
MarkLogic Server

XCC Developer's Guide

Release 4.2
October, 2010

Last Revised: 4.2-1, October, 2010

Copyright

© Copyright 2002-2012 by MarkLogic Corporation. All rights reserved worldwide.

This Material is confidential and is protected under your license agreement.

Excel and PowerPoint are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. This document is an independent publication of MarkLogic Corporation and is not affiliated with, nor has it been authorized, sponsored or otherwise approved by Microsoft Corporation.

Contains LinguistX, from Inxight Software, Inc. Copyright © 1996-2006. All rights reserved. www.inxight.com.

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

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

Contains Rosette Linguistics Platform 6.0 from Basis Technology Corporation, Copyright © 2004-2008 Basis Technology Corporation. All rights reserved.

This product includes 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 © 1998-2001 The OpenSSL Project. All rights reserved.

Contains software derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright © 1991-1992, RSA Data Security, Inc. Created 1991. All rights reserved.

Contains ICU with the following copyright and permission notice:

Copyright © 1995-2010 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.

Table of Contents

XCC Developer's Guide

Copyright	2
1.0 Introduction to XCC	5
1.1 Overview of XCC	5
1.1.1 XCC Client Libraries Communicate With an XDBC Server	5
1.1.2 Client-Server Architecture	6
1.1.3 Automatically Pools Connections	6
1.2 API and Other Documentation	7
1.3 XCC Requirements	7
1.3.1 XCC MarkLogic Server Requirements	7
1.3.2 XML Contentbase Connector for Java (XCC/J) Requirements	7
1.3.3 XML Contentbase Connector for .NET (XCC/.NET) Requirements	7
2.0 Programming in XCC	8
2.1 Configuring an XDBC Server	8
2.2 XCC Sessions	8
2.3 Point-In-Time Queries	9
2.4 Automatically Retries Exceptions	9
2.5 Coding Basics	9
2.6 Accessing SSL-Enabled XDBC App Servers	11
2.6.1 Creating a Trust Manager	11
2.6.2 Accessing a Keystore	13
2.6.3 Managing Client Side Authentication	14
3.0 Downloading and Using the XCC API	15
3.1 XCC/J Java Packages	15
3.2 XCC/.NET C# Packages	16
4.0 Using the Sample Applications	17
4.1 Setting Up Your Environment	17
4.1.1 Setting Up Your MarkLogic Server Environment	17
4.1.2 Setting Up Your Java Environment	17
4.1.3 Setting Up Your .NET Environment	18
4.2 Sample Applications	18
4.2.1 ContentFetcher	18
4.2.2 ContentLoader	19
4.2.3 HelloWorld	19

4.2.4	ModuleRunner	19
4.2.5	SimpleQueryRunner	20
5.0	Technical Support	21

1.0 Introduction to XCC

The XML Contentbase Connector (XCC) is an interface to communicate with MarkLogic Server from a Java or .NET middleware application layer. This chapter provides background on XCC and includes the following sections:

- [Overview of XCC](#)
- [API and Other Documentation](#)
- [XCC Requirements](#)

1.1 Overview of XCC

The XML Contentbase Connector (XCC) is used to communicate between a Java or .NET application layer and MarkLogic Server. XCC replaces the XDBC libraries, and the XDBC libraries are now deprecated. You can still use XDBC 3.0 to communicate with MarkLogic Server 4.0, but if you want to use any of the functionality newer than 3.1 (for example, point-in-time queries), you must use XCC; XDBC will no longer have any new features added (bug fixes only), so you should use XCC.

This section provides an overview of XCC and includes the following parts:

- [XCC Client Libraries Communicate With an XDBC Server](#)
- [Client-Server Architecture](#)
- [Automatically Pools Connections](#)

1.1.1 XCC Client Libraries Communicate With an XDBC Server

XCC has a set of client libraries that you use to build applications that communicate with MarkLogic Server. There are Java and .NET versions of the client libraries. XCC requires that an XDBC server is configured in MarkLogic Server.

