DbForms User's Guide

Version 2.5

DbForms User's Guide: Version 2.5

Table of Contents

List of Figures

List of Tables

List of Examples

Part I. DbForms in a Nutshell

Table of Contents

Chapter 1. Introduction

1.1. About this document

This document assumes that the reader has a general understanding of how web applications are developed and deployed. This should include exposure to java servlets, JSP's, HTML and Web Application Archive files (WAR). Also important, is a good working knowledge of relational databases.

1.2. Related documents

The following pdf articles describe extensions to DbForms:

DbFormsDevelopersGUI [articles/DbForms_DevGui.pdf], J. Peer, Vienna, 2001

XSLPreprocessingLayerforDbForms [articles/XSL_DbForms.pdf], J. Peer, Vienna, 2001

1.3. About DbForms

1.3.1. What is it - a framework or a toolkit?

There are many views of what a framework is. Academic researchers tend to centre their definition around the concept of abstract classes:

- A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It is a reusable design for all or part of a software system [Johnson].
- Implementations of concrete classes or other software components which may be assembled together to build an application are often sold to industry under the name framework as well, but should be called toolkit instead.

DbForms is designed to be both a framework and a toolkit. Part I of this paper focuses mainly on the toolkit aspects of DbForms it demonstrates to the reader how to rapidly build a database-driven web-based Application using DbForms components. Part II deals more with the framework aspects of this product.

1.3.2. What are the benefits of using DbForms?

DbForms enables developers to build sophisticated web-based database driven applications in very short time and with very little efforts. DbForms applications are built in a manner conceptually similar to RAD - database building tools such as Microsoft Access (for Windows-based applications) or Sybase PowerSite (for web-based applications). (Note: All named products are properties of their respective owners.)

Rapid Application Development (RAD) tools in general, allow a user to place database components and action elements on templates which then get executed at runtime. As you will soon discover, dbForms uses these same techniques for its own framework.

An important benefit of using DbForms is its openness to other systems. You may use DbForms in conjunction with common JSP pages, Struts-based pages, etc. This means that you are free to use DbForms where it brings you the most benefits (dramatically reduce development efforts, etc.) and to use other

techniques if you think that they offer a better solution.

Chapter 2. DbForms Concepts

2.1. Technical background

DbForms is based on the following specifications: Java Servlets 2.3 and Java Server Pages 1.2 by Sun Microsystems. For more information on server-side Java technology please see [Servlet].

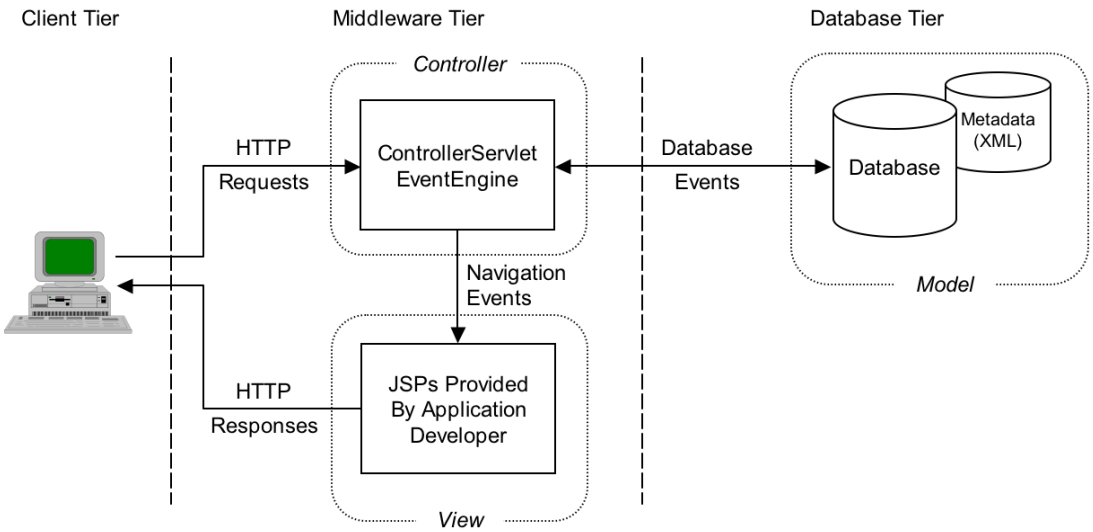
DbForms makes extensive use of the JSP Tag Library Extension included in the JSP 1.2 specification [Taglib].

