# Simple database access sample application

{scrollbar}

top

This article guides you through the JDBC features in the Apache Geronimo appplication server. To demonstrate the JDBC features, we use a simple Inventory application which has JSP, Servlets to handle web related features and inbuilt Derby as database.

Inventory Application will use the Service Provider Interface(SPI) method to access it's database. In this method the application uses a JDBC DataSource interface to establish connections with the database. This is the preferred access method for a J2EE application for several reasons:

- Program code will be totally database independent. Driver information, database location, and configuration parameters are stored in the J2EE Server.
- It allows the use of connection pooling. The J2EE Server connection manager effectively manages connections to greatly improve performance and scalability.
- It enables the database to be used by Enterprise JavaBeans (EJB) to implement business logic as part of the J2EE Server. Implementing an EJB tier, though not required, lays the foundation for creating a highly scalable, distributed application architecture.

After reading this article you should be able get the best out of the JDBC features of Geronimo, such as defining database pools and using DataSources to access databases.

This article is organized in to following sections.



## Overview of JDBC Features overview

JDBC implementation in application servers vary from application server to other. Following table gives a feature list of JDBC in Apache Geronimo.

| Feature | Description |
|---|---|
| JDBC access | Geronimo does not have any direct integration with JDBC but supports access through the generic J2CA framework. The TranQL project has J2CA adapters for various databases. |
| JCA implementation | Geronimo supports the JCA 1.5 specification and is backward compatible to the JCA 1.0 specification. |
| Data sources supported | TranQL has generic wrappers for each data sources. |
| Data source failover | TranQL has specialized drivers for certain databases (including Apache Derby, Oracle and DB2) that provide a tighter integration with the advanced features of the driver.<br>It is at this level that features such as load-balancing and failover would be provided. You can also use a C-JDBC wrapper for providing database clustering and failover. |
| XA support | Supports XA transactions, Local Transactions, and No transaction. |
| Connection Manager Configurability | The J2CA framework is interceptor based which allows different parts of the connection framework to be plugged in. |
| JTA implementation | Transaction support is provided through Geronimo Specific Transaction Managing Framework and HOWL. |
| Connection pooling and management | Custom Geronimo Code and TranQL used for connection pooling and management. |
| Legacy driver support | Geronimo provides this through the TranQL- connector JDBC to JCA wrapper in Geronimo. Supports JDBC 3.0 and 2.1. |