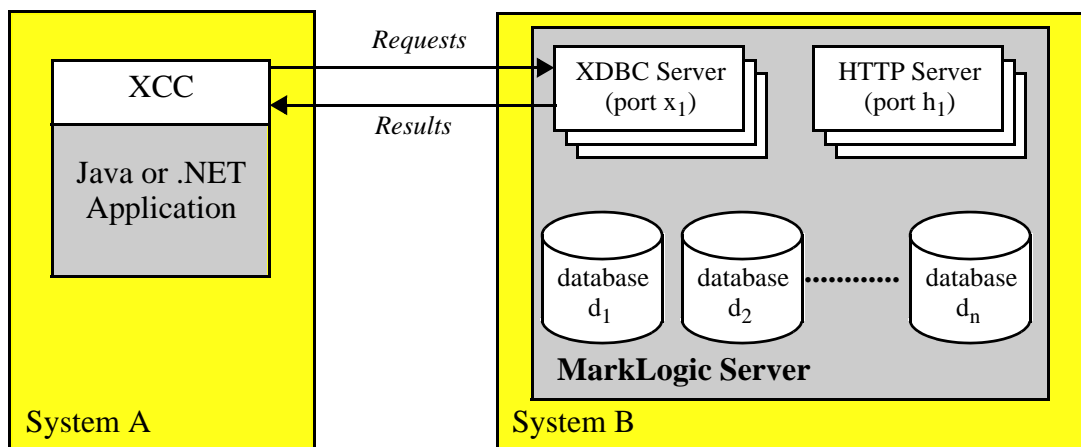
An XDBC server responds to XDBC and XCC requests. XDBC and XCC use the same wire protocol to communicate with MarkLogic Server. You can write applications either as standalone applications or ones that run in an application server environment. Your XCC-enabled application connects to a specified port on a system that is running MarkLogic Server, and communicates with MarkLogic Server by submitting requests (for example, XQuery statements) and processing the results returned by those programs. These XQuery programs can incorporate calls to XQuery functions stored and accessible by MarkLogic Server, and accessible from any XDBC-enabled application. The XQuery programs can perform the full suite of XQuery functionality, including loading, querying, updating and deleting content.

XQuery requests submitted via XCC return results as specified by the XQuery code. These results can include XML and a variety of other datatypes. It is the XCC application's responsibility to parse, process and interpret these results in a manner appropriate to the variety of datatypes available. There are a number of publicly available libraries for assisting with this task, or you

may write your own code. In order to accept connections from XCC-enabled applications, MarkLogic Server must be configured with an XDBC Server listening on the designated port. Each XDBC Server connects by default to a specific database within MarkLogic Server, but XCC provides the ability to communicate with any database in the MarkLogic Server cluster to which your application connects (and for which you have the necessary permissions and privileges).

1.1.2 Client-Server Architecture

XCC communicates with MarkLogic Server with a client-server architecture, where the XCC application is the client and MarkLogic Server is the server. The following figure illustrates the high-level architecture:



As shown in the diagram above, the XCC-enabled application can run on the same system as an instance of MarkLogic Server (a host), or it can run on a completely different system, as long as the two systems are networked together.

In the diagram, the XCC application running on System A has opened an XDBC connection to port x_1 on System B. On System B, MarkLogic Server is configured with an XDBC Server listening to port x_1 , and that XDBC Server connects to database_{d1}. Consequently, the configuration shown in the diagram above allows the XCC application on System A to submit XQuery requests (including query, load, update and delete) for evaluation against database_{d1}.

Note: Do not use an HTTP load balancer with an XCC application. XCC communicates with MarkLogic Server via the XDBC protocol, which is not the same as HTTP.

1.1.3 Automatically Pools Connections

XCC automatically does connection pooling, so you do not need to write any connection pooling logic in your application. The XCC `session` object automatically obtains and releases connections for XCC applications as needed.

1.2 API and Other Documentation

This document provides an introduction to the XCC developer libraries. For detailed API documentation for XCC and for MarkLogic Server, or to learn how to configure XDBC servers in MarkLogic Server, see the appropriate documents:

- Java API documentation (Javadoc available on developer.marklogic.com)
- .NET API documentation (available on developer.marklogic.com)
- *MarkLogic Server Application Developer's Guide*
- *MarkLogic Server Administrator's Guide*
- *MarkLogic XQuery and XSLT Function Reference*

1.3 XCC Requirements

This section lists the requirements for XCC and has the following parts:

- [XCC MarkLogic Server Requirements](#)
- [Do not use an HTTP load balancer with an XCC application. XCC communicates with MarkLogic Server via the XDBC protocol, which is not the same as HTTP.](#)
- [XML Contentbase Connector for .NET \(XCC/.NET\) Requirements](#)

1.3.1 XCC MarkLogic Server Requirements

XCC requires MarkLogic Server 3.x or later. You must connect to MarkLogic Server 3.1 or later to use any of the 3.1-specific or later features (for example, point-in-time queries).

