

# Quick Start: DataDirect XQuery®

This quick start provides basic information for getting started with DataDirect XQuery immediately after installation. It covers the following topics:

- ["Getting Started with DataDirect XQuery"](#)
- ["Using the Command-Line Utility"](#)
- ["Additional Resources"](#)

---

## Getting Started with DataDirect XQuery

This section shows you how to get up and running with DataDirect XQuery. It covers the following topics:

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

### 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.

NOTE: If you are connecting to PostgreSQL, you must add the PostgreSQL JDBC driver jar file to the CLASSPATH in addition to ddxq.jar. Refer to your PostgreSQL JDBC driver documentation for the name of the jar file.

## 2. Configuring Connections

DataDirect XQuery provides multiple ways to configure connections to XML data sources and relational data sources. This section shows how to use XQJ to create a DDXQDataSource instance in your Java application explicitly.

### ***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 section shows how to configure connection information to access 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 example. This example specifies a connection URL (represented by *URL*) for the relational data source that you want to access and the user ID and password required to access the relational data source.

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("URL");
XQConnection conn = ds.getConnection("myuserid", "mypsword");
```

## ***Sample Connection URLs***

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

### **DB2 for Linux/UNIX/Windows**

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

### **DB2 for z/OS and iSeries**

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

### **Informix**

```
jdbc:xquery:informix://server_name;1526;InformixServer=dbserver_name
```

### **Microsoft SQL Server**

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

### **MySQL Enterprise**

```
jdbc:xquery:mysql://server_name
```

### **Oracle**

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

## PostgreSQL

`jdbc:postgresql:your_database`

## 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 example, this code does not include 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", "mypswn");
// 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, ddxq/lib):

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

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

```
java -jar ddxq/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 perform one of the following actions:

- 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
```

NOTE: If you are connecting to PostgreSQL, you must add the PostgreSQL JDBC driver jar file to the CLASSPATH in addition to ddxq.jar. Refer to your PostgreSQL JDBC driver documentation for the name of the jar file.

- Add the class path to the command line:

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

See [Example 8](#).

NOTE: If you are connecting to PostgreSQL, you must add the PostgreSQL JDBC driver jar file to the CLASSPATH in addition to ddxq.jar. Refer to your PostgreSQL JDBC driver documentation for the name of the jar file.

The following table lists the options available for the utility.

| Option                                    | Description   |
|---|---|
| <code>-cr <i>classname</i></code>         | Specifies the CollectionURIResolver class to use. See the <a href="#">NOTE</a> about setting your CLASSPATH for custom URI resolvers.   |
| <code>-e [<i>xhtml</i> <i>xml</i>]</code> | Generates an XQuery execution plan and, optionally, specifies the format of the plan. If a format is not specified, XHTML is generated. |

| Option                                  | Description   |
|---|---|
| <code>-jdbc jdbcurl</code>              | <p>Specifies a connection URL. See <a href="#">“Relational Data Source Connections”</a> on page 2.</p> <p>NOTE: On UNIX and Linux, the value for this option must be enclosed with double quotes, for example:</p> <pre data-bbox="701 435 1286 531">java -jar ddxq.jar -jdbc "jdbc:xquery:sqlserver://localhost:1433; databaseName=pubs;user=sa"</pre> |
| <code>-mr classname</code>              | <p>Specifies the ModuleURIResolver class to use. See the <a href="#">NOTE</a> about setting your CLASSPATH for custom URI resolvers.</p>  |
| <code>-noext</code>                     | <p>Disallows calls to Java methods.</p>   |
| <code>-o filename</code>                | <p>Sends results (output) to specified file.</p>  |
| <code>-option<br/>property=value</code> | <p>Specifies XQuery or JDBC global options to use.</p>  |
| <code>-p</code>                         | <p>Displays a stack trace in case of an exception.</p>  |
| <code>-r classname</code>               | <p>Specifies the URIResolver class to use. See the <a href="#">NOTE</a> about setting your CLASSPATH for custom URI resolvers.</p>  |
| <code>-s file URI</code>                | <p>Specifies an initial context item in the form of a file name or a URI.</p>   |
| <code>-t</code>                         | <p>Displays version and timing information.</p>   |
| <code>-?</code>                         | <p>Displays the help for the command-line utility.</p>  |
| <code>param=value</code>                | <p>Specifies a query string parameter and its value.</p>  |
| <code>#param=value</code>               | <p>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:</p> <pre data-bbox="701 1520 1143 1545">java -jar ddxq.jar q.xq "#i=2"</pre>  |

| Option                     | Description   |
|----------------------------|---|
| <code>+param=value</code>  | Specifies a query document parameter and its value. |
| <code>!option=value</code> | Specifies a serialization option and its value.     |

### 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 3: Retrieves Values and Writes Them to a File

This example retrieves all values for UserId and writes the results to a file named out.xml.

```
java -jar ddxq.jar -o out.xml
{doc('..\..\examples\xml\users.xml')/users/UserId}
```

### Example 4: Executes an XQuery in a File

This example executes the XQuery contained in the file myXQuery.xq using the initial context item input.xml.

```
java -jar ddxq.jar -s input.xml myXQuery.xq
```

### Example 5: Binds a Query Document Parameter

This example executes the XQuery contained in the file myXQuery.xq binding the query document parameter inputDoc to the input.xml document.

```
java -jar ddxq.jar myXQuery.xq +inputDoc=input.xml
```

### Example 6: Binds a Query String Parameter and Sets an Option

This example executes the XQuery contained in the file `myXQuery.xq` binding the query string parameter `param1` to the character string `Jonathan` and setting the serialization option `indent` to `yes` so that results are indented.

```
java -jar ddxq.jar myXQuery.xq param1=Jonathan !indent=yes
```

### Example 7: Accesses a Relational Data Source

This example executes the XQuery contained in the file `myXQuery2.xq` that accesses a relational data source. See the [NOTE](#) about specifying connection URLs.

```
java -jar ddxq.jar -jdbc
"jdbc:xquery:sqlserver://localhost:1433;databaseName=pubs;
user=sa" myXQuery2.xq
```

### 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}
```

---

## Additional Resources

In addition to this quick start, you might find these resources useful:

- 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](#).

© 2008. DataDirect Technologies Corp. All rights reserved.  
11/08, 4.0