**JDBC Extensibility**

# Overview

ThingWorx can use JDBC drivers to connect to any JDBC compatible database (such as SQL Server, MySQL, Oracle, etc.). It can also be done using the ThingWorx Edge Microserver and the ThingWorx Host/Resource using OLEDB and ODBC if the Database happens to be behind a firewall.

If a JDBC Extension does not exist in ThingWorx for your database, you can easily create one and import it.

Creating such an extension can be done in two ways:

1. Manually editing an example metadata file, downloading the JDBC driver and packaging them together.
2. Using the JDBCCreator Extension that automates those steps for you.

**Disclaimer**: in both steps you need to obtain the JDBC drivers on your own import the extension manually afterwards.

For more information on creating a database extension, follow the steps below.

You can also view the [following tutorial video](https://www.youtube.com/watch?v=ZlxFok1Y_HU) on creating a JDBC extension in ThingWorx.

# Option 1. Creating a JDBC Extension for ThingWorx

1. Obtain the example metadata file Example Database Metadata.xml.
2. Download the appropriate JDBC driver.
3. Build the extension structure by creating the directory lib/common
4. Place the JAR file in the following directory location: lib/common/<JDBC driver jar file>
5. Modify the **name** attribute of the **ExtensionPackage** entity in the metadata. xml file as necessary.
6. Point the **file** attribute of the **FileResource** entity to the name of the JDBC JAR file.
7. The metadata also contains a Thing Template. The name is set to

**MySqlServer**, but can be modified as needed.

1

1. Select the lib folder and metadata.xml file and send to a zip archive.

 **Tip**

The name of the zip archive should match the name given in the name

attribute of the **ExtensionPackage** entity in the metadata.xml file.

1. Import the newly created extension.
2. To use the JDBC extension, create a new Thing and assign the new Thing Template that was imported with the JDBC extension.

# Option 2. Using the JDBC Creator Extension.

1. Open the JdbcCreator mashup, click the “Choose File” button and select the JDBC driver file you want to use from your local file system
2. Click the “Upload” button and a link will be displayed below the buttons, “Download Extension File”
3. Click this link and save the zip file. This is your extension. You can now import this into the Thingworx platform. The imported extension will create a Thing Template with the name of the file plus “JDBCTemplate”. For example a file named “myDbFile.jar” would create a Template named “myDbFileJDBCTemplate”
4. Create a new Thing based on this template and connect to your data source using JDBC

# Configuration Field Descriptions

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| **JDBC Driver Class Name** | Depends on the driver being used. |
| **JDBC Connection String** | Defines the information needed to establish a connection with the database. Connection string formatting can be found at [connectionstrings.com.](http://www.connectionstrings.com/) |
| **ConnectionValidationString** | A simple query that verifies return values from the database (regardless of table names to be executed). |

**Queries and Commands against the Database**

Once you have the configuration set up, in services you can create SQLQueries and SQLCommands to enact upon the database you have connected to.

# Passing Variables

As you build your query, use [[Parameter Name]] for parameters/variables substitution and <<string replacement >> for string substitution.

# Example

DELETE FROM <> WHERE (FieldName = '[[MatchName]]');

DELETE FROM << TableName >> WHERE ( FieldName = [[MatchNumber]]);

 **Note**

It is extremely DANGEROUS to use **<< ... >>** string substitution, because it puts you at risk of SQL Injection. We recommend you use caution when considering this method of parameter passing. However, to create a very dynamic set of queries, you need to pass in the table names as **<< Name of the Table >>**. Also, if you need to use the IN clause, your collection will need to be passed in with **<< Item1, Item2, Item3,>>**

**JDBC Configuration Examples**

# JDBC Configuration for SQLServer

The following table contains the configuration for SQL Server/SQL Server Express

 **Note**

SQL Server must have SQL Authentication enabled. this means you must either be in SQL Authentication or Mixed mode

|  |  |
| --- | --- |
| **Option** | **Value** |
| Connection Test/String Query | SELECT GetDate() |
| JDBC Connection URL | jdbc:sqlserver://<Server>\  <Instance>;databaseName=<Database name> |
| JDBC Driver Class Name | com.microsoft.sqlserver.jdbc.SQLServer Driver |
| Example:  jdbc:sqlserver://10.128.0.169\SQLEXPRESS;databaseName=ThingWorxSample | |

# JDBC Configuration for Teradata

|  |  |
| --- | --- |
| **Option** | **Value** |
| Connection Test/String Query | Select count(\*) from dbc.tables; |
| JDBC Connection URL | Jdbc:Teradata://DatabaseServerName/user= Value,  password=Value,database=Value |
| JDBC Driver Class Name | Com.teradata.jdbc.TeraDriver |
| Example:  jdbc:Teradata://10.128.0.169/user=TWUser;password=TWPass, database=ThingWorxSample | |

**JDBC Connection to IBM DB2 UDB on an AS400**

|  |  |  |
| --- | --- | --- |
| **Option** | | **Value** |
| Connection Test/String Query | | SELECT count(\*) FROM SYSIBM.TABLES; |
| JDBC Connection URL | | jdbc:as400://servername/;Libraries= TESTDTA;  TranslateBinary=true;Prompt= false;TransactionIsolation=none |
| JDBC Driver Class Name | com.ibm.as400.access.AS400JDBCDriver | |
| Example:  jdbc:as400://10.128.0.135/;Libraries=ThingWorxSample;  TranslateBinary=true;Prompt=false;TransactionIsolation=none | | |

3