Database Connections

Servoy utilizes a three tier architecture for database access: Clients communicate with the Servoy Application Server and the Application Server communicates with the database.

The Application Server connects to databases through JDBC, a universal Java technology for Java applications to communicate with SQL databases.

For each defined database, the Servoy Application Server manages a pool of database connections, the size of which is configurable, to minimize to overhead of connection creation.

In This Chapter

High Level Overview

The Servoy Application Server connects to databases through JDBC. In order to connect to a database, Servoy requires the right JDBC driver to be placed in the `{servoyInstall}/application_server/drivers` directory, after which a restart of the Application Server is required.

Connections to databases can be configured through the Servoy Admin page and all settings are stored in the servoy.properties file located in `{servoyInstall}/application_server`.

Servoy Clients send their query requests to the Servoy Application Server, which in turn connects to the database to execute the query and return the result to the Client

The Servoy Application Server manages a pool of database connections for each database server, in order to minimize the overhead of creating new connections for each request.

JDBC Drivers

In order to be able to connect to a database, the Servoy Application Server requires the JDBC Driver for the specific database. JDBC drivers usually come with the database or are provided separately by the database vendor or third party vendors.

The JDBC drivers are stored in the `../application_server/drivers/` directory. Servoy ships JDBC drivers for a several databases. For a full overview of the JDBC drivers shipped with a specific Servoy version, check the Servoy Stack info page in the Reference Guide for the Servoy version used. Additional JDBC Drivers can be manually placed in this directory or can be uploaded through the Upload Library section of the Servoy Admin Page. In both scenario's the Application Server requires a restart.

⚠ While one database is more strict than the other, it is important to use the correct version JDBC Driveror a specific version of the database. Notably Oracle is very strict: even using the JDBC driver of another minor version might cause unexpected results! Refer to the documentation of the database vendor for the correct JDBC driver.

> **oraclelobfix.jar**
>
> The *oraclelobfix.jar* file in the `../application_server/drivers` directory is not an Oracle JDBC driver, but a library containing a fix for an issue in the Oracle JDBC driver. Since release 8.3 this file is no longer needed.

JDBC 3.0 vs. JDBC 4.0

There are two active versions of the JDBC specification, namely version 3 and version 4. The version 4 specification is only compatible with Java 6 or higher, whereas version 3 is compatibly with Java 5 and higher. As Servoy is compatibly with Java 5 and higher, Servoy supports JDBC drivers that conform to the JDBC 3.0 specification.

As of Servoy 5.2.9 Servoy will also support JDBC drivers that conform to the JDBC 4.0 specification.

⚠ Note that JDBC 3.0 and JDBC 4.0 should not be mistaken for JDBC type 3 or 4: JDBC types (1 through 4) are an indication how the communication between the Java process and the database is implemented, whereas JDBC 3.0 or JDBC 4.0 says something about the Java API exposed by the JDBC driver itself.

Connecting to Databases

Connections to databases can be made from the Database Server page of the Servoy Admin page. The JDBC driver for the database to which the connection needs to be made has to be loaded into the Servoy Application Server already, see the JDBC Drivers paragraph above.

In order to connect to a database, the following information is required:

Setting	Property (default UDM sample database setup)	Description	Comment
Database Server Name	server.{index}.serverName=udm	The name by which the database is referenced in Solutions	Solution are designed against this name thus this name is referenced in the design of solutions
Username	server.{index}.userName=DBA	The database username that needs to be used for the connection	
Password	server.{index}.password=	The password that goes with the database username	
URL	server.{index}.URL=jdbc: postgresql://localhost:5432/udm	The JDBC URL through which the database can be accessed	Refer to the database and/or JDBC driver documentation for the URL syntax
Driver	server.{index}.driver=org.postgresql. Driver	The JDBC Driver classname	Refer to the database and/or JDBC driver documentation for the classname to use
Catalog	server.{index}.catalog=<none>	The specific catalog to connect to	Not all databases support this option*
Schema	server.{index}.schema=<none>	The specific schema to connect to	not all databases support this option*

* Catalog & Schema: JDBC defines that a database may have a set of catalog and each catalog may have a set of schema's. However, each database/JDBC driver vendor has interpreted this differently. In general a Catalog contains all the system/metadata tables/views, while the schema contains all the "user" defined tables, views, triggers etc. Within the context of Servoy, the Catalog is hardly used, while the schema setting is used when connection to Oracle

Besides the above mentioned settings that deal with how to connect to the database, a database Server definition within Servoy has a few additional settings:

Setting	Property (default UDM sample database setup)	What is does	Comment
Maximum prepared statements idle	server.{index}. maxPreparedStatementsIdle=100	All Servoy generated SQL statements are in the form of Prepared statements, to increase the performance of statement execution. This setting determines how many prepared statements are kept in cache.	While exposed in Servoy, tuning this setting requires indepth insight of the JDBC driver used. As such, this setting is not further documented. Refer to the JDBC driver documentation for more information.

