

# DomSQL

## Domino JDBC Access



### Classic Models, Inc.

701 Gateway Boulevard,  
San Francisco, CA 94107

### Sales Invoice

#### Customer Details

Online Diecast Creations Co.  
Dorothy Young  
2304 Long Airport Avenue

Nashua, NH 62005  
USA

#### Order Details

Customer Number: 363  
Order Number: 10100  
Order Date: 2003-01-06 00:00:00.000  
Ship Date: 2003-01-10 00:00:00.000  
Office: Boston  
Sales Representative: Steve Patterson

Code	Description	Quantity	MSRP (\$)	Discount	Unit Price	Total (\$)
S18_1749	1917 Grand Touring Sedan	30	170.00	0.2	136.00	4,080.00
S18_2248	1911 Ford Town Car	50	60.54	0.09002312520647499	55.09	2,754.50
S18_4409	1932 Alfa Romeo 8C2300 Spider Sport	22	92.03	0.18004998370096714	75.46	1,660.12
S24_3969	1936 Mercedes Benz 500k Roadster	49	41.03	0.1398976358761882	35.29	1,729.21
TOTAL \$						10,223.83

Nov 30, 2011 9:39 AM

# Introduction

The Domino JDBC Access (a.k.a. DomSQL) exposes Domino data as relational tables and provides easy access to the tables using a JDBC driver.

JDBC stands for Java Database Connectivity. It is a set of technologies that are part of the standard Java Runtime environment, used to connect to relational database.

## Release Notes for DomSQL 2014-08-29

This release focussed on refactoring the DomSQL project to work with the latest version of the XPages Extension Library. In v901\_release\_7 of the ExtLib, restructuring work took place in the area of the XPages Relational Support Feature. This has had a knock-on effect, requiring some changes in the com.ibm.xsp.extlib.relational.domsql plugin and in the XPagesDomSQL.nsf application. Due to these changes, there are two important things to note.

**Important Note 1:** From this release (2014-08-29) onwards, the DomSQL project requires a v9.0.1 Domino server (and Domino Designer) that has ExtLib\_v901\_r7 or later installed.

**Important Note 2:** The XPagesDomSQL.nsf demo application requires the XPages RDBMS Enablement project to be installed to the server. This can be downloaded from:

<http://www.openntf.org/main.nsf/project.xsp?r=project/XPages%20RDBMS%20Enablement>

## Objective

This project has been done with the following objectives in mind:

- Expand the query capability of Note/Domino (N/D)  
N/D views already provide a powerful query mechanism, with unique capabilities like categorization, responses... but native N/D views lack some key features like dynamic queries and joins between views. The features Domino views lack are exactly those at which SQL excels
- Reporting – Data analytics  
SQL is the de facto standard used by reporting/data analysis tools. Making N/D data behave like relational data suddenly opens it to a large set of tools that understand JDBC.

As of now, it only does read-only operations on the Domino data. It doesn't allow, for example, relational statements like INSERT, UPDATE or DELETE.

Also, rather than write a full SQL engine for Domino, this project uses another very popular open source project called SQLite (<http://www.sqlite.org/>). SQLite is a well-known, robust engine, well tested and widely used within many projects. Also, SQLite has some very specific characteristics (virtual tables, dynamic typing...) that make the integration with Notes/Domino data easier.

## Installing the software

The DomSQL driver supports two modes for accessing data

- Local access

In this mode, the database engine executes within a Notes/Domino process, which is the HTTP process in case of a Domino server, or the Eclipse process in case of the Notes client. This is the preferred configuration to use when the data is being accessed from XPages, as it doesn't require any network communication between the JDBC client and the DB engine.

Moreover, it automatically executes the database operations on the behalf of the Web user, which is the user authenticated by the XPages runtime.

- Remote access

In this mode, a server process is started and waits for remote connections. This process is executed as a Domino 'tasklet' or DOTS, which is an extension to the Domino server also available from openNTF.

### ***Installing the library for local, XPages access***

***Prerequisite:*** the OpenNTF version of the XPages Extension Library must be installed on both the Domino Designer and the Domino Server. It is required because the relational data access for XPages is part of this extension library.

*The exception is if you only want to use the remote driver. In this case, the XPages runtime is not involved and the XPages Extension Library is not required.*

The library is provided as a regular Eclipse update site. It installs similarly to the Extension Library on either the server or the client.

To install it on the server, you can either:

- Copy the plugins and features contained in the updateSite.zip into your application data/domino/workspace/applications/eclipse directory, and then restart the http task.
- Or you can create an update site NSF (see: [http://www-10.lotus.com/ldd/ddwiki.nsf/dx/XPages\\_Extension\\_Library\\_Deployment](http://www-10.lotus.com/ldd/ddwiki.nsf/dx/XPages_Extension_Library_Deployment)).

For client installation, please refer to the Extension Library documentation as the process is identical.

### ***Installing the library for remote access***

This configuration allows the data to be accessed from an external Java application, like Eclipse Birt or DBVizualizer. These applications don't even have to run on the same machine as the Domino server.