XML parsing facilities and other parts of DbForms are based on code taken from Apache Group's Jakarta-Struts project [Struts].

2.2. The Model-View-Controller Design paradigm

DbForms implements the concepts of the Model-View-Controller design pattern [Gamma] which leads to the development of 3-tiered web-applications.

Figure 2.1. Architecture of DbForms



Like most applications and application frameworks, DbForms does not completely separate these three components. For instance, the **Controller Servlet** is the Controller Component of DbForms. Consider, however, the use of a hyperlink rendered by the user's web browser: clicking on it will trigger some operation hence implementing some controller functionality.

2.2.1. The Model: Database objects described by database metadata

The aim of DbForms is to perform operations on databases. The tables and views utilized by DbForms must be declared in an XML configuration file (`dbforms-config.xml`), which will be parsed and evaluated at web application start-up time.

Example 2.1. Defining the model

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dbforms-config PUBLIC "http://www.dbforms.org/dtd/dbf_conf.dtd">

<dbforms-config>
  <table name="customer"> ❶
    <field name="id" fieldType="int" isKey="true" /> ❷
    <field name="firstname" fieldType="char" />
    <field name="lastname"  fieldType="char" />
    <field name="address"   fieldType="char" />
  </table>

  <table name="orders"> ❸
    <field name="orderid" fieldType="int" isKey="true" /> ❹
    <field name="customerid" fieldType="int" isKey="true" autoInc="true" />
    <field name="date"       fieldType="char" />
    <field name="annotation" fieldType="char" />
    <field name="amount"     fieldType="int" />
  </table>

  <dbconnection name="jdbc/dbformstest" isJndi="true"/>
</dbforms-config>
```

❶❸ As shown, every table or view to be accessed by DbForms has to be declared inside a `table` element.

❷❹ All relevant table fields need to be declared in `field` subelements nested in their respective `table` element. (There exists a tool for generating this XML data automatically by reading database metadata, see Chapter 5, *DevGui* for more information.)

2.2.1.1. Defining the logical model

In DbForms, it usually makes no difference whether a `table` element really represents a database table or a logical view defined in the RDBMS. The circumstances under which views may be used instead of simple tables depends entirely on the RDBMS being used. Before using views, check that the RDBMS supports them. When tables are mentioned in this document in connection to some capability, the same capability may also work with views or joined tables.

Please see the section on Chapter 13, *Query Support* for more information on how to use the DbForms configuration file `dbforms-config.xml` to specify flexible joins, dynamic queries (which are alternatives for views in the database) and aliases.

2.2.1.2. Defining physical access to the model

DbForms needs to be able to create connections to the database containing the tables and fields declared in the `dbforms-config.xml` file, The following subsections show three different examples of these connections.

Versions up to 1.1.2 restrict you to use only *one* database per Web-Application. 1.1.2pr1 and after support multiple databases in a single application. See Multiple Database Connections for more info on that.

2.2.1.2.1. Accessing the Application Servers connection pool via JNDI

Example 2.2. Defining the database connection (1)

```
<dbconnection
  name   = "java:comp/env/jdbc/dbformstest"
  isJndi = "true"/>
```

In the example above DbForms assumes that the JNDI entry "jdbc/dbformstest" is configured correctly in the Application Server or Servlet engine's configuration (e.g. `data-sources.xml`) and takes all the connections it needs from that JNDI entry.

2.2.1.2.2. Using arbitrary connection pool

If you are not using JNDI in conjunction with a connection pool manager like Poolman, you have to tell DbForms which class it has to load in order to access the connection pool manager to be used. In addition, analogous to the JNDI key, you have to tell DbForms how the connection should be identified in the connection pool.

Example 2.3. Defining the database connection (2)

```
<dbconnection
  name   = "jdbc:poolman://dbformstest"
  isJndi = "false"
  class  = "com.codestudio.sql.PoolMan"/>
```

In the example above DbForms assumes that the connection pool entry called `jdbc:poolman://dbformstest` is correctly configured in the associated connection pool managers properties file and takes all the connections it needs from that connection pool.

The full configuration of data sources (how many pooled instances to create, duration of invalidation timeouts, etc.) is not in the scope of this user's guide. Please refer to your Application Server or Servlet engine documentation for more details.

DbForms now makes is easier to use connection pooling with the Protomatter and Jakarta connection pools. See Chapter 20, *Connection Support* for more information on using these or other pools.

2.2.1.2.3. Using no connection pool

If you just want to test the functionality of DbForms and you do not care about speed (at the moment), then you might want to define a simple database connection as follows:

Example 2.4. Defining the database connection (3)