## Application Overview application

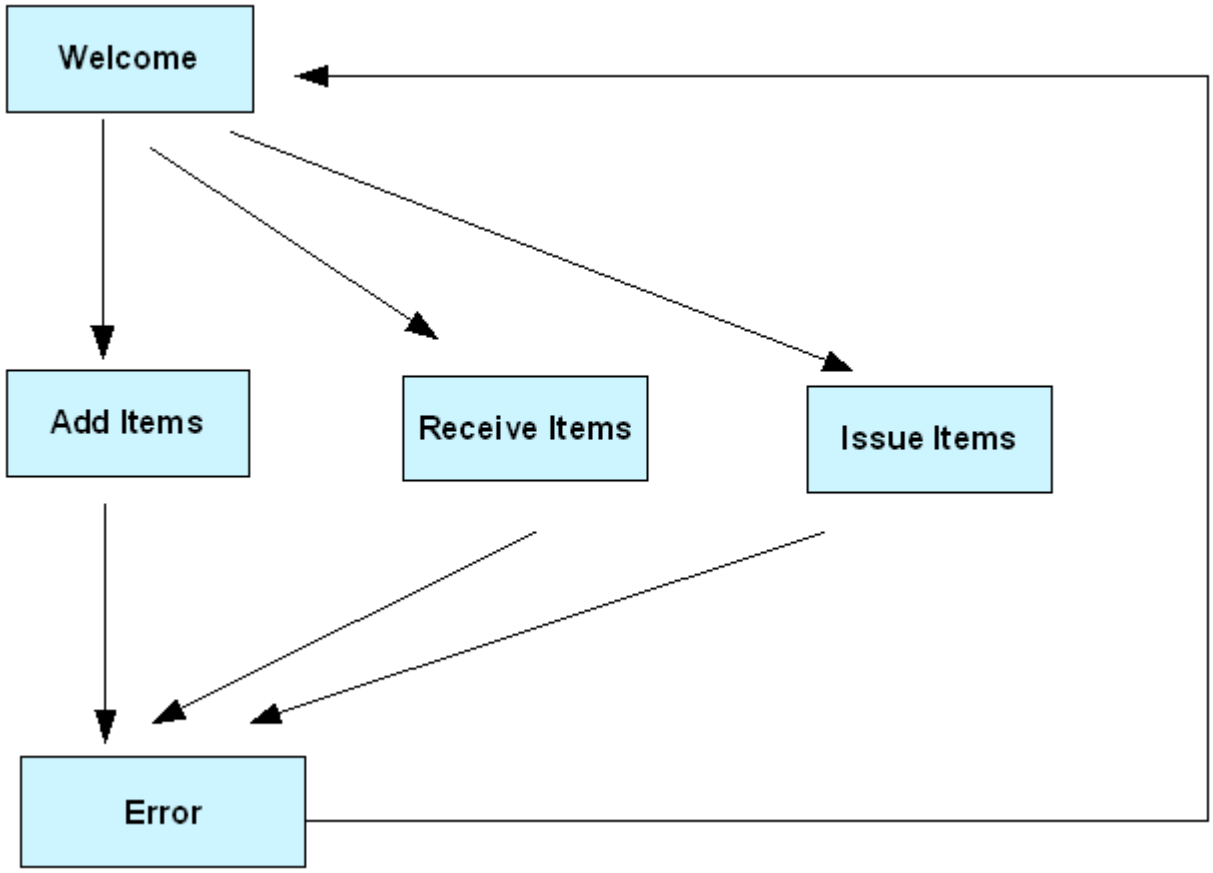
The Inventory application in this article only supports three basic usecases of such applications.

1. Add Items to the Stock
2. Recieve Items
3. Issue Items
   The application workflow statrs with adding item information to the stock. Then it allows enter goods recieving and issuing information. All those updated information are stored in the inbuilt Derby database.

The Inventory Web Application has following list of pages

- Welcome
- Add Item
- Recieve Goods
- Issue Goods

The following figure illustrates the application flow.




Welcome page of the application acting as a notice board which displays current stock of each item. Through the Welcome page users can access Add Item, Recieve Goods or Issue Goods Pages. Upon successful completion of each activity, the page will be redirected back to the Welcome page with updated stock information. Add Item helps to define items in the stock, then 0 number of items will be added to the stock. Recieve and Issue Goods pages represent Goods Recieving and Issueing activities of the application respectively.

## Application contents

The Inventory application consist of following list of packages and classes.

- org.apache.geronimo.samples.inventory
  - Item - represents Item in the Inventoy.
- org.apache.geronimo.samples.inventory.services
  - InventoryManager - represents list of services offered by the inventory.
- org.apache.geronimo.samples.inventory.dao
  - ItemDAO - contains all database access methods.
- org.apache.geronimo.samples.inventory.exception
  - DuplicateItemIdException - custom exception to handle duplication item id scenario.
  - NotSufficientQuantityException - Custom exception to handle not sufficient quantity situation.
- org.apache.geronimo.samples.inventory.util
  - DBManager - handle database related activities such as issueing database connections.
- org.apache.geronimo.samples.inventory.web
  - AddItemServlet - dispatch add item information to service layer.
  - IssueingServlet - dispatch issueing items information to service layer.
  - RecievingServlet - dispatch receving items information to service layer.

The list of web application files in the application is depicted in the following.

java |- jsp |- add.jsp |- error.jsp |- issue.jsp |- recv.jsp |- WEB-INF |- geronimo-web.xml |- web.xml |- welcome.jsp

Application defines a datasource with the help of **geronimo-web.xml** and **web.xml** files. **geronimo-web.xml** add a link between the database pool already deployed in the server. It referes database pool via it's **artifactId**.

xmlsolidgeronimo-web.xml <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"> <environment> <moduleId> <artifactId>InventoryApp</artifactId> </moduleId> <dependencies> <dependency> <groupId>console.dbpool</groupId>