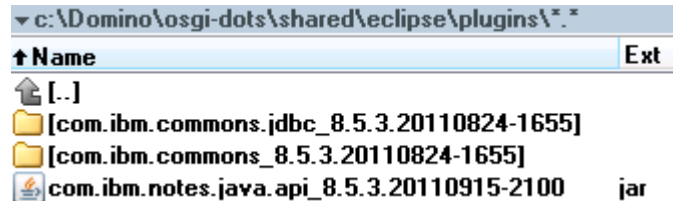
In this case, there are 2 main parts to install:

1. The server process

The server process executes within DOTS, currently available as another openNTF project (<http://www.openntf.org/internal/home.nsf/project.xsp?action=openDocument&name=OSGI%20Tasklet%20Service%20for%20IBM%20Lotus%20Domino>). Note that you need a version published after 12/1/2011, as it contains

some required features.  
Here are the steps then:

1. Install DOTS in your domino server, directory `osgi-dots`
2. Copy a few plug-ins from the HTTP OSGi instance (`<domino>/osgi/...`). The following plug-ins are required to be copied from `osgi-dots`:



Name	Ext
[..]	
[com.ibm.commons.jdbc_8.5.3.20110824-1655]	
[com.ibm.commons_8.5.3.20110824-1655]	
com.ibm.notes.java.api_8.5.3.20110915-2100	jar

3. Copy the plugins into `osgi-dots/shared/eclipse/plugins`

The final directory should look like this:



Name	Ext
[..]	
[com.ibm.commons.jdbc_8.5.3.20110824-1655]	
[com.ibm.commons_8.5.3.20110824-1655]	
com.ibm.domino.domsqldb.driver_1.0.0	jar
com.ibm.domino.domsqldb.sqlite.win32_1.0.0	jar
com.ibm.domino.domsqldb.sqlite.win64_1.0.0	jar
com.ibm.domino.domsqldb.sqlite_1.0.0	jar
com.ibm.dots_2.0.3.201112012333	jar
com.ibm.notes.java.api_8.5.3.20110915-2100	jar
com.ibm.xsp.extlib.relational.domsqldb_1.0.0.DEV	jar

4. Start the DOTS process by typing the following command in the console:  
`load DOTS`

Note that you can also auto start dots by adding it in your `notes.ini` file

`ServerTasks=Replica,...,DOTS`

If the process started properly, you should see a notification in the server console:



```
> load dots
> Initializing DomSQL RMI server for Domino
Starting DomSQL RMI server, port: 8089
DomSQL RMI server started, port: 8089
```

The default port is 8089 but it can be changed through a property in `notes.ini`:

`DomSQL_Port=xxxx`

## 2. The client driver

The client driver is provided in a single jar file. It should be installed in the JVM of the JDBC client. See examples later in this document for DbVizualizer and Eclipse Birt. The driver is provided as a single jar file plugin, that acts as an OSGi plug-in in an OSGi environment, or as a simple jar library in a JVM.

The name of the jar is `com.ibm.domino.domsqldb.sqlite_<version>.jar`

## JDBC URLs

JDBC uses URLs to connect to a database. The general format for DomSQL is the following:

```
jdbc:domsql:[//server:port/]<nsf>/<domsql>[?parameters]  
[//server:port]
```

This is used when connecting to a remote server using the remote driver. It is optional and, if not specified, the driver assumes that it is executed in process, and not remotely.

The port is optional and, if not specified, it is set to the default server value, 8089.

```
<nsf>
```

Name of the NSF database that contains the JDBC source definition (.domsql)

```
<domsql>
```

The name of the DomSQL definition file (.domsql) to use, defined within the NSF (WebContent/WEB-INF/jdbc). An NSF can contain multiple definition files, exposing different tables that are accessible through different URLs.

```
[?parameters]
```

Optional parameters, used to pass authentication parameters when using a remote connection. It uses `user=` for the user name, and `password=` for the password. The user/password pair to be used is the Domino internet user/password.

Here are some sample URLs

- Connecting from an XPages to a local database (in process):  
`jdbc:domsql:MyDB.nsf/All`
- Connecting to a remote database as anonymous:  
`jdbc:domsql://myserver.ibm.com/MyDB.nsf/All`
- Connecting to a remote database as Phil:  
`jdbc:domsql://myserver.ibm.com/MyDB.nsf/All?user=Phil&password=pwd`

## Getting started

### 1- For XPages

This assumes that the plugins are properly installed in the Domino HTTP server, as well as the `XPagesDomSQL.nsf` demo database in the server directory.

You should also be sure to initialize the database by going to the setup page (from the page banner) and create documents in the database.

Use the default parameters and only create the sample documents for now. See the Birt chapter for Birt data.

Finally open a browser, and type the following URL:

<http://localhost/XPagesDomSQL.nsf/DynamicView.xsp>

If everything goes right, then you should be able to select a table and see the data. For example

Connection:

Select a Table:

<u>ID</u>	<u>_O</u>	ID	FIRSTNAME	LASTNAME	EI
303930		CN=Aaron Carney/O=renovations	Aaron	Carney	aa
309862		CN=Aaron Cervantes/O=renovations	Aaron	Cervantes	aa
299286		CN=Aaron Ewing/O=renovations	Aaron	Ewing	aa
304926		CN=Aaron Gibson/O=renovations	Aaron	Gibson	aa

This may look like a regular Domino view, but the data actually comes from an SQL request via JDBC!

