

# **SDN Community Contribution**

# (This is not an official SAP document.)

# **Disclaimer & Liability Notice**

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.



# **Applies To:**

SAP Web AS 6.40 Java SP11 (Sneak preview Slim Edition)

MaxDb 7.5

Hibernate 3.0.5

### **Summary**

Working with object-oriented software and a relational database can be cumbersome and time consuming in enterprise environments. To make life easier there are object/relational mapping (ORM) tools on the market. ORM refers to the technique of mapping a data representation from an object model to a relational model with a SQL-based schema. Hibernate is such an ORM tool. This article describes how to create a sample J2EE application using Hibernate on a SAP Web AS 6.40.

By: Peter Mayringer

Company: Capgemini

Date: 27 Jul 2005

### **Table of Contents**

Applies To:
Summary2
Table of Contents
What is Hibernate
Download and Install Hibernate
Configuration
Message World
Java Dictionary project5
Web application project6
Welcome.html6
Message class
MessageController Servlet8
Enterprise application project10
And Action!10
Author Bio



### What is Hibernate

Working with object-oriented software and a relational database can be cumbersome and time consuming in enterprise environments. To make life easier there are object/relational mapping (ORM) tools on the market. ORM refers to the technique of mapping a data representation from an object model to a relational model with a SQL-based schema. Hibernate is such an ORM tool.

On their website Hibernate is described as follows: "Hibernate is a powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition, and the Java collections framework. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with Java-based Criteria and Example objects. Unlike many other persistence solutions, Hibernate does not hide the power of SQL or JDBC from you and guarantees that your investment in relational technology and knowledge is as valid as always. Hibernate is downloaded more than 3000 times every day. Hibernate is a Professional Open Source project."

Hibernates goal is to relieve the developer from 95 percent of common data persistence related programming tasks. Hibernate is most useful with object-oriented domain models and business logic in the Java-based middle-tier. Hibernate can help you with the common translation from a tabular (relational) representation to a graph of objects.

### **Download and Install Hibernate**

You can download the hibernate framework from www.hibernate.org.

I used version 3.0.5 to write this article. On the hibernate website you can find a getting started guide, full documentation and several tools e.g. for creating a configuration file using a wizard. After downloading you can extract the zip file to a directory of your choice.

# Configuration

You need to add some jar files to your build path. Hibernate is packaged as a jar file: hibernate3.jar. Besides this jar file, you need to add some 3<sup>rd</sup> party jar files. Hibernate uses the following 3<sup>rd</sup> party jar files:

- antlr.jar (Hibernate uses ANTLR to produce query parsers, this library is also needed at runtime)
- asm.jar (Hibernate uses code generation library to enhance classes at runtime)
- cglib.jar
- commons-collections.jar (Hibernate uses various utility libraries from the Apache Commons project)
- commons-logging.jar
- dom4j.jar (for parsing XML configuration and metadata files)
- ehcache.jar (used for 2nd level chaching)
- log4j.jar (optional)



To setup your hibernate configuration you can use a XML document (hibernate.cfg.xml) or a plain text document (hibernate.properties). I recommend to use the XML format because it is easier to configure more advanced features like caching. I only added the minimal configuration settings to get Hibernate running. For more advanced settings see the official Hibernate Reference Documentation. Instead of using the default datasource you can also use a custom JNDI datasource which you added to your project. Don't forget to add 'java:comp/env/' if you make use of a self-defined JNDI datasource. The Hibernate configuration file should be stored in the WEB-INF/classes folder of your web project. In the mapping files section you can add your persistent classes. Another option is to add your persistent classes when you create a Hibernate Configuration object. I used the last option as you will see later in this document.

```
<!DOCTYPE hibernate-configuration PUBLIC
```

```
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

<hibernate-configuration>

```
<session-factory>
    <!-- properties --->
    <property name="connection.datasource">jdbc/SAPJ2EDB</property>
    <property
name="dialect">org.hibernate.dialect.SAPDBDialect</property>
    <property name="show_sql">false</property>
    </property
    <property name="show_sql">false</property>
    </property>
    </property>
    </property name="show_sql">false</property>
    </property>
    </property>
```