```
<dbconnection
```

```
    name     = "jdbc:mysql://localhost/fashion"
    isJndi   = "false"
    conClass = "org.gjt.mm.mysql.Driver"
    username = "scott"
    password = "tiger"/>
```

Note: Many drivers pass in JDBC properties using the URL such as:

```
<dbconnection name = "jdbc:mysql://localhost/fashion?charSet=ISO-8859-1"
   ...
```

If your driver does not allow this and you need to pass in a property, please see Section 20.1.4, “Setting JDBC Properties” for another way to set JDBC properties.

2.2.2. The View: JSP templates provided by the application developer

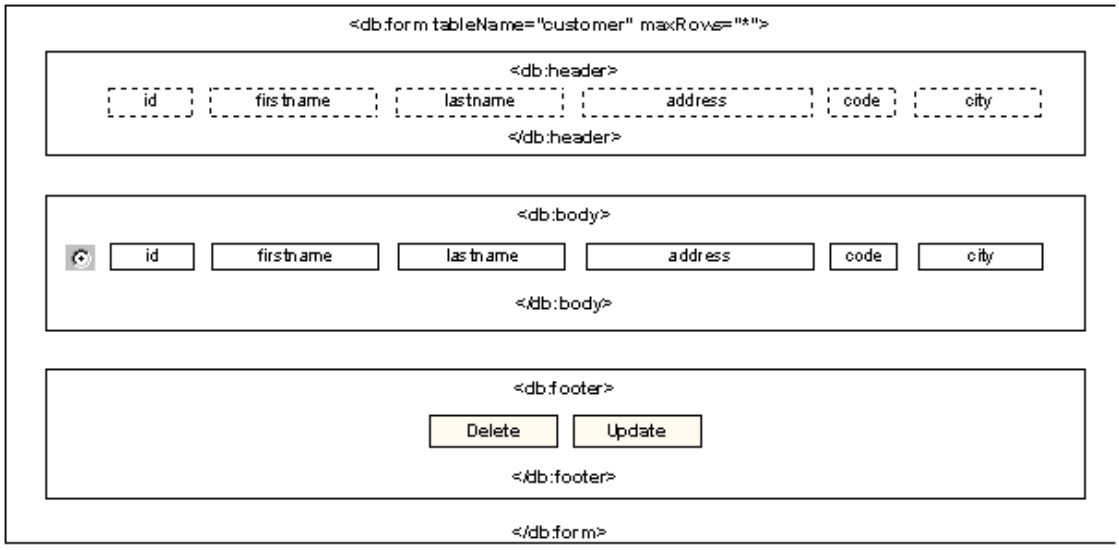
The view portion of a DbForms application is constructed using JSP technology. JSP files may contain static HTML elements as well as dynamic elements containing Java code (definitions, statements, expressions). For more information about JSP, refer to [JSP].

With release 1.1 of the JSP API, the concept of **Custom tag libraries** [Taglib] was introduced. Custom tags allow a developer to encapsulate even the most sophisticated Java code into an easy-to-use lightweight JSP tag. You may think of Tag libraries as a sort of advanced macro.

DbForms is, essentially, a collection of custom tags for placing data forms and data fields on JSP pages.

2.2.2.1. The structure of a DbForms View

Figure 2.2. Main components of a typical view



2.2.2.2. The basic concepts of a DbForm

Each DbForms View JSP may have one or more root tags of the type **dbform**. Every dbform tag has to contain exactly 1 header tag, exactly 1 body tag and exactly 1 footer tag, in exactly that order.

Each of those tags may contain subelements such as data fields, input fields, action buttons, and, of course, plain HTML text and JSP code.

Header and footer tags are commonly used for titles of pages, for labeling tables, for placing action and navigation buttons, input fields to enter new data, etc.

Header and footer tags get evaluated only once.

The body tag is used for showing data rows coming from the database, and for providing the user with functionality to edit that data. How many times the body tag and its subelements get executed (i.e., evaluated and rendered) depends on the value of the `maxRows` attribute of the form element (and of course, on the number of rows actually stored in the table).

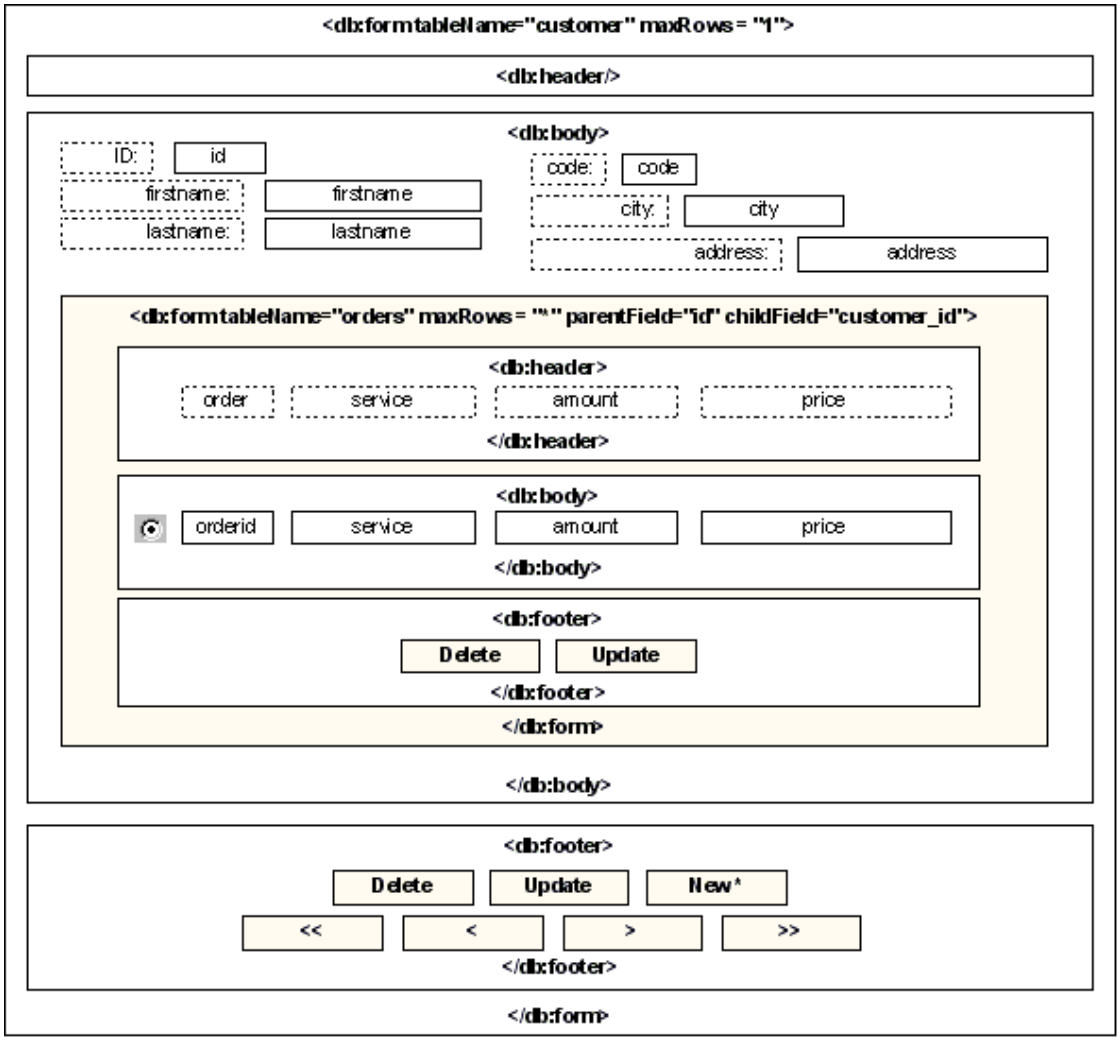
If `maxRows` is set to a number, the body gets executed up to that many times, limited by the number of rows in the database table.

If `maxRows` is set "*", the body gets executed an indefinite number of times, limited only by the number of rows in the database table. This should be used with caution lest the page become very large.

Nested forms

Every form may contain one or more nested subforms inside its body tag.

Figure 2.3. Example of a nested form



The orders form is nested within the body element of the customer form. The user will see *one* customer per page (because `maxRows` is set to 1) and *all* the orders (because `maxRows = *`) the customer has pending. The user may navigate through the list of customers by clicking the navigation buttons.

2.2.3. The Controller: Event parsing, dispatching and executing facilities

The Controller includes several components:

Controller-Servlet: this servlet is the single-point-of-entry for all incoming HTTP-requests.

