throwing exceptions. It does not return errors or otherwise indicate problems by any other means. The exceptions are located in `com.marklogic.client` and are:

- `FailedRequestException`: Indicates a problem at the REST API level.
- `ForbiddenUserException`: Indicates credentials used to connect to a REST instance are not sufficient for the requested task. Equivalent to a `403` HTTP status code.
- `MarkLogicBindingException`: Indicates a problem binding a value.
- `MarkLogicInternalException`: Indicates a defect in the API. Call MarkLogic Support.
- `MarkLogicIOException`: `RuntimeException` Thrown when a code block internally throws `java.lang.IOException`.
- `MarkLogicServerException`: The MarkLogic REST Server threw an exception.
- `ResourceNotFoundException`: Thrown when the server responds with an HTTP `404` status.
- `UnauthorizedUserException`: Thrown when a user attempts an operation to which they do not have the rights for.

10.2 General Troubleshooting Techniques

The following are some general guidelines for troubleshooting your program.

- To troubleshoot unexpected search results, pass the query option for `debug`, which returns errors in the query options, and the `return-qtext` option, which returns the pre-parsed query text for the search.
- Remember that documents with no read permission are hidden.
- To troubleshoot exceptions, pay close attention to any messages returned from the server.
- Set the MarkLogic Server error log to `debug` and view the server log (`<marklogic-dir>/Logs/ErrorLog.txt`) for more details.
- To monitor the HTTP requests against the REST Server, look at the access logs under the `<marklogic-dir>/Logs` directory for your REST App Server (for example, `1234_AccessLog.txt` for the server running on port 1234).
- Configure managers with a request logger to confirm requests are correct.

- To troubleshoot extensions, first execute the XQuery code in an XQuery environment. Then look at the requests and server log.
- Check the query options builder output to make sure it is what you expect, either with `QueryOptionsHandle.toString()`, which outputs the XML representation of the query options, or by checking the stored options against what is expected. Errors reported by MarkLogic Server refer to the structure of this document.
- When you have a mismatch between query options and existing indexes, you can look at the `/v1/config/indexes?format=html` endpoint on your REST Server.
- If you want a closer look at the requests against the REST Server, use a network sniffer to watch the HTTP trafic against the REST Server. You can also try to execute an equivalent request for the REST API using cURL or some other HTTP client.

11.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement. For evaluation licenses, MarkLogic may provide support on an “as possible” basis.

For customers with a support contract, we invite you to visit our support website at http://support.marklogic.com to access information on known and fixed issues.

For complete product documentation, the latest product release downloads, and other useful information for developers, visit our developer site at http://developer.marklogic.com.

If you have questions or comments, you may contact MarkLogic Technical Support at the following email address:

support@marklogic.com

If reporting a query evaluation problem, please be sure to include the sample XQuery code.

Combined Product Notices

MarkLogic Server 6.0 and supporting products.
Last updated: August 19, 2013

COPYRIGHT

Copyright © 2013 MarkLogic Corporation. All rights reserved.
This technology is protected by U.S. Patent No. 7,127,469B2, U.S. Patent No. 7,171,404B2, U.S. Patent No. 7,756,858 B2, and U.S. Patent No 7,962,474 B2.

The MarkLogic software is protected by United States and international copyright laws, and incorporates certain third party libraries and components which are subject to the attributions, terms, conditions and disclaimers set forth below.

1. Contains SAP BusinessObjects Text Analysis XI from SAP AG. Copyright © 1996-2011. All rights reserved.

2. Highslide Software from Highslide Software Torstein Honsi. All Highslide Software is protected by local and international copyright laws. All rights reserved.

3. Icons developed by Yusuke Kamiyamane. Copyright © 2011 Yusuke Kamiyamane. All rights reserved. Icons are licensed subject to http://creativecommons.org/licenses/by/3.0/legalcode.

4. Antenna House OfficeHTML Copyright © 2000-2008 Antenna House, Inc. All rights reserved.

5. Argus Copyright ©1999-2008 Iceni Technology Ltd. All rights reserved.

6. Rosette Linguistics Platform 6.5.2/6.5.3 from Basis Technology Corporation, Copyright © 2004-2008 Basis Technology Corporation. All rights reserved.

7. ISYS Search. Copyright © 2011 ISYS™ Search Software, Inc. All rights reserved.

8. Software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved. Copyright © 1998-2011 The OpenSSL Project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)" 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. 5. Products derived from this

software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. 6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)" THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Additional terms may apply to the foregoing software as further set forth at http://www.openssl.org/source/license.html

9. ICU v. 4.2.1 available from http://site.icu-project.org// Copyright © 1995-2011 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

10. Tidy available at http://www.w3.org/People/Raggett/tidy/. Copyright © 1998-2008 World Wide Web Consortium (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

11. 'zlib' general purpose compression library v. 1.2.3. Copyright © 1995-2010 Jean-loup Gailly and Mark Adler. http://www.zlib.net/zlib_license.html

12. RSA Data Security, Inc. MD5 Message-Digest Algorithm, copyright © 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function. License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work. RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind. These notices must be retained in any copies of any part of this documentation and/or software.

13. TRE software, available at http://laurikari.net/tre/. Copyright © 2001-2009 Ville Laurikari <vl@iki.fi>. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: (1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

14. Eigen Library, version 2.0.10, available at http://eigen.tuxfamily.org/index.php?title=Main_Page. Eigen Library is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. Eigen Library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with Eigen Library. If not, see: http://www.gnu.org/copyleft/lesser.html

15. FunctX XQuery Library v.1.0 available at http://www.xqueryfunctions.com/xq/, Copyright © Datypic (Priscilla Walmsley). You can redistribute such library and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the

hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License as available at http://www.gnu.org/licenses/lgpl-2.1.html#SEC4 for more details.

