

1 Quick Start

This quick start provides basic information for getting started with DataDirect XQuery immediately after installation. Specifically, this quick start includes the following topics:

1. [Setting the CLASSPATH](#)
2. [Configuring Connections](#)
3. [Developing a Java Application that Executes a Query](#)

In addition, this quick start describes the DataDirect XQuery command-line utility that is available for quickly running and testing XQueries through a console window. See [“Using the Command-Line Utility” on page 7](#) for details.

For complete information about the many DataDirect XQuery features, we recommend that you read the [DataDirect XQuery User’s Guide and Reference](#).

For information about product requirements, refer to the [DataDirect XQuery Installation Guide](#).

1. Setting the CLASSPATH

Only one DataDirect XQuery jar file, `ddxq.jar`, must be defined in your CLASSPATH. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate DataDirect XQuery on your computer. If `ddxq.jar` is not defined on your CLASSPATH, you receive a `ClassNotFoundException` exception when trying to use DataDirect XQuery.

Set your CLASSPATH to include:

```
install_dir/lib/ddxq.jar
```

where *install_dir* is the path to your DataDirect XQuery installation directory.

2. Configuring Connections

DataDirect XQuery provides multiple ways to configure connections to XML data sources and relational data sources. This quick start demonstrates the method of using XQJ to create a DDXQDataSource instance in your Java application explicitly. See the *DataDirect XQuery User's Guide and Reference* for information about other methods of configuring connections.

XML Data Source Connections

If your Java application contains queries that access an XML file, you can directly access the file as shown in the following XQJ code, where the location and name of the XML file is specified as a parameter of fn:doc(), an XQuery function.

```
DDXQDataSource ds = new DDXQDataSource();
XQConnection conn = ds.getConnection();
conn.createExpression().executeQuery("doc('path_and_filename')");
```

Relational Data Source Connections

How you configure connection information for relational databases using XQJ depends on whether you are accessing a single database or multiple databases. This quick start demonstrates configuring connection information for accessing a single database. For information about accessing multiple

databases, see the [DataDirect XQuery User's Guide and Reference](#).

To configure a single relational data source connection, use the DDXQDataSource class as shown in the following XQJ code, which specifies a connection URL for the relational data source you want to access and specifies the user ID and password required to access the relational data source.

See ["Sample Connection URLs" on page 5](#) for samples of a connection URL for each supported database.

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:dbtype://server_name:port;property=value[;...]");
XQConnection conn = ds.getConnection("myuserid", "mypsword");
```

where:

<i>dbtype</i>	Valid values are db2, oracle, sqlserver, and sybase.
<i>server_name</i>	The TCP/IP address or TCP/IP host name of the database server to which you are connecting.
<i>port</i>	The number of the TCP/IP port.
<i>property=value</i>	Connection properties. For a complete list of connection properties for all databases, see DataDirect XQuery User's Guide and Reference . All connection property names are case-insensitive. For example, DatabaseName is the same as databasename. Note that DB2 requires that some properties are specified in the connection URL. See "DB2 Required Connection Properties" on page 4 for a list and description of the required DB2 connection properties.

You, also, can specify a user ID and password in the connection URL, for example:

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:oracle://server1:1521;user=myuserID;
password=mypswd");
```

DB2 Required Connection Properties

DB2 Property	Description
DatabaseName Required for Linux/UNIX/Windows	The name of the database to which you want to connect. This property is supported only for DB2 UDB for Linux/UNIX/Windows.
LocationName Required for z/OS and iSeries	The name of the DB2 location that you want to access. For DB2 UDB for z/OS, your system administrator can determine the name of your DB2 location using the following command: DISPLAY DDF For DB2 UDB for iSeries, your system administrator can determine the name of your DB2 location using the following command. The name of the database that is listed as *LOCAL is the value you should use for this property. WRKRDBDIRE This property is supported only for DB2 UDB for z/OS and DB2 UDB for iSeries.

Sample Connection URLs

The following URLs are examples of the minimum information that must be specified in a connection URL.

DB2 UDB for Linux/UNIX/Windows

```
jdbc:xquery:db2://server_name:50000;databaseName=your_database
```

DB2 UDB for z/OS and iSeries

```
jdbc:xquery:db2://server_name:446;locationName=db2_location
```

Microsoft SQL Server

```
jdbc:xquery:sqlserver://server_name:1433
```

Oracle

```
jdbc:xquery:oracle://server_name:1521
```

Sybase

```
jdbc:xquery:sybase://server_name:5000
```

3. Developing a Java Application that Executes a Query

Using DataDirect XQuery, a Java application uses XQJ to execute a query. The Java package name of the XQJ classes is:

```
com.ddtek.xquery3
```

The Java class name of the DataDirect XQuery implementation of the XQJ standard interface, XQDataSource, is:

```
com.ddtek.xquery3.xqj.DDXQDataSource
```