The regular XPages JDBC data sources provided by the Extension Library, as well the connections pool, can be used with the DomSQL driver.

The XPages runtime automatically executes the query on the behalf of the web user currently running the page.

## 2- For remote access

First, get the server started and the database filled with data. Then, you can test that the server is up and running by selecting the remote connection from the XPages demo page:

Connection:

Select a Table:

<u>ID</u>	<u>_O</u>	ID	FIRSTNAME	LASTNAME
303930		CN=Aaron Carney/O=renovations	Aaron	Carney
309862		CN=Aaron Cervantes/O=renovations	Aaron	Cervantes
299286		CN=Aaron Ewing/O=renovations	Aaron	Ewing

By default, the demo database allows anonymous access to data. If you changed the ACL, then you'll have to edit the `allremote.jdbc` connection with `WebContent/WEB-INF/jdbc` and add a user/password to the JDBC URL.

# Configuring the JDBC databases

Similar to the Domino REST services, the JDBC access to NSF data has to be configured explicitly. This ensures that no data is exposed unintentionally, and also allows some configuration settings.

This is done through XML files located in the `WebContent/WEB-INF/jdbc` directory, with the extension `.domsql`. The name of the file, without the extension, actually defines the name of the JDBC database.

The content of the XML file is like bellow:

```
<DomSQL [sqliteFile='db file name'] [systemColumns='@xxx[|@yyy]']>
  <Nsf [database='database'] [defaultViews='true/false'] [systemColumns='@xxx[|@yyy]']>
    <View table='sqlTableName' view='viewName' [systemColumns='@xxx[|@yyy]']
[defaultColumns='[col1[|col2...]]' [options='opt[|opt2...]]']>
      [<Column name='dominoColName' sqlName='name' type='int|double|date|text|boolean' />]
    </View>
  </Nsf>
  <Init>
    <Sql>...statement...</Sql>
  </Init>
</DomSQL>
```

The DomSQL runtime creates an actual SQLite database for each `.domsql`, when it is first accessed. By default, it creates a temporary database and lets SQLite choose where to store it, and delete it when the database is closed. But the `sqliteFile` attribute can be used to change the actual file name to use, or `:memory:` means a in memory database.

The DomSQL tag also defines all the tables that will be available to the JDBC connection. The tables can come from the current NSF or any other NSF. In fact, each `<Nsf>` tag could point to a different NSF database. A connection to this DB can query/join/union data coming from different NSFs in a single SQL statement!

When the database name is missing, it means the current database, the one containing the `.domsql` file. The NSF can refer to a local server, or a remote one using the well known `server!!nsf` syntax.

If `defaultViews` is specified, the runtime reads all the views definition from the NSF and automatically exposes all of them as relational tables.

But the views being exposed can also be defined manually using the `<View>` tag. This allows a precise selection of the views to expose and it allows some customization (ex: the name of the relational table).

For all the tables, the engine defines a set of system columns to be exposed as regular relational columns. If nothing is explicitly specified, it only exposes one column: `'@id'`, for the NoteID. But this can be changed, by providing a list of columns separated with a pipe | character like in `'@id|@unid'`.

The possible values are:

`@id,@unid,@class,@siblings,@children,@descendants,@anyunread,@le`

```
vels,@score,@unread,@position, @rowid  
or @all means all of them.
```

The list of the columns to be added to the table can be set using the `defaultColumns` attribute. This attribute should contain the column names, separated by a pipe. If the list is an empty string, the no default column is added. In this case, it solely relies of the `<Column>` children.

Because the view columns in Domino are not typed, DomSQL does not make them typed in SQLite. As a result, the JDBC driver always considers that the columns are of type string in the meta-data (at runtime, it returns a value with its actual type)

Now, to force a column to be of a certain type, the `View` tag has some `Column` child tags. Such tags change an existing column, but they cannot be used to add a new column.

Most of the time setting the actual column type is not required.

The `Column` entry can also be used to set the name of the column in the SQL table, as sometimes the auto-generated programmatic names in Domino might not be user friendly (e.g., `$115...`).

The `options` attributes allows some View specific options, separated by a pipe. The currently available options are the the following ones:

```
SeparateMulti
```

This splits a single view row into multiple rows, based on the number of values in the multiple value fields. Although similar to the Domino native option, it doesn't require this option to be set at the view level, and slightly differs from it:

It only takes into account the multiple fields that are part of the table.

It increments a system pseudo field `_rowid` (`@rowid`) that holds the row id, starting at 0.


All the system columns values are repeated, for all the rows (`_id`, `_unid...`)

See the paragraph on 'handling multiple value items' for more information.

Finally, the `Init` tag can contain SQL statements that will be executed when the database is opened. This allows, for example, a developer to create a set of SQL views based on the tables., or to execute other SQLite statements like pragmas.