Query validation type	server.{index}.connectionValidationType=0	Some databases automatically end connections when they have been idle for a certain period of time. This setting controls if and how Servoy validates a connection leased from the connection pool, before using it. There are three variations: • *exception validation (value = 0, default)*: With exception validation no validation occurs before using the connection. When a exception occurs, the connection is destroyed. While this method has no overhead, the downside of this method is that the user is presented with the exception. • *query validation (value = 1):* With query validation, each connection is validated by executing a validation query upon leasing the idle connection from the pool. If the execution of the validation query results in an exception, the connection is destroyed and a new connection is leased from the pool. This process continues, until a connection correctly handles the validation query. The actual validation query can be set using the 'Validation query' setting (see below). This method has the greatest overhead of the three options, because the validation query is fired for every request towards the database. • *metadata validation (value = 2):* With meta data validation the JDBC driver is asked for some metadata about the connection. This method is not as useful on all databases, as some JDBC driver cache the meta data, so they return their result without actual communication with the database itself In most scenario's the exception validation will be sufficient, as the Servoy Application Server and the Database Server are connected via a reliable network connection (or are hosted on the same machine). Instead of opting in for any of the other validation types, it is advised to solve any connection issues between the Application Server and the Database Server, instead of changing the validation type. Changing the validation type should be the last resort. • *driver based validation (value= 3, since 8.4)*: With driver based validation, the connection validation mechanism of the JDBC driver is used; note that this method may send a query to the database server, depending on the driver implementation. In case of an old (JDBC3) driver the validation will fall back to exception validation.	MySQL is notorious for invalidating existing connections after little idle time. While setting 'Query validation' as the validation type will solve the issue in most cases, it's better to configure MySQL to not invalidate connections or not invalidate them as fast
Validation query	server.{index}.validationQuery=	The SQL statement fired at the database if the 'Query validation type' is set to 'Query validation'. The SQL statement used should be a statement with as little overhead as possible, for example an efficient query such as 'SELECT 1'. While this statement is valid SQL92, it does not work on all databases through	

Data model cloned from	server.{index}.dataModelCloneFrom=	This setting allows marking a Database Server as a clone of another Database Server. When marked as such, if a Solution is imported on the Servoy Application Server, any updates to the datamodel of the master Database Server are also applied to the Database Servers that are marked as a clone of the master Database Server.	Solutions in Servoy are designed against a named Database Server. The actual Database Server against which a Solution runs can be switched at runtime from within the solution's code. When a Solution is imported on a Servoy Application Server, it will automatically update the datamodel in the database to match the datamodel against which the Solution was designed. In case the functionality to switch the Database Server against which the Solution runs is used, it's essential that the datamodel updates are done on all the 'clones' of the master database against which the Solution was designed. This setting can be used to identify a database as a clone of another database. When marked as such, if a solution gets imported that updates the datamodel in the master database, the datamodel in the clone will also be updated.
Enabled	server.{index}.enabled=true	Whether or not the database is enabled.	
Skip System Tables	server.{index}.skipSysTables=false	Whether or not System Tables and Views from the database are to be exposed in Servoy.	
Log server	servoy.log_server=	Servoy has functionality that allows to automatically track all insert /updates/deletes on tables. This functionality can be enabled through the Security layer inside the Solution. This functionality relies on one of the enabled Database Servers configured on the Servoy Application Server being marked at 'Log server'. This setting defines if the Database Server is the Log server.	
Enable stored procedures	server.{index}.queryProcedures=false	Whether server should expose stored procedures from the database. Default value is false except for Progress database where value is true. If set to true, will show stored procedures in solution explorer view and in code completion under datasources. From datasources node can call a stored procedure.	
Maximum number of values for IN-clauses in SQL queries	server.{index}.selectINValueCountLimit=200	When Servoy needs to query the database using an IN-clause, it will use a temporary table if the number of values is above a certain limit. This limit can be configured per database server, or globally (setting servoy.selectINValueCountLimit). Tthe default limit is 200.	Since release 8.3.2. Note that when this is set to a too large value, queries may fail because the driver may not accept such large queries with so many parameters.