Note: Do not use an HTTP load balancer with an XCC application. XCC communicates with MarkLogic Server via the XDBC protocol, which is not the same as HTTP.

1.3.2 XML Contentbase Connector for Java (XCC/J) Requirements

XCC/J has the following requirements:

- Java 1.5 or later
- MarkLogic Server 3.1 or later (on any platform)

1.3.3 XML Contentbase Connector for .NET (XCC/.NET) Requirements

XCC/.NET has the following requirements:

- .NET 2.0 Service Pack 1 (SP1), or later
- MarkLogic Server 3.1 or later (on any platform)

2.0 Programming in XCC

XCC allows you to create multi-tier applications that communicate with MarkLogic Server as the underlying content repository. This chapter describes some of the basic programming concepts used in XCC. It includes the following sections:

- [Configuring an XDBC Server](#)
- [XCC Sessions](#)
- [Point-In-Time Queries](#)
- [Automatically Retries Exceptions](#)
- [Coding Basics](#)
- [Accessing SSL-Enabled XDBC App Servers](#)

2.1 Configuring an XDBC Server

Use the Admin Interface to set up an XDBC server, specifying a name, port, a database to access, and other configuration parameters. For detailed instructions how to configure an XDBC Server, see the *Administrator's Guide*. You need an XDBC Server for an XCC program to communicate with MarkLogic Server.

2.2 XCC Sessions

XCC programs use the `Session` interface to set up and control communication with MarkLogic Server. XCC automatically creates and releases connections to MarkLogic Server as needed, and automatically pools the connections so that multiple requests are handled efficiently.

A `Session` handles authentication with MarkLogic Server and holds a dynamic state, but it is a lightweight object. It is OK to create and release `Session` objects as needed and as makes logical sense for your program. Do not expend effort to pool and reuse them, however, because they are not expensive to create. For example, if your program is doing multiple requests one after another, create a `Session` object at the beginning and close it when the last request is complete.

You set up the connection details with the `ContentSource` object. You can submit the connection details when you invoke the XCC program with a URL that has the following form:

```
xcc://username:password@host:port/database
```

Also, there are discrete arguments to the constructors in the API to set up any or all portions of the connection details.

2.3 Point-In-Time Queries

Point-in-time queries allow you to query older versions of content in a database. In an XCC application, you set up the options for any requests submitted to MarkLogic Server with the `RequestOptions` class. One of the options you can set is the effective point-in-time option. Therefore, to set up a query to run at a different point in time, you just set that option (the `setEffectivePointInTime` method in Java) on the `RequestOptions`. The query will then run at the specified point in time.

There are several things you must set up on MarkLogic Server in order to perform point-in-time queries. For details, see the “Point-In-Time Queries” chapter of the *Application Developer’s Guide*.

2.4 Automatically Retries Exceptions

Certain exceptions that MarkLogic Server throws are *retryable*; that is, the exception is thrown because of a condition that is transitory, and applications can try the request again after getting the exception. XCC will automatically retry any retryable exceptions. You can control the maximum number of retryable exceptions with the `RequestOptions` interface.

2.5 Coding Basics

To use XCC, there are several basic things you need to do in your Java or .NET code:

- Import the needed libraries.
- Set up the `ContentSource` object to authenticate against MarkLogic Server.
- Create a new `Session` object.
- Add a request to the session object.
- Get back a `ResultSequence` object from MarkLogic Server.
- Do something with the results (print them out, for example).
- Close the session.

The following are Java code samples that illustrate these basic design patterns:

```
package com.marklogic.xcc.examples;

import com.marklogic.xcc.ContentSource;
import com.marklogic.xcc.ContentSourceFactory;
import com.marklogic.xcc.Session;
import com.marklogic.xcc.Request;
import com.marklogic.xcc.ResultSequence;

URI uri = new URI("xcc://user:pass@localhost:8000/mycontent");
ContentSource contentSource =
    ContentSourceFactory.newContentSource (uri);

Session session = contentSource.newSession();
```

```
Request request = session.newAdhocQuery ("\"Hello World\"");  
  
ResultSequence rs = session.submitRequest (request);  
  
System.out.println (rs.asString());  
  
session.close();
```

2.6 Accessing SSL-Enabled XDBC App Servers

There are three basic approaches for an XCC application to create a secure connection to an SSL-enabled XDBC App Server, which include:

- [Creating a Trust Manager](#)
- [Accessing a Keystore](#)
- [Managing Client Side Authentication](#)

These approaches are described in this section and demonstrated in the `HelloSecureWorld.java` example distributed with your MarkLogic XCC software distribution.