EventEngine: a kind of assistant to the Controller-servlet. It focuses on filtering requests for WebEvents and instantiates them.

WebEvent Objects: all Objects derived from this abstract super-class have the ability to initialize themselves by reading a given request. These events get executed either by the controller directly or by the View.

The following example should give you a better picture of what the controller does and how it interacts with other components:

i. A user presses the delete row button on a DbForms application.

ii. The user's browser submits data via an HTTP POST command to the Controller servlet.

iii. The Controller-Servlet delegates the incoming request to the EventEngine which determines the main or primary event, (the event the user *explicitly* triggered by clicking a button). Secondary *implicit* events may also be generated. For example, under some circumstances, automatic updates of all changed input fields of all data rows will be generated.

iv. The EventEngine component parses the request and determines the kind of action the user wants to execute.

v. The EventEngine creates the appropriate WebEvent (in this case, a DeleteEvent) and delegates the Request object to this newly created WebEvent which initializes itself and returns the event back to the Controller-Servlet.

vi. The Controller tells the event, if it is a Database Event, to execute its built-in operation. Other types of event, e.g. Navigation Events, are delegated to the appropriate View component.

vii. The controller invokes EventEngine again to check if there are additional Secondary (implicit) events to be executed. If so, the appropriate WebEvents objects are created and executed in the same manner as the main event described above.

viii. The controller determines the View component to which the request should be forwarded.

ix. If the View component is a JSP page containing DbForms tags, those tags will search for Navigation Events to be executed. Once these are completed, a response will be generated.

x. This response is then rendered by the user's web browser.

Part II. Installation

Table of Contents

Chapter 3. Compatibility notes

Please see the release notes for a full list of changes.

Also, please note that all changes here are cumulative, meaning that (for example) if you upgrade from `1.1.2` to `1.1.4pr2`, you would need to read the sections for `1.1.3`, `1.1.4pr1`, and `1.1.4pr2`.

3.1. From Earlier Versions to DbForms 1.1.3

The `format` attributes of tags `queryData` and `tableData` have different semantics now that are not 100% backwards compatible to the 1.1.2 and earlier releases. See the *Tag Library* [../../taglib/DbFormsTags_Frame.html] for more information.

The insert button on JSP pages will not appear unless the attribute `showAlways="true"` is added to the `insertButtonTag`. Otherwise, it will only appear in insert mode such as through a `navNewEvent`.

3.2. From Earlier Versions to DbForms 1.1.4pr1

If you are upgrading from 1.1.3 or from 1.1.4dev (first dev release), you must make three changes to `dbforms-config.xml`.

These changes need to be made to the `<servlet-class>` element only and the `<servlet-name>` can remain as it is in your `dbforms-config.xml` file.

i.