All settings for the Database Servers are stored in `{serverInstall}/application_server/servoy.properties`. For example the connection defined for the UDM sample database:

```
server.4.URL=jdbc\:postgresql\://localhost\:5432/udm
server.4.catalog=
server.4.driver=org.postgresql.Driver
server.4.password=encrypted\:XAFg2JKIdj0\=
server.4.schema=
server.4.serverName=udm
server.4.userName=DBA
```

Note that properties that have not been set with a different value as the default will not be stored in servoy.properties.

Manual Configuration

While the Servoy Admin page provides a User interface to create an manage Database Server configurations, it is possible to manually configure the database connection in the `{serverInstall}/application_server/servoy.properties` file.

When manually adding or removing database server definitions in the servoy.properties file, make sure to update the `ServerManager.numberOfServers` property afterwards to reflect the actual number of servers defined. Note that the server definitions start at index 0 and should be sequentially numbered.

⚠ When manually editing the servoy.properties file, make sure the Servoy Application Server is shut down first, as the Servoy Application Server will update the servoy.properties file on shutdown/restart, thus overwriting any changes make directly to the file.

Connection Pooling

Clients do not directly access the databases, instead all their query requests are send to the Application Server which then delegates the query to the correct database.

In order to minimize the overhead of connection creation, the Application Server manages a pool of database connections per configured database server. On each database server, there are several settings related to the pool's behavior. These settings are available through the settings for each individual database on the Database Servers page on the Servoy Admin page:

Setting	Property	What is does	Comment
Maximum connections active	server.{index}. maxConnectionsActive=30	Determines the maximum number of connections that will be made to the database simultaneous	If set too low, handling requests towards the database might slow down, as one request needs to wait until another request is processed and the connection is returned to the pool If set too high, the exceptions might occur if the database cannot handle that many concurrent connections or if more memory is required than is available
Maximum connections idle	server.{index}. maxConnectionsIdle=10	Determines the maximum number of unused connections that are in the pool	Active connections that are done processing a request are returned to the connection pool as idle connections. If the number of idle connections goes over the maximum, the connections are removed. As instantiating new connections takes time, the value shouldn't be too low. On the other side idle connections take up resources, so the number shouldn't be too high either. Must be lower that the 'Maximum connections active' setting

Each connection to a database consumes memory and resources both on the Servoy Application Server as well as on the database server/engine side, easily adding up to several Mb of memory usage on both the Application as well as the Database server side. Instantiating a new connection takes time. Thus the two settings above must be balanced to provide the optimal performance, while not consuming too many resources.

When a request from a client needs to be handled, an idle connection is leased from the pool. If there is no idle connection left a new connection will be created, if the 'Maximum connections active' value has not been reached. When the request is finished, the connection is returned to the pool. If the Maximum connections idle' value is exceeded, the returned connection is destroyed.

If the 'Maximum connections active' value is reached, no new connections can be leased from the pool. In this case, the request from the client is on hold, until a connection is released to the pool. When this happens, the user will experience a hanging client, until a connection becomes available again and the request can be handled.

The handling of a single request from a client usually takes only a few milliseconds (see the performance log on the Servoy Admin page for details on query execution times). However, if a Client is using transactions, the connection is leased from the pool for the duration of the transaction. When running solutions that use long running transactions, the connection pool settings need to be adjusted accordingly.

The Database Servers page on the Servoy Admin page shows per database the number of Active and idle connections, compared to their respective maximum value

Dimensioning the Connection Pool

By default, the maximum active connections setting is set to 10. This could be too low when serving may clients from one Application Server or when the clients do many requests to the database or use long running transactions. As rule of thumb, if the actual used active connection regularly goes above 70% of the maximum a higher number of maximum active connections should be configured.

Database Limitations

The maximum number of active connections is also limited by the maximum number of connections the database itself is configured to allow.

For the bundled PostgreSQL database engine for example, the maximum is 100 connections. However, these 100 connections are for all connections made to the PostgreSQL database server instance. This means that if there are multiple Database Servers defined in the Servoy Application Server which are all hosted on the same PostgreSQL database server instance, the max. 100 connections are for all Database Servers combined. This must be taken into account when setting up the maximun number of active connections.

Update Servoy Sequences

Servoy supports an internal Primary Key sequence generator mechanism, in addition to database managed PK sequence generators. If Servoy Sequences are used within a Solution and either Servoy is connected to an existing database with data for the first time, or while Servoy was connected to the database and external processes inserted new records in the tables for which Servoy manages the sequences when Solutions insert records into those tables, the internal counter will be out of sync with the actual records in the database. The option to update the Servoy Sequences will look at the max value in each table and update the internal counter inside Servoy, bringing them in sync again.

When records are inserted through Servoy while the internal Servoy sequence counter is out of sync with the actual data in the database, exceptions can occur.