# Optimizing the performance

- Ensure that the correct indexes are in place  
The driver can use indexes when evaluating the `WHERE` or `ORDER BY` clauses. Make sure that the view has the right columns being sorted, particularly when joins are involved. When running queries, you can set a flag to display the missing indexes. This should guide you in what is needed. Set the following option from the Debug page:  
 `NATIVE_TRACE_PERFORMANCE_HINTS`
- Avoid using count  
SQLite doesn't have a mechanism to efficiently count rows in virtual tables. When the SQL count function is used, then SQLite reads all the matching records and counts them. This can lead to poor performance for complex or lengthy queries. This is particularly sensitive when you use an XPages JDBC data source. Be sure that you're not counting the actual number of records for the pager.
- Use a dedicated machine for data reporting  
Take advantage of Domino replication. If you are executing complex reports that consume a lot of CPU or disk access, think about having a dedicated machine that replicates the data you need and executes the reports.
- Use an SSD disk  
This is particularly true if your requests read the whole data set many times. An SSD disk can really boost the performance.
- Use SQLite in memory  
Be careful if your data set is too big... Anyway, this can be achieved by setting the `sqliteFile` attribute to `':memory:'`.
- Access the prepared statements metadata  
The SQLite engine requires a query to be executed in order to retrieve the result set metadata. As a result, calling `PreparedStatement.getMetaData()` before calling `execute()` returns invalid results. DomSQL overcomes this limitation by executing the statement only to get the metadata, if a call to `getMetaData()` is performed. This is required by the Birt engine, but this can lead to a performance decrease because it actually executes the query twice.

# Technical Information

## SQLite databases

An SQLite database is created for every `.domsql` file, when it is accessed for the first time, and it is eventually destroyed after a period of inactivity or when the process (HTTP or DOTS) is closed.

SQLite databases can be of different kinds:

- **Temporary**  
This is what DomSQL uses by default. This means that a SQLite file is created in a temporary directory and it will be deleted when the database is closed. It used to hold the temporary tables created by the engine
- **Memory**  
The database definition and the tables are stored in memory. Note that the virtual tables still dynamically read their data from the NSF. Only the temporary tables are stored in memory.
- **File**  
Similarly to a temporary DB, SQLite uses a physical file to store the tables. But, in this case, the database file is not destroyed when the server is closed. As such, it can host some persistent data.  
This mode is still experimental and it is not yet advised to use.

## SQLite virtual tables

Behind the scenes, DomSQL exploits SQLite by exposing the Domino views using virtual tables (<http://www.sqlite.org/vtab.html>). When a database is accessed for the first time, a SQLite database is created and a custom virtual table module is assigned to it. The `.domsql` file is parsed and one SQLite virtual table is created per exposed view, using `CREATE VIRTUAL TABLE` statements. This is completely transparent to the user.

How the virtual module is implemented is beyond this document, but the source code is provided :-)

## System Columns

A domino view entry not only contains the columns defined within the view, but also a set of system values, like the NotesID, UNID, Position... If nothing is specified for a view, then one pseudo column containing the NotesID is created. But a developer can choose to add many of these system columns by declaring them in the `.domsql` file.

Ex:

```
<View view='AllContacts' table='AllContactsSystem' systemColumns='@all'>
</View>
<View view='AllContacts' table='AllContactsUnid' systemColumns='@unid'>
</View>
```

## Column types and SQLite column affinity

Domino columns are not typed, meaning that you don't know in advance what will be the type of the value in a column. More than that, 2 different rows can have values of different types for the exact same column. This is very different from how RDBMS generally works.

Fortunately, SQLite works exactly the same as it uses dynamic value typing. When a SQLite virtual column is created, then a type doesn't have to be specified. But, if specified, it is used by SQLite for automatic conversion, based on type affinity (see: <http://www.sqlite.org/datatype3.html>).

Most of the time, not specifying a type works better with Domino data, except that the JDBC metadata always returns the column type as strings, even though the actual values can be of any type. This might break some applications expecting more precise metadata. To overcome this, the column type can be set in the `.domsql` file, which sets the type at the SQLite level.

Ex:

```
<View view='CUSTOMERS'>
  <Column name='CUSTOMERNUMBER' type='int'></Column>
  <Column name='SALESREPEMPLYEENUMBER' type='int'></Column>
  <Column name='CREDITLIMIT' type='double'></Column>
```

### **Handling dates**

SQLite is missing a true date type and is dealing with dates as either strings or numbers.

To comply with this, DomSQL is returning Domino dates as strings formatted using the ISO8601 specification:

```
YYYY-MM-DD HH:MM:SS.SSS
```

This is compatible with SQLite date format and it is human readable when the string is being displayed.

Moreover, when the JDBC `getObject(int col)` method of a `ResultSet` is called, if the value coming from SQLite is a string of this exact ISO8601 format, then it converts it to an actual `java.sql.Timestamp` object, and returns it instead of the string.

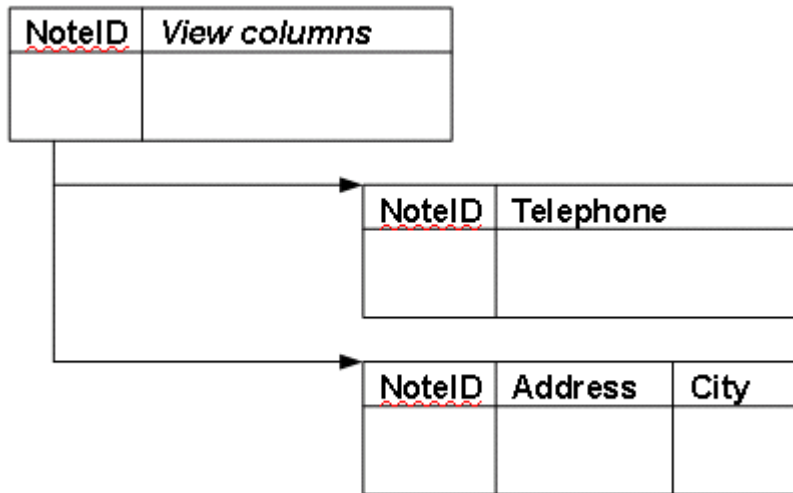
### **Handling multiple value items**

A Domino column can contain multiple value items, which is not supported by SQLite (SQLite doesn't support SQL arrays).