<artifactId>InventoryPool</artifactId> </dependency> </dependencies> </environment> <context-root>/inventory</context-root> <!-- define a reference name to the db pool--> <resource-ref> <ref-name>jdbc/InventoryDS</ref-name> <resource-link>InventoryPool</resource-link> </resource-ref> </web-app>

Following is the **web.xml** of the Inventory application. It uses same name as in the **geronimo-web.xml**, which is used to create the datasource.

xmlsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001 /XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4"> <welcome-file-list> <welcome-file>welcome.jsp</welcome-file> </welcome-file-list> <servlet> <display-name>AddItemServlet</display-name> <servlet-name>AddItemServlet</servlet-name> <servlet-class>org.apache.geronimo.samples.inventory.web.AddItemServlet</servlet-class> </servlet> <servlet> <display-name>IssueingServlet</display-name> <servlet-name>IssueingServlet</servlet-name> <servlet-class>org.apache.geronimo.samples.inventory. web.IssueingServlet</servlet-class> </servlet> <servlet> <display-name>RecievingServlet</display-name> <servlet-name>RecievingServlet</servlet-name> <servlet-class>org.apache.geronimo.samples.inventory.web.RecievingServlet</servlet-class> </servlet> <servlet-mapping> <servlet-name>AddItemServlet</servlet-name> <url-pattern>/add_item</url-pattern> </servlet-mapping> <servlet-mapping> <servlet-name>IssueingServlet< /servlet-name> <url-pattern>/issue</url-pattern> </servlet-mapping> <servlet-mapping> <servlet-name>RecievingServlet</servlet-name> <url-pattern> /recv</url-pattern> </servlet-mapping> <!-- reference name exposed as a datasource --> <resource-ref> <res-ref-name>jdbc/InventoryDS</res-ref-name> <res-type>javax.sql.DataSource</res-type> <res-auth>Container</res-auth> <res-sharing-scope>Shareable</res-sharing-scope> </resource-ref> </web-app>

Next important phase of the application is accessing defined datasource from the source code. This part is handled by the **DBManager** class.

javasolidDBManager.java public static Connection getConnection(){ Connection con = null; try { Context context = new InitialContext(); DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/InventoryDS"); con = ds.getConnection(); } catch (NamingException e) { e.printStackTrace(); } catch (SQLException e) { e.printStackTrace(); } return con; }

### Sample Database

The sample database that is being used to demonstrate this application is in-built Derby database. The name of the sample database is **InventoryDB** and it consists of two tables, namely ITEM and ITEM_MASTER. The fields for each of these tables are described below.

| Table Name | Fields |
|---|---|
| ITEM | ITEM_ID (PRIMARY KEY)<br>ITEM_NAME<br>DESCRIPTION |
| ITEM_MASTER | ITEM_ID (PRIMARY KEY)<br>QUANTITY |

The ITEM table stores the data related to the items while ITEM_MASTER stores the quantity in hand of each item.

## Tools used

The tools used for developing and building the Inventory sample application are:

### Apache Derby

Apache Derby, an Apache DB subproject, is a relational database implemented in Java. Its footprint is so small you can easily embed it in any Java-based solution. In addition to its embedded framework, Derby supports a more familiar client/server framework with the Derby Network Server.
http://db.apache.org/derby/index.html

### Eclipse

The Eclipse IDE was used for development of the sample application. This is a very powerful and popular open source development tool. It has integration plug-ins for the Geronimo too. Eclipse can be downloaded from the following URL:
http://www.eclipse.org

### Apache Ant

Ant is a pure Java build tool. It is used for building the war files for the Inventory application. Ant can be downloaded from the following URL:
http://ant.apache.org

Back to Top

# Configuring, Building and Deploying the Sample Application configure

Download the Inventory application from the following link:
Inventory

After decompressing the given file, the inventory directory is created.

## Configuring

Configuration of the application consists of creating the database and defining the connection pool to access it.