```
</hibernate-configuration>
```

### **Message World**

Let's get to work. I created a very simple sample application.

There is one HTML input form with two fields. One for message 1 and one for message 2.

This form can be submitted and is processed by a servlet. The servlet creates two message objects and persists them using hibernate. Message 1 is references message 2. Afterwards you can use SQL studio to display the contents of the table containing the two messages.





### Java Dictionary project

For persisting the message in a relational database we create a Java Dictionary project and define one table TMP\_MESSAGES. This table has three columns. MESSAGE\_ID holds the auto-incremented unique ID of the message. MESSAGE\_TEXT contains the text of the message. You already guessed, didn't you? NEXT\_MESSAGE\_ID contains a reference to the next message, which is the id of the next message.



🕑 Welcome.html	J) Hiberna	ate.java 🔬 hib	ernate.cfg.xml	TMP_MESSAG	ES 🗙			
Edit table								
General Information								
Describe the table								
Table:	TMP_ME:	5SAGES						
Description:	TMP_ME:	5SAGES						
Multi-client enabled								
Columns								
Define the columns of the	e table							
Column Name	Key	Simple Type Package	e Simple Type	Built-In Type	Length	Decimals	Not Null	Description
MESSAGE_ID	~			long			~	
MESSAGE_TEXT				string				
NEXT_MESSAGE_ID				long				

Create an archive and deploy this dictionary project.

### Web application project

Create a J2EE Web project.

#### Welcome.html

Here is the source code for the Welcome.html:



</FORM>

</BODY>

</HTML>

### Message class

We also need a Message class. We are going the persist the Message objects using Hibernate. The Message class has three properties: id (a unique id, the number is generated automatically by using a Hibernate feature), text (the actual message text), nextMessage (a reference to a next message). We define getter and setter methods for every property, like a normal JavaBean.

```
public class Message {
  private Long id;
  private String text;
  private Message nextMessage;
  private Message() {}
  public Message(String text) {
        this.text = text;
  }
  public Long getId() {
        return id;
  }
  public void setId(Long id) {
        this.id = id;
  }
  public String getText() {
        return text;
  }
  public void setText(String text) {
        this.text = text;
  }
  public Message getNextMessage() {
        return nextMessage;
  }
  public void setNextMessage(Message nextMessage) {
```



```
this.nextMessage = nextMessage;
}
```

Every class that needs to be persisted using Hibernate should be accompanied by a mapping configuration file. This file should be named <class name>.hbm.xml and it should be stored in the same folder as the .java file of the class. In the configuration file you can see that I use a auto-incremental field for the id of the message. And you can see the relation of message1 to message2 is described also. Using this configuration file Hibernate figures out how to store the Message objects in the database.

### MessageController Servlet

For simplicity I put all coding for Hibernate in this servlet. It would make sense however if you put the Hibernate configuration in a separate class.

- 1. A Hibernate object is configured and the persistent classes are added (addClass())
- 2. A Hibernate session is created using a SessionFactory
- 3. A transaction is started.
- 4. Message object 1 is created and saved.
- 5. Now Message object 2 is created and added to Message 1 in the nextMessage property.
- 6. The transaction is committed and the session closed.

The SAP Developer Network: <u>http://sdn.sap.com</u>



```
package com.capgemini.nl;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
public class MessageController extends HttpServlet {
  protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
              Configuration cfg = new Configuration();
              cfg.configure();
              cfg.addClass(com.capgemini.nl.Message.class);
              SessionFactory sf = cfg.buildSessionFactory();
              String text = request.getParameter("text");
              String text_sec = request.getParameter("text_sec");
              Session session = sf.openSession();
              Transaction tx = session.beginTransaction();
              Message message = new Message(text);
              session.save(message);
              if (text_sec!=null) {
                    Message message_sec = new Message(text_sec);
                    message.setNextMessage(message_sec);
              }
```



```
tx.commit();
              session.close();
              response.setContentType("text/html");
              PrintWriter out = response.getWriter();
              out.println("<HTML><HEAD><TITLE>Hibernate</TITLE></HEAD>");
              out.println("<BODY bgcolor=#E6E6FA><H1>Hibernate
Example</H1>");
              out.println( message.getText() );
              out.println("</BODY></HTML>");
  }
  protected void doPost(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
              doGet(request, response);
  }
}
```

### **Enterprise application project**

We create a Enterprise application project and add the web project to it. Build the archive and deploy the generated ear file.

You are now ready to test your application.

#### And ... Action!

Let's see how it works on my local J2EE engine.

Using	Hibernate	on	SAP	WAS
-------	-----------	----	-----	-----



Address 🕘 http://localhost:50000/HibernateExample/Welcome.html				
Example Hibernate				
message text: Hello				
2nd message text: World				
insert				
Address http://localhost:50000/HibernateExample/servlet/Me	essageController			
Hibernate Example				

Use SQL Studio to show the contents of the table. Or you could enhance the sample application to show the contents on a web page. You see that message Hello has a reference to message World. This is done by Hibernate because of the Message mapping configuration.

III SEL	ECT * FROM "S	APJ2EDB"."TMP	_MESSAGES"	
	MESSAGE_ID	MESSAGE_TEXT	NEXT_MESSAGE_ID	
1	7	Hello	8	
2	8	World	?	

I think you now have a good starting point to dive into Hibernate and start developing some more sophisticated applications.

# **Author Bio**

1

Peter Mayringer is senior consultant at Capgemini in the Netherlands. He has over 8 years of experience in software engineering. His main expertise areas are J2EE and SAP Netweaver. He is Sun certified Java Programmer, Web Component Developer and Business Component developer. <u>peter.mayringer@capgemini.com</u>