When such a multiple value is encountered, it returns a string containing all the values separated by a pipe '|'. Note that this separator cannot be changed.

Now, the multiple values items can be projected in a separate table, where each value appears on a row, thus allowing relational queries on them.

Let suppose that we have a view, contacts, with 3 multiple value fields. One is unrelated to any other fields (telephone), while the 2 others are related (address & city). The proper relational schema for this is the following:



This can easily be achieved by defining 3 tables, each of them being a projection of the same view, but with different options:

```

<Nsf defaultViews='true'>
  <View view='Contacts' table='MainTable' systemColumns='@id'>
  </View>
  <View view='Contacts' table='Telephones' options='SeparateMulti'
    systemColumns='@id|@rowid' defaultColumns='Telephone'>
  </View>
  <View view='Contacts' table='Addresses' options='SeparateMulti'
    systemColumns='@id|@rowid' defaultColumns='Address|City'>
  </View>
</Nsf>

```

The first table, 'MainTable', projects the view just as is. Note that an optimization would also remove the multiple value fields from this table

The other tables, 'Telephones' & 'Addresses', project the same view but with only the multiple value fields, and with the option 'SeparateMulti'. It also asks for the system columns @rowid, which adds a pseudo integer column holding the row position (0..n),

### Handling data range

Similarly, Domino value ranges are not supported by SQLite. In case of a range, then the 2 values are returned in a single string, separated by a pipe '|'. This separator cannot be changed.

## Notes.ini options

Several options are available in notes.ini

DomSQL\_Port=<int>

Specify the RMI port to use for the remote driver. Defaults to 8089.

DomSQL\_Timeout=<int>

Specify the maximum time, in minutes, that a SQLite database remains open without activity. A value <0 means that it will never time out. It defaults to 10 minutes when not specified or when it is set to 0.



# FAQ

- How does it compare to NotesSQL?
  - NotesSQL is a full implementation including write operations, while DomSQL only allows read only operations.
  - NotesSQL is based on top of ODBC while DomSQL uses JDBC. NotesSQL requires some global configuration in the windows environment while DomSQL doesn't require any configuration.
  - NotesSQL runs on a limited set of server environments while DomSQL is available on both client and server. The C++ layer of DomSQL should be portable to any server platform, as it doesn't require any ODBC library to be available, and SQLite is highly portable.
  - DomSQL is based on a well known database engine, SQLite, whose grammar is already understood by many DB clients (such as report engines...).
  - DomSQL provides an optimized in process data access, tightly integrated with the XPages runtime. This makes it perform very well.  
Also, with the XPages Extension Library relational data sources, it makes it very easy to consume from XPages applications. Security is transparently applied.
  - DomSQL not only exposes the views as is, but allows some fine grained customization of the tables, as well as a selection of the views being exposed. It also allows some SQL statements to be executed when the DB is first accessed, to create actual SQL views for example.
  - DomSQL is provided as a set of OSGi plug-ins and does not require any installation steps beyond installing the plug-ins. Even the native libraries are installed as fragments,, this makes the upgrade process easy.
  - The entire source code for DomSQL is provided. It only uses public Notes/Domino APIs.
- Can I store and access SQLite native data?

Theoretically, yes. But it hasn't been tested and it is currently not a priority to make this robust.

## Known Issues

- The RMI server for remote access doesn't stop properly because RMI is keeping some sockets open. This can probably be solved by providing some custom socket factories in the future

## Current limitations

- It currently only works on Windows, 32 and 64 bits. While most of the JDBC driver and the remote server is written in Java, the SQLite engine is a C library and thus must be compiled for each supported platforms. There are no technical blockers in supporting more platforms, it is just a matter of time, test, and use cases.

- Blob, Clob, SQLXML, Struct and other complex data types are not supported by the driver.

## Possible future enhancements

Bellow is a list of possible future enhancements:

- Performance
  - Broader use of Indexes  
The indexes are currently only used when the operator == is used, or when an order by is requested. This can be extended for some other operators, like >=, <=...
  - Work closely with the NSF team and see what can be optimized using internal functions
  - Decrease the number of calls done by the remote driver to the server. Increase the cache between the two.
- Data Access
  - Document access  
A query can currently only access the columns from a view but the engine can be extended so that the actual document can be opened and its fields accessed. Of course, it will slowdown the query but it can be very useful for some use cases
  - New custom functions  
SQLite allows custom functions to be added to its SQL dialect. So we can imagine having functions matching the native @functions or provide some N/D specific features
  - Full Text Search  
Allow a FT clause to be applied to the view query. Not sure that this will be reflected in the SQL query, but this is something to think about
- Security
  - Support SSL for the remote driver, using custom RMI socket factories