16. DeployJava.js v. 1.8, Copyright © 2006, 2011, Oracle and/or its affiliates. All rights reserved. ORACLE PROPRIETARY/CONFIDENTIAL. Use of the foregoing library is subject to the following license terms. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Oracle nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

17. YUI libraries v.2.8 available at http://developer.yahoo.com/yui/. Copyright © 2011 Yahoo! Inc. All rights reserved. Use of the foregoing libraries is subject to the following license terms. Redistribution and use of this software in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Yahoo! Inc. nor the names of YUI's contributors may be used to endorse or promote products derived from this software without specific prior written permission of Yahoo! Inc. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Additional notices related to components included in the foregoing library are also available at the following URL: http://yuilibrary.com/license/

18. XSLTForms libraries available at http://sourceforge.net/projects/xsltforms/ . You can redistribute such libraries and/or modify them under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. These libraries are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License as available at http://www.gnu.org/licenses/lgpl-2.1.html#SEC4 for more details.

19. jQuery v.1.7 libraries available at http://jquery.com/ and jQuery UI v.1.8.2 libraries available at http://jqueryui.com/. Copyright © 2011 John Resig. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OFMERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE ANDNONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BELIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTIONOF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTIONWITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

20. JSON v. 2 libraries available at http://www.json.org/json2.js. Copyright © 2002 JSON.org. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. The Software shall be used for Good, not Evil. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

21. CodeMirror libraries available at http://codemirror.net/. Copyright © 2011 by Marijn Haverbeke marijnh@gmail.com. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

22. History.js available https://github.com/balupton/History.js/ Copyright © 2011, Benjamin Arthur Lupton. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Benjamin Arthur Lupton nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

23. Sample content provided for use with the software is copyrighted by the respective authors as contributed by each of them to Wikipedia pursuant to the Creative Commons Attribution-ShareAlike License found at http://creativecommons.org/licenses/by-sa/3.0/.

24. Visual Studio Tools for Office 3.0 (Microsoft VSTO 3.0) licensed under Microsoft Public License Microsoft Public License (Ms-PL) found at http://msdn.microsoft.com/en-us/library/ff647676.aspx.

25. The MarkLogic Connector for Hadoop contains Apache Jakarta Commons Modeler. Copyright 2001-2007 The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

26. jemalloc http://www.canonware.com/jemalloc/index.html, released under the terms of the following BSD-derived licenses: Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Portions of the software are copyright (C) 2002-2013 Jason Evans, and portions copyright (C) 2009-2013 Facebook, Inc. and copyright (C) 2007-2012 Mozilla Foundation. Neither the name of Facebook, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. All rights reserved.

27. Gziphelper; version: 1.0, Feb, 2003, authored by Gao Dasheng, is distributed subject to the following terms: This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: (1.) the origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required; (2) altered source versions must be plainly marked as such, and must not be misrepresented as being the original software; (3) this notice may not be removed or altered from any source distribution.

28. Some features of MarkLogic allow you to develop and use Google Maps. Review Google Map's privacy policy at http://www.google.com/policies/privacy/, legal notices at http://maps.google.com/help/legalnotices_maps.html, and terms of use at http://maps.google.com/help/terms_maps.html.

29. commons-codec, HttpComponents, commons-logging, and Google Maps Utility are used under the Licensed under the Apache License, Version 2.0 (the "License"). You may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

30. Mapstraction. Copyright (c) 2012, Mapstraction. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

31. logback-classic and logback-core are provided in object code form only under the Eclipse Public License - v 1.0 available at http://www.eclipse.org/legal/epl-v10.html. Copyright (C) 1999-2012, QOS.ch. All rights reserved. Source code for the Program is available http://logback.qos.ch/.

32. Simple Logging Facade for Java (SLF4J). Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

33. cURL v. 7.22.0 available at http://curl.haxx.se/. Copyright © 1996 - 2011, Daniel Stenberg, <daniel@haxx.se>. All rights reserved. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

34. heatmap.js distributed subject to the "THE BEER-WARE LICENSE" (Revision 42): <phk@FreeBSD.ORG> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return Poul-Henning Kamp.

35. Except for components that are otherwise subject to other open source licenses, the MarkLogic ODBC Driver is released under the GNU Lesser General Public License as published by the Free Software Foundation, version 2.1 of the License.

36. Solr-commons-csv-3.5.0 Copyright 2005-2006 The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

37. Avro v. 1.7.1. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

38. Guava v. 11.0.2. Copyright © Google. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

39. Hadoop-auth v. 2.0.0. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

40. hadoop-hdfs v. 2.0. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

41. commons-lang v. 2.4, 2.5. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

42. commons-configuration v. 2.0. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

43. jackson-core-asl v. 1.8.8. Copyright © FasterXML. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

44. jackson-mapper-asl v. 1.8.8. Copyright © FasterXML. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

45. log4j v. 1.2.16. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

46. protobuf-java v. 2.4.0a. Copyright © Google. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Google nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

47. xpp3 v. 1.1.3.3. Copyright © 2002 Extreme! Lab, Indiana University. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab(http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. The names "Indiana Univeristy" and "Indiana Univeristy Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. Products derived from this software may not use "Indiana Univeristy" name nor may "Indiana Univeristy" appear in their name, without prior written permission of the Indiana University. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IFADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

48. xstream v. 1.4.2. Copyright © 2003-2006, Joe Walnes. Copyright © 2006-2009, 2011 XStream Committers. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR

BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

TRADEMARK NOTICE

The MarkLogic name and logo are registered trademarks of MarkLogic Corporation in the United States and/or other countries. Excel and PowerPoint are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Any other names or logos included in the Software, this notice or the documentation are property of the respective trademark owners.