The following sample Java code illustrates the basic steps that an application would perform to execute an XQuery expression using DataDirect XQuery. This example accesses a Microsoft SQL Server data source. To simplify the code, this example contains no error handling.

```
// import the XQJ classes
import com.ddtek.xquery3.*;
import com.ddtek.xquery3.xqj.DDXQDataSource;

// establish a connection to a relational data source
// specify the URL and the user ID and password
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
XQConnection conn = ds.getConnection("myuserid", "mypasswd");
// create an expression object that is used to execute a query
XQExpression xqExpression = conn.createExpression();

// the query
String es = "for $h in collection('holdings')/holdings " +
           "where $h/stockticker='AMZN' " +
           "return $h";

// execute the query
XQResultSequence result = xqExpression.executeQuery(es);
result.writeSequence(System.out, null);

// free all resources
result.close();
xqExpression.close();
conn.close();
```

NOTE: XQJ examples are shipped with the product and are located in the /examples subdirectory in the DataDirect XQuery installation directory.

Using the Command-Line Utility

The DataDirect XQuery command-line utility allows you to quickly run and test XQueries through a console window.

To invoke this utility, enter the following command at a prompt from the lib subdirectory of your DataDirect XQuery installation directory (for example, ddxq30/lib):

```
java -jar ddxq.jar
```

Alternatively, you can specify the path to the lib directory in the command line, for example:

```
java -jar ddxq30/lib/ddxq.jar
```

NOTE: If your XQuery needs to locate classes other than the DataDirect XQuery classes, for example, if you are specifying a custom URI resolver, you must either:

- Set your CLASSPATH to include the path to the jar files or directories for these classes and invoke the utility using the following command:

```
java com.ddtek.xquery.Query
```

- Add the class path to the command line:

```
java -cp c:\myClasses com.ddtek.xquery.Query
```

See [Example 8](#).

The following table lists the options available for the utility.

Option	Description
<code>-cr classname</code>	Specifies the CollectionURIResolver class to use. See the DataDirect XQuery User's Guide and Reference for information about Collection URI resolvers. See the previous NOTE about setting your CLASSPATH for custom URI resolvers.
<code>-e [xhtml xml]</code>	Generates an XQuery execution plan and, optionally, specifies the format of the plan. If a format is not specified, XHTML is generated. See the DataDirect XQuery User's Guide and Reference for an explanation of execution plans.
<code>-jdbc jdbcurl</code>	Specifies a connection URL. See " Relational Data Source Connections " on page 2.
<code>-mr classname</code>	Specifies the ModuleURIResolver class to use. See DataDirect XQuery User's Guide and Reference for information about Library Module URI resolvers. See the previous NOTE about setting your CLASSPATH for custom URI resolvers.
<code>-noext</code>	Disallows calls to Java methods.
<code>-o filename</code>	Sends results (output) to specified file.
<code>-option property=value</code>	Specifies XQuery or JDBC global options to use.
<code>-p</code>	Displays a stack trace in case of an exception.
<code>-r classname</code>	Specifies the URIResolver class to use. See DataDirect XQuery User's Guide and Reference for information about Document URI resolvers. See the previous NOTE about setting your CLASSPATH for custom URI resolvers.

Option	Description
<code>-s file URI</code>	Specifies an initial context item in the form of a file name or a URI.
<code>-t</code>	Displays version and timing information.
<code>-?</code>	Displays the help for the command-line utility.
<code>param=value</code>	Specifies a query string parameter and its value.
<code>#param=value</code>	Specifies a query number parameter and its value. On UNIX and Linux, the value for this option must be enclosed with double quotes, for example: <code>java -jar ddxq.jar q.xq "#i=2"</code>
<code>+param=value</code>	Specifies a query document parameter and its value.
<code>!option=value</code>	Specifies a serialization option and its value. See the DataDirect XQuery User's Guide and Reference for a list of serialization options.

Example 1: Executes a Simple XQuery

This example executes the simple query {2+5}.

```
java -jar ddxq.jar {2+5}
```

Example 2: Retrieves Values from an Initial Context Item

This example retrieves all values for UserId from the initial context item users.xml.

```
java -jar ddxq.jar -s ..\..\examples\xml\users.xml  
{//users/UserId}
```


Example 8: Specifies a Document URI

This example retrieves all values for `UserId`, specifies a document URI, and writes the results to a file named `out.xml`.

```
java -cp c:\myClasses com.ddtek.xquery.Query  
-r myURIResolver -o out.xml {doc('users.xml')/users/UserId}
```

NOTES:

- For information about getting started with the examples shipped with DataDirect XQuery, see the `examplesreadme.txt` file in the `/examples` subdirectory in the DataDirect XQuery installation directory.
- For information about using the DataDirect XML Converters, refer to:
<http://www.xmlconverters.com>
- For information about using Stylus Studio, refer to:
http://www.stylusstudio.com/xquery/datadirect_xquery.html
- For information about getting started with <Oxygen/> XML Editor for Eclipse (DataDirect XQuery Edition), refer to:
http://www.xquery.com/xml_tools