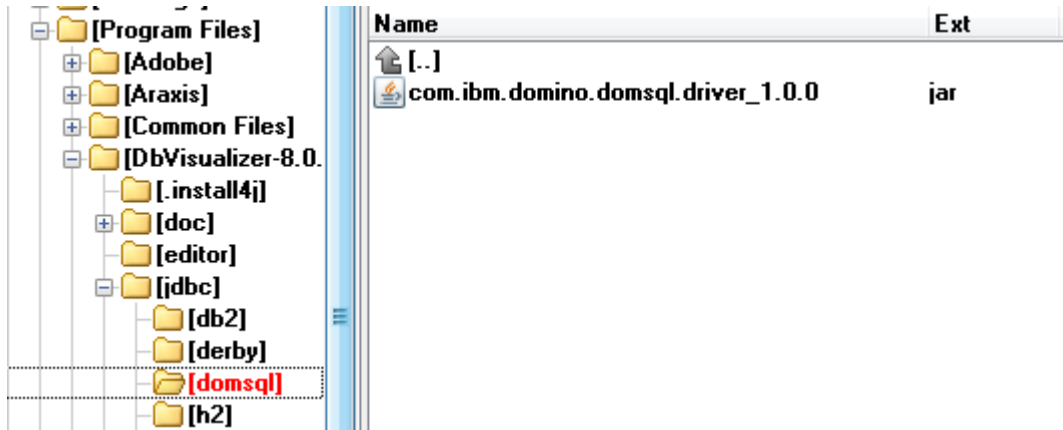
# Known Issues

The RMI server doesn't stop properly and leaves some sockets opened. This prevents the task from being stopped and restarted. If you need to restart this task, then you have to shutdown the entire server. The RMI documentation is very lazy on how this should be done.

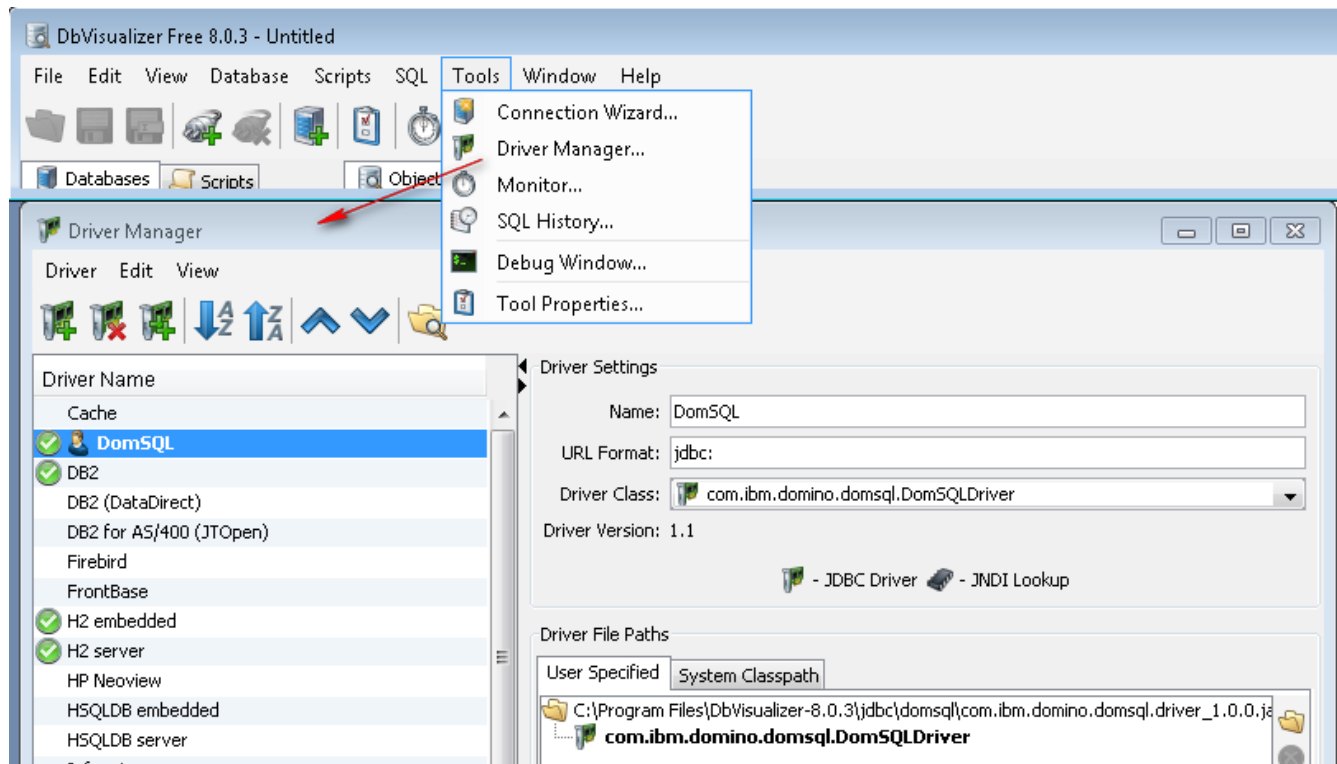


# Configuring DbVisualizer

DbVisualizer is a very easy to use JDBC database explorer. You can download a copy from <http://www.dbvis.com/>. After installing it and in order to connect to a DomSQL database, you need first to copy the DomSQL driver into a place that is accessible to DbVisualizer . I suggest that you create a domsql directory inside DbVisualizer installation directory and copy the driver there:

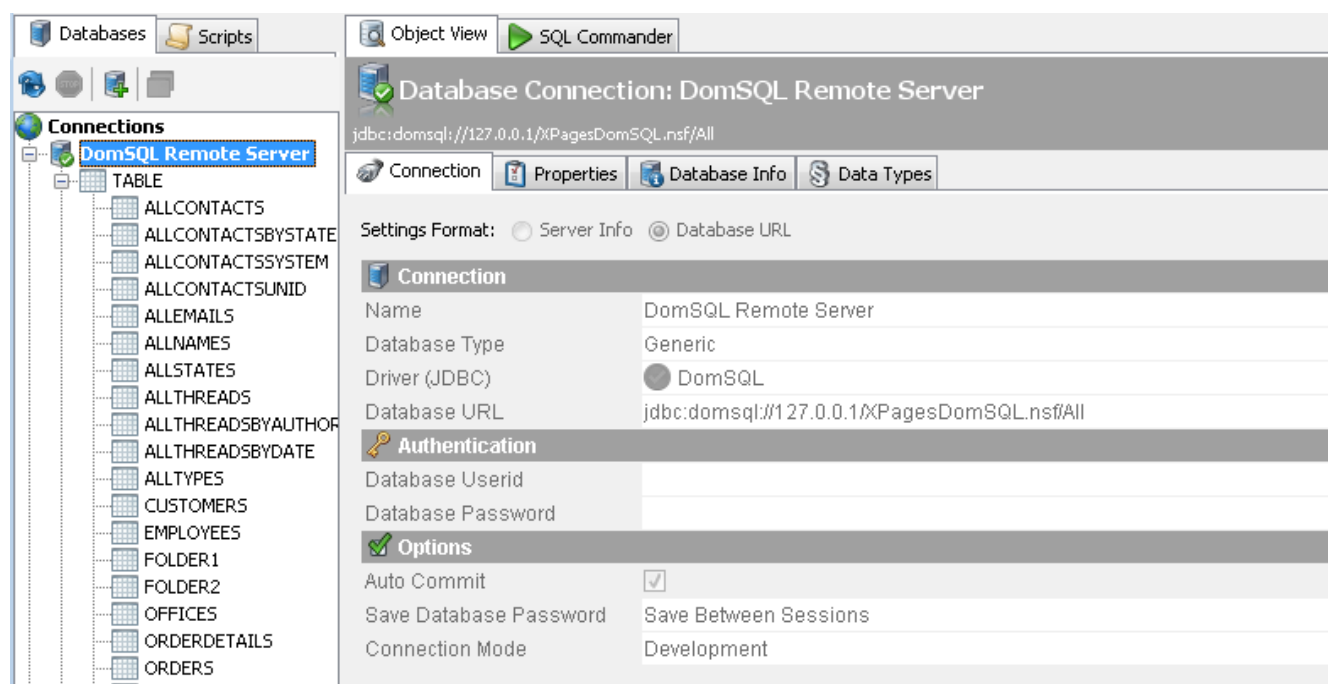


Then, you need to add this driver in the driver configuration manager of DbVisualizer:



Finally, you have to create a new connection in the connection list, pointing to the sample DB.

In the example bellow, we are point to the “All” database from XpagesDomSQL.nsf. Any database can be accessed this way:



- This is it! You should now be able to browse the tables and access their data and metadata, as well as the driver capabilities.

# Configuring Birt

Birt is a very powerful, free report engine you can download from the Eclipse Web Site (<http://www.eclipse.org/birt/phenix/>). DomSQL has been tested using Birt 3.7.1 and its associated sample reports.

Here are the step for using Birt:

## 1. Download and install Birt

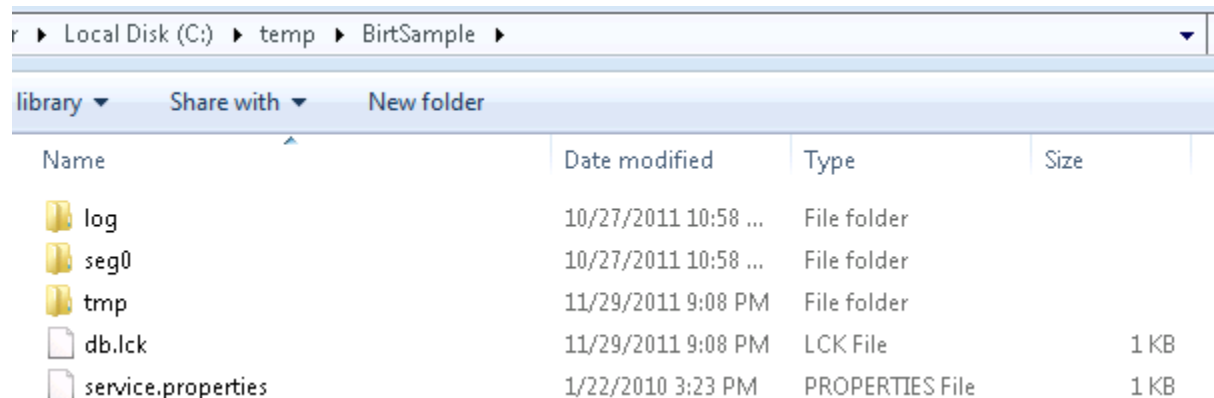
Be sure you get the 3.7.1 version. If you're not familiar with Birt, I strongly advise you to download the "All-In-One" package

## 2. Birt sample demo reports are using a sample relational database, based on Apache Derby. To run the demo on top of Domino data, we need to copy the content of the tables into Domino documents.

