Table of Contents

DBAccess supports connection pooling by the interface com.sibvisions.rad.persist.jdbc.IConnectionPool.

DBAccess gets and releases connections from the pool with

```
public Connection getConnection() throws SQLException;
public void releaseConnection(Connection pConnection);
```

DBAccess has following methods for handling connection pooling:

```
public void releaseConnection() throws DataSourceException

public boolean isConnectionPoolEnabled()
public void setConnectionPoolEnabled(boolean pConnectionPoolEnabled)

public EventHandler<IConfigureConnectionListener> eventConfigureConnection()
public EventHandler<IUnconfigureConnectionListener>
eventUnconfigureConnection()
```

The method releaseConnection is for manual releasing connections. If there is an open transaction, releaseConnection throws a DataSourceException with message “Connection is modified and cannot be released!”

By default, connection pooling is disabled which means that a connection is created on open, and released on close of DBAccess.

If connection pooling is enabled and autocommit enabled, a connection is created on demand where needed and is automatically released when no transaction is open after last call. This check will be queued via invokeFinally (see Server-side Call events).

If connection pooling is enabled and autocommit disabled, the connection will be released after each request as long as no transaction was started. If a transaction was started, the connection will be released with next commit or rollback call.

To be able to configure custom connection settings, use eventConfigureConnection and eventUnconfigureConnection. Configure connection is especially important because it's not guaranteed that the same connection will be used for different calls. If you need custom settings, use events to configure the connection. The configure event will be triggered automatically by open and getConnection and unconfigure will be triggered by close and releaseConnection.

If you don't use connection pooling, you can configure your connection without events because there will be exactly one database connection per logged-in user. We recommend using event for connection configuration!

Configuring Connection Pool

Using an application server, it's possible to configure a datasource resource. The JNDI name can be configured in your applications config.xml:

```
<datasource>
  <db name="default">
```

```
    <url>jdbc/dbname</url>
  </db>
</datasource>
```

The DBAccess is created in Session life-cycle object:

```
protected DBAccess getDBAccess() throws Exception
{
    DBAccess dba = (DBAccess)get("dBAccess");

    if (dba == null)
    {
        DBCredentials dbcred = DBSecurityManager.getCredentials(
SessionContext.getCurrentSessionConfig());

        dba = DBAccess.getDBAccess(dbcred);
        // enables connection pooling
        dba.setConnectionPoolEnabled(true);
        dba.open();

        put("dBAccess", dba);
    }

    return dba;
}
```

It is also possible to create the connection pool with standard JDBC configuration.

```
<datasource>
  <db name="default">
    <url>jdbc:oracle:thin:@localhost:1521:XE</url>
    <username>test</username>
    <password>test</password>
  </db>
</datasource>
```

In this case, the datasource has to be created manually, e.g. in application life cycle object:

```
protected DataSource getDataSource()
{
    DataSource dataSource = (DataSource)get("dataSource");

    if (dataSource == null)
    {
        DBCredentials dbcred = DBSecurityManager.getCredentials(
SessionContext.getCurrentSessionConfig());

        dataSource = new OracleDataSource();
        dataSource.setURL(dbcred.getUrl());
        dataSource.setUser(dbcred.getUserName());
        dataSource.setPassword(dbcred.getPassword());
```

```
        put("dataSource", dataSource);
    }

    return dataSource;
}
```

and in the session life cycle object:

```
protected DBAccess getDBAccess() throws exception
{
    DBAccess dba = (DBAccess)get("dBAccess");

    if (dba == null)
    {
        dba = DBAccess.getDBAccess(new
DataSourceConnectionPool(getDataSource()));
        // enables connection pooling
        dba.setConnectionPoolEnabled(true);
        dba.open();

        put("dBAccess", dba);
    }

    return dba;
}
```

Configuring Stateful Database Connections

It is possible to configure every connection taken from the connection pool. In Oracle, for example, it's possible to store user information in packages.

```
public DBAccess getDBAccess() throws Exception
{
    DBAccess dba = (DBAccess)get("dBAccess");

    if (dba == null)
    {
        DBCredentials dbcred = DBSecurityManager.getCredentials(
SessionContext.getCurrentSessionConfig());

        dba = DBAccess.getDBAccess(dbcred);
        // enables connection pooling
        dba.setConnectionPoolEnabled(true);
        dba.eventConfigureConnection().addListener(new
IConfigureConnectionListener()
        {
            public void configureConnection(ConnectionEvent pEvent) throws
SQLException
```

```
            {
                // sets user for database internal usage, e.g. logging
                ISession session = SessionContext.getCurrentSession();

                pEvent.getDBAccess().executeProcedure("REGISTRY.login",
session.getUserName());
            }
        });
        dba.open();

        put("dBAccess", dba);
    }

    return dba;
}
```