### Creating and Populating Database

After starting Apache Geronimo log into the console and follow the given steps to create the **InventoryDB** database.

solidInventoryDB.sql CREATE TABLE item( item_id VARCHAR(10) PRIMARY KEY, item_name VARCHAR(25), description VARCHAR(100) ); CREATE TABLE item_master( item_id VARCHAR(10) PRIMARY KEY, quantity INTEGER ); INSERT INTO item VALUES('001', 'Item 1', 'Test Item 1'); INSERT INTO item VALUES('002', 'Item 2', 'Test Item 2'); INSERT INTO item VALUES('003', 'Item 3', 'Test Item 3'); INSERT INTO item VALUES('004', 'Item 4', 'Test Item 4'); INSERT INTO item_master VALUES('001', 12); INSERT INTO item_master VALUES('002', 8); INSERT INTO item_master VALUES('003', 49); INSERT INTO item_master VALUES('004', 34);

1. Select **DB Manager** link from the **Console Navigation** panel on the left.
2. Give the database name as **InventoryDB** and click **Create** button.
3. Select **InventoryDB** to the **Use DB** field.
4. Open **InventoryDB.sql** in the **inventory/config** directory from a text editor.
5. Paste the content **InventoryDB.sql** to the **SQL Commands** text area and press **Run SQL** button.

### Deploying Database Connection Pool Plan

The application server going to access **InventoryDB** through a connection pool. Following are the list of steps to create the pool.

xmlsolidInventoryPool.xml <?xml version="1.0" encoding="UTF-8"?> <connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.1"> <dep: environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1"> <dep:moduleId> <dep:groupId>console.dbpool</dep:groupId> <dep: artifactId>InventoryPool</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>rar</dep:type> </dep:moduleId> <dep:dependencies> <dep: dependency> <dep:groupId>geronimo</dep:groupId> <dep:artifactId>system-database</dep:artifactId> <dep:type>car</dep:type> </dep:dependency> < /dep:dependencies> </dep:environment> <resourceadapter> <outbound-resourceadapter> <connection-definition> <connectionfactory-interface>javax.sql. DataSource</connectionfactory-interface> <connectiondefinition-instance> <name>InventoryPool</name> <config-property-setting name="Driver">org. apache.derby.jdbc.EmbeddedDriver</config-property-setting> <config-property-setting name="UserName">app</config-property-setting> <config-property-setting name="ConnectionURL">jdbc:derby:InventoryDB</config-property-setting> <connectionmanager> <local-transaction/> <single-pool> <max-size>10</max-size> <min-size>0</min-size> <match-one/> </single-pool> </connectionmanager> </connectiondefinition-instance> </connection-definition> </outbound-resourceadapter> </resourceadapter> </connector>

1. Click on **Deploy New** from the **Console Navigation**.
2. Load the **tranql-connector-1.2.rar** to the **Archive** input box from the **<geronimo_home>/repository/tranql/tranql-connector/1.2/tranql-connector-1.2.rar** location.
3. Load the **InventoryPool.xml** to the **Plan** input box from **inventory/config** directory.
4. Press **Install** button to deploy connection pool in to the server.

Upon successful deployment **InventoryPool** will appear on the **Database Pools** list of the Geronimo Console.

## Building

Inventory application comes with an Ant script to help users to build from source code. Use the command prompt to navigate into the **inventory** directory and just give **ant** command to build. It will create the **Inventory.war** file under the **releases** folder within **inventory**. Now, you are ready to deploy Inventory web application in to the Geronimo Application server.

## Deploying

Deploying sample application is pretty straight forward as we are going to use the Geronimo Console.

1. Click the **Deploy New** link on the **Console Navigation** panel.
2. Load **Inventory.war** file from **inventory/releases** folder into the **Archive** input box.
3. Press **Install** button to deploy the application in the server.

Back to Top

# Testing of the Sample Application testing

To test the sample application, open a browser and type http://localhost:8080/inventory. The Welcome page of Inventory application which is acts as a notice board will be loaded.

The user can directly access Add Items, Recieve Goods and Issue Goods functionalities from the Welcome page.

# Summary summary

This article has shown you how to use JDBC features inside the Geronimo Application Server. You followed step-by-step instructions to build, deploy and test a sample application to elaborate these features.

The highlights of this article are: -

- JDBC features in Apache Geronimo.
- Create a database and populate the data in Geronimo with in built Derby Database.
- Deploy a database pool plan to access a database.
- Deploy web archives to access database via the pool defined in Geronimo.