The Derby database is provided within an Eclipse plug-in in Birt. Here is where it is located:

```
birt\eclipse\plugins\org.eclipse.birt.report.data.oda.sampledb_3.7.1.v20110905\db\BirtSample.jar
```

You first have to unzip it into a temporary directory, let's say `c:\temp\BirtSample`, as this is how the datasource is configured in the XPages sample app. Your directory should look like this:



If you didn't install it in `c:\temp\BirtSample`, you should go to the demo NSF and edit the `birtderby.jdbc` connection to point to your location.

```
<url>jdbc:derby:c:\temp\BirtSample</url>
```

## 3. The DomSQL Demo NSF comes with a set of views that matches the Birt Derby Demo DB tables, but the NSF doesn't have any data in it. The data have to be imported from the derby demo database from the database setup page.

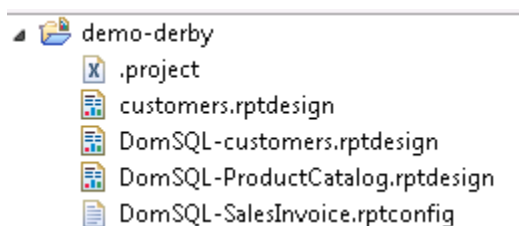
**Create a sample dataset based on the Birt Derby demo database**

☒ Create documents from the Birt sample demo db

If the derby demo DB is properly installed and referenced as explained above, then

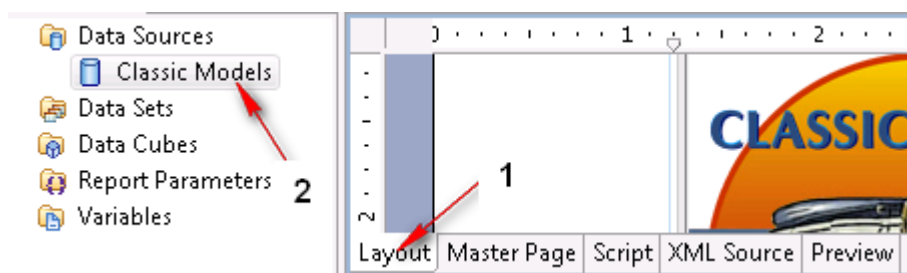
documents will be created in the NSF.

4. From the Birt studio, and the sample report project, make a copy of all the reports and prefix then with “DomSQL-” (or anything else). This can be achieved by simple copy/paste operations in the Eclipse navigator



5. Now you have to change the JDBC data source associated with each report.

Be sure that you have the “Layout” tab selected. Edit the data source from the data view, by double clicking it:



You should first ensure that the DomSQL driver is registered to Birt. Click on the “Manage Driver...” button and add the `com.ibm.domino.domsql.driver_1.0.0` jar file.



Once this is done, you have to change the connection and make it point to your Domino server, as shown below:

**Edit the selected data source.**

Driver Class:

com.ibm.domino.domsql.DomSQLDriver (v1.1)

Database URL:

jdbc:domsql://127.0.0.1/XPagesDomSQL.nsf/Birt

The demo database doesn't have any ACL set, so you can connect as anonymous and thus you don't need a user/password.

6. Finally, the reports have been designed for the Derby database and the tables are accessed using a schema prefix. This is currently not supported by the SQLite database, so you should update the SQL queries by removing the “classic” prefix.

For each report, go to the XML source and remove the prefix. The request:

```
<xml-property name="queryText"><![CDATA[SELECT PRODUCTLINE,  
SUM( PRICEEACH*QUANTITYORDERED ) AS "TOTALREVENUE"  
FROM CLASSICMODELS.ORDERDETAILS, CLASSICMODELS.PRODUCTS  
WHERE CLASSICMODELS.ORDERDETAILS.PRODUCTCODE = CLASSICMODELS.PRODUCTS.PRODUCTCODE  
GROUP BY CLASSICMODELS.PRODUCTS.PRODUCTLINE]]></xml-property>
```

should be changed to:

```
<xml-property name="queryText"><![CDATA[SELECT PRODUCTLINE,  
SUM( PRICEEACH*QUANTITYORDERED ) AS "TOTALREVENUE"  
FROM ORDERDETAILS, PRODUCTS  
WHERE ORDERDETAILS.PRODUCTCODE = PRODUCTS.PRODUCTCODE  
GROUP BY PRODUCTS.PRODUCTLINE]]></xml-property>
```

7. When executing the reports, you should get the exact same result whenever the data comes from the original Derby database or the Domino NSF one
8. Now enjoy it with your own reports :-)