2.6.1 Creating a Trust Manager

This section describes how to use a simple Trust Manager for X.509-based authentication. The Trust Manager shown here does not validate certificate chains and is therefore unsafe and should not be used for production code. See your Java documentation for details on how to create a more robust Trust Manager for your specific application or how to obtain a Certificate Authority from a keystore.

Note: If you want to create an XCC application that works with MarkLogic Server version 4.2 (with SSL) and version 4.0 (without SSL), you can use the MarkLogic Server 4.2 `marklogic-xcc-4.2.x.jar` file on your 4.0 system.

To enable SSL access using a trust manager, import the following classes in addition to those described in “Coding Basics” on page 9:

```
import javax.net.ssl.SSLContext;
import com.marklogic.xcc.SecurityOptions;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

import java.security.KeyManagementException;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateException;
```

Create a trust manager and pass it to the `SSLContext.init()` method:

```
protected SecurityOptions newTrustOptions()
    throws Exception
{
    TrustManager[] trust = new TrustManager[] { new X509TrustManager() {
        public void checkClientTrusted(
            X509Certificate[] x509Certificates,
            String s)
            throws CertificateException {
            // nothing to do
        }

        public void checkServerTrusted(
            X509Certificate[] x509Certificates,
            String s)
            throws CertificateException {
            // nothing to do
        }

        public X509Certificate[] getAcceptedIssuers() {
            return null;
        }
    }
};

SSLContext sslContext = SSLContext.getInstance("SSLv3");
sslContext.init(null, trust, null);
return new SecurityOptions(sslContext);
}
```

Call `ContentSourceFactory.newContentSource()` with a host name, port, user name, password, and SSL security options defined by `newTrustOptions()`:

```
ContentSource cs =
    ContentSourceFactory.newContentSource (host,
                                           port,
                                           username,
                                           password,
                                           null,
                                           newTrustOptions());
```

Note: If you are passing a URI to `ContentSourceFactory.newContentSource()`, specify a connection scheme of `xccs`, rather than `xcc`, as shown in “Accessing a Keystore” on page 13.

2.6.2 Accessing a Keystore

You can use the Java `keytool` utility to import a MarkLogic certificate into a keystore. See the Java JSSE documentation for details on the use of the `keytool` and your keystore options.

You can explicitly specify a keystore, as shown in this example, or you can specify a null keystore. Specifying a null keystore causes the `TrustManagerFactory` to locate your default keystore, as described in the *Java Secure Socket Extension (JSSE) Reference Guide*.

To enable SSL by accessing certificates in a keystore, import the following classes in addition to those described in “Coding Basics” on page 9:

```
import com.marklogic.xcc.SecurityOptions;
import com.marklogic.xcc.ContentSource;
import com.marklogic.xcc.ContentSourceFactory;

import java.io.FileInputStream;
import java.net.URI;
import javax.net.ssl.KeyManager;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;
import javax.net.ssl.SSLContext;

import java.security.KeyStore;
import java.security.cert.X509Certificate;
```

Get the signed certificate from a keystore and pass it to the `SSLContext.init()` method:

```
protected SecurityOptions newTrustOptions()
    throws Exception
{
    // Load key store with trusted signing authorities.
    KeyStore trustedKeyStore = KeyStore.getInstance("JKS");
    trustedKeyStore.load(
        new FileInputStream("C:/users/myname/.keystore"),
        null);

    // Build trust manager to validate server certificates using the
    // specified key store.
    TrustManagerFactory trustManagerFactory =
        TrustManagerFactory.getInstance("SunX509");
    trustManagerFactory.init(trustedKeyStore);
    TrustManager[] trust = trustManagerFactory.getTrustManagers();

    SSLContext sslContext = SSLContext.getInstance("SSLv3");
    sslContext.init(null, trust, null);
    return new SecurityOptions(sslContext);
}
```

Call `ContentSourceFactory.newContentSource()` with a URI:

```
ContentSource cs =
    ContentSourceFactory.newContentSource (uri,
                                           newTrustOptions());
```

The URI is passed from the command line in the form of:

```
xccs://username:password@hostname:port
```

2.6.3 Managing Client Side Authentication

You can define a `KeyManager`, if your client application is required to send authentication credentials to the server. The following example adds client authentication to the `newTrustOptions` method shown in “Accessing a Keystore” on page 13:

```
protected SecurityOptions newTrustOptions()
    throws Exception
{
    // Load key store with trusted signing authorities.
    KeyStore trustedKeyStore = KeyStore.getInstance("JKS");
    trustedKeyStore.load(
        new FileInputStream("C:/users/myname/.keystore"),
        null);

    // Build trust manager to validate server certificates using the
    // specified key store.
    TrustManagerFactory trustManagerFactory =
        TrustManagerFactory.getInstance("SunX509");
    trustManagerFactory.init(trustedKeyStore);
    TrustManager[] trust = trustManagerFactory.getTrustManagers();

    // Load key store with client certificates.
    KeyStore clientKeyStore = KeyStore.getInstance("JKS");
    clientKeyStore.load(
        new FileInputStream("C:/users/myname/.keystore"),
        null);

    // Get key manager to provide client credentials.
    KeyManagerFactory keyManagerFactory =
        KeyManagerFactory.getInstance("SunX509");
    keyManagerFactory.init(clientKeyStore, "passphrase");
    KeyManager[] key = keyManagerFactory.getKeyManagers();

    // Initialize the SSL context with key and trust managers.
    SSLContext sslContext = SSLContext.getInstance("SSLv3");
    sslContext.init(key, trust, null);
    return new SecurityOptions(sslContext);
}
```

3.0 Downloading and Using the XCC API

The XCC API is available by downloading the XCC packages from developer.marklogic.com.

This chapter describes the basics of setting up your XCC environment, and includes the following sections:

- [XCC/J Java Packages](#)
- [XCC/.NET C# Packages](#)

For a description of the sample applications included with XCC, see “Using the Sample Applications” on page 17.

3.1 XCC/J Java Packages

The Java distribution of XCC has the following directory structure:

Document or Directory	Description
docs/	Includes the Javadoc for XCC in both expanded HTML and compressed zip format.
lib/	Contains the <code>marklogic-xcc-4.2.x.jar</code> file, which is the XCC libraries, and the <code>marklogic-xcc-examples-4.2.x.jar</code> file, which has the compiled versions of the sample applications. Note that the name of the XCC jar file has the version number encoded.
src/	Includes the source code for the sample applications.
Readme.txt	Includes the version number and any last-minute updates not included in the documentation.

3.2 XCC/.NET C# Packages

The .NET distribution of XCC has the following directory structure:

Document or Directory	Description
<code>dll/</code>	Contains the <code>MarkLogicXcc.dll</code> file, which is the XCC libraries, as well as some supporting DLLs.
<code>docs/</code>	Includes the .NET API documentation for XCC in both expanded HTML and compressed zip format.
<code>src/</code>	Includes the source code for the sample applications.
<code>Readme.txt</code>	Includes the version number and any last-minute updates not included in the documentation.

4.0 Using the Sample Applications

The XCC packages contain a number of sample applications. Each sample application is provided along with its source code, giving you a starting point for creating your own applications. This chapter describes the sample applications and contains the following sections:

- [Setting Up Your Environment](#)
- [Sample Applications](#)

4.1 Setting Up Your Environment

Before running the sample applications, be sure to set up the necessary environment to run the application. This section has the following parts:

- [Setting Up Your MarkLogic Server Environment](#)
- [Setting Up Your Java Environment](#)
- [Setting Up Your .NET Environment](#)

4.1.1 Setting Up Your MarkLogic Server Environment

Before you run the sample applications, complete the following steps:

1. Install MarkLogic Server, or have a MarkLogic Server installation to which you can connect. For details on installing MarkLogic Server, see the *Installation Guide*.
2. Create and configure an XDBC Server using the Admin Interface. See the MarkLogic Server *Administrator's Guide* for details on how to create and configure an XDBC Server.
3. Configure a user for the XDBC Server you created. For example, add a user to the security database with the username as `user` and the password as `pass`. See the MarkLogic Server *Administrator's Guide* for details on adding a user to the security database.

4.1.2 Setting Up Your Java Environment

If you are using XCC/J, you must have Java installed on your client machine. Additionally, you will need the following set up to run the sample applications:

- Set your `JAVA_HOME` environment variable, if it is not already set. For example, if you are running a Windows machine, set `JAVA_HOME` in a command window as in the following example:

```
set JAVA_HOME=c:\Sun\SDK\jdk
```

Substitute the directory in which Java is installed in your environment.

- Set your `CLASSPATH` environment variable correctly, or use the `-classpath` option to pass the appropriate classpath on the command line. Make sure to use the correct name for the `marklogic-xcc-4.2.x.jar` file in your `CLASSPATH`, as the name corresponds to the service release version number.

4.1.3 Setting Up Your .NET Environment

If you are using XCC/.NET, you must have .NET 2.0 Service Pack 1 (SP1), or later installed on your client machine.

4.2 Sample Applications

The source code and API documentation for the sample applications are included in the XCC packages. A `marklogic-xcc-examples-4.2.x.jar` file is included with the Java distribution of XCC. The commands to launch the sample programs in this section assume you have renamed the jar file to `xccexamples.jar`. The sample applications are as follows:

Sample	Description
ContentFetcher	This class fetches documents from the contentbase and writes their serialized contents to a provided <code>OutputStream</code> .
ContentLoader	This program accepts a server URI (in the format expected by <code>ContentSourceFactory.newContentSource(java.net.URI)</code>) and one or more file pathnames of documents to load.
HelloWorld	This very simple class prints out the string "Hello World".
ModuleRunner	This is a very simple class that will invoke a named XQuery module on the server and return the result.
SimpleQueryRunner	This is a very simple class that will submit an XQuery string to the server and return the result.

4.2.1 ContentFetcher

This program fetches a document from MarkLogic Server and serializes its contents. You can serialize the contents to the standard output (display it on the screen) or to a file using the `-o` option. The following is a sample command to run the `ModuleRunner` class:

```
java -classpath "c:/xcc/marklogic-xcc-4.2.0.jar;
c:/xcc/xccexamples.jar"
com.marklogic.xcc.examples.ContentFetcher
xcc://username:password@localhost:8021
/mydocs/hello.xml -o myHelloFile.xml
```

This sends the contents of the document at `/mydocs/hello.xml` to the file `myHelloFile.xml` (in the same directory in which the command is run). It connects to the default database of the XDBC Server listening on port 8021 of the local machine, using the credentials `username` and `password` to authenticate the connection.

4.2.2 ContentLoader

This program loads the specified document in the database. It loads the file with a URI equal to the fully-qualified pathname of the file. The following is a sample command to run the `ContentLoader` class:

```
java -classpath "c:/xcc/marklogic-xcc-4.2.0.jar;  
c:/xcc/xccexamples.jar"  
com.marklogic.xcc.examples.ContentLoader  
xcc://username:password@localhost:8021 hello.xml
```

This loads the file at `hello.xml` to a document with the fully-qualified pathname of `hello.xml` (for example, `c:\xcc\examples\hello.xml`). It loads it into the default database of the XDBC Server listening on port 8021 of the local machine, using the credentials `username` and `password` to authenticate the connection.

4.2.3 HelloWorld

This program runs a query on MarkLogic Server that returns the string "Hello World". The following is a sample command to run the `HelloWorld` class:

```
java -classpath "c:/xcc/marklogic-xcc-4.2.0.jar;  
c:/xcc/xccexamples.jar"  
com.marklogic.xcc.examples.HelloWorld  
xcc://username:password@localhost:8021
```

4.2.4 ModuleRunner

This program allows you to invoke a module on the server. The module must exist under the XDBC server root, either in the database (when a modules database is configured) or on the filesystem (when the filesystem is configured for modules). The following is a sample command to run the `ModuleRunner` class:

```
java -classpath "c:/xcc/marklogic-xcc-4.2.0.jar;  
c:/xcc/xccexamples.jar"  
com.marklogic.xcc.examples.ModuleRunner  
xcc://username:password@localhost:8021 hello.xqy
```

This invokes the module named `hello.xqy`. The request is submitted to the XDBC Server running on the local machine at port 8021, using the credentials `username` and `password` to authenticate the connection. The module path is resolved relative to the XDBC Server root.

4.2.5 SimpleQueryRunner

This program allows you to store XQuery in a file and then submit the XQuery to MarkLogic Server. The following is a sample command to run the `SimpleQueryRunner` class:

```
java -classpath "c:/xcc/marklogic-xcc-4.2.0.jar;  
c:/xcc/xccexamples.jar"  
com.marklogic.xcc.examples.SimpleQueryRunner  
xcc://username:password@localhost:8021 hello.xqy
```

This submits the contents of the `hello.xqy` file to a MarkLogic Server XDBC Server running on the local machine at port 8021, using the credentials `username` and `password` to authenticate the connection.

5.0 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.