```
From:
  <servlet-name>config</servlet-name>
  <servlet-class>org.dbforms.ConfigServlet</servlet-class>

To:
  <servlet-name>config</servlet-name>
  <servlet-class>org.dbforms.servlets.ConfigServlet</servlet-class>
```

ii.

```
From:
  <servlet-name>control</servlet-name>
  <servlet-class>org.dbforms.Controller</servlet-class>

To:
  <servlet-name>control</servlet-name>
  <servlet-class>org.dbforms.servlets.Controller</servlet-class>
```

iii.

```
From:
  <servlet-name>file</servlet-name>
  <servlet-class>org.dbforms.util.FileServlet</servlet-class>

To:
  <servlet-name>file</servlet-name>
```

```
        <servlet-class>org.dbforms.servlets.FileServlet</servlet-class>
```

Not doing so will result in an error message:

```
java.lang.ClassCastException
  at org.dbforms.taglib.DbFormTag.setPageContext(DbFormTag.java:1087)
  at org.apache.jsp.menu_jsp._jspService(menu_jsp.java:91)
  at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:137)...
```

Similarly, to use JapsperReports please adjust the `<servlet-class>` to be:

```
    <servlet>
        <servlet-name>startreport</servlet-name>
        <display-name>startreport</display-name>
        <servlet-class>org.dbforms.servlets.StartReportServlet</servlet-class>
    </servlet>
```

3.3. From Earlier Versions to DbForms 1.1.4pr2

3.3.1. From all versions

(also see note for 1.1.4pr1 above) *When upgrading from any previous release please note that:* To have better control of the fields written to the database, we changed the interceptor interface of the preUpdate, preInsert, and preDelete events to:

```
   public int preXXXXXX(HttpServletRequest request,
                        Table table, FieldValues fieldValues,
                        DbFormsConfig config, Connection con)
          throws ValidationException
```

Now you have better control over the fields written to the database. You can remove, change, and add fields. Note however, that changes to fields marked with `isKey="true"` in the `dbforms-config.xml` file, are still disregarded by the preDelete and preUpdate methods.

In many cases, you only need to change the interfaces in your old interceptor code because the interface of the new FieldValues class is nearly the same as the previously used Hashtable.

However, to retrieve values from `fieldValues` you will need to change your interceptors in one of two ways. For example to retrieve a String

```
   String kanji = (String) fieldValues.get("kanji");
```

you now need to use one of the two following approaches.

i.

```
String kanji = (String)fieldValues.get("kanji").getFieldValueAsObject();
```

This will parse the string and build an object of the required type. If the input field is of type String, it returns a String.

ii.

```
String kanji = fieldValues.get("kanji").getFieldValue();
```

This returns the String as it is written to the form. No further parsing is done as it returns a String.

Please also note that fields can now be removed using `fieldValues.remove("yourNameOfTheField");` *which was not previously possible.*

3.3.2. From 1.1.4pr1

If you are upgrading from 1.1.4pr1, please adjust the following for the `<filterCondition>` tag (used embedded within the new `<filter>` tag). You must change your filter values for strings to not use the apostrophe. This is now done automatically.

So:

```
name like '?'
```

must be changed to

```
name like ?
```

3.4. From Earlier Versions to DbForms 2.0

To upgrade to 2.0 please see the sections on 1.1.4pr1 and 1.1.4pr2 as they were the basis for this release.

3.5. From Earlier Versions to Dbforms 2.3

To upgrade from any previous versions, an adjustment needs to be made in the `dbforms-config.xml` file if you are using BLOB-fields stored in your database and wish to retain the original way of storing that data. To do that, set the optional attribute `"blobHandling"` to `"classic"`. Please see BLOB-fields for details.

```
<dbforms-config>
   <table name="pets" blobHandling="classic">
   ...
```

3.6. From Earlier Versions to DbForms 2.4.rc1

The `pattern` attribute for `textField`, `dateField`, `dateLabel`, `textArea`, and `textAre-`

`aForBlobs` now support global settings for both type and language.

Previously, the `dateField` was set via the `<date-format>` attribute in `dbforms-config.xml`. Now, all `pattern` attributes are set via message resources. Please see Internationalization for details.

Chapter 4. Installation

You can utilize a binary distribution of DbForms in your own web applications by following the steps in the subsequent sections.

Users of MS SQL Server should refer instead to Chapter 6, *Jump-start a MS SQL Server* instead.

If you have questions, please drop a line to our mailing list at jdbforms-interest@lists.sourceforge.net. The messages for this list are archived in the Forums area of the DbForms project on SourceForge.net. These message are a good source of information for resolving problems or getting additional information about using DbForms.

4.1. Prerequisites

i. Download and install a Java 2 Development Kit implementation for your operating system platform (version 1.4 or 1.5).

ii. Define an environment variable JAVA_HOME that points at the base directory of your JDK installation, and add the directory "$JAVA_HOME/bin" to your PATH variable.

iii. Download and install a servlet container that supports the Servlet API Specification, version 2.3 or later, and the JavaServer Pages (JSP) Specification, version 1.2 or later. (A useful servlet container is *Tomcat* [http://jakarta.apache.org], version 4.0.x or later. Tomcat 3.3.x is only compatible until DbForms v1.1.2.)

iv. Download and unpack a DbForms binary distribution from SourceForge.net.

4.2. Running the examples

4.2.1. Bookstore example

The best way to do this is to download the Bookstore example in WAR file which is ready to run with minimal effort. The WAR file can be placed in the `webapps` directory of Tomcat, for example, and Tomcat will expand it. You may need to add a `Context` element to Tomcat's `server.xml` file. The Bookstore example in WAR form contains all dependencies needed to run and uses an in-memory database (HSQL) that does not require additional configuration or installation.

Once these prelimiaries are performed, point your web browser to the host and port where Tomcat (or whatever servelet container or application server you are using) is running and open `bookstore/howto/`. There are a few examples in this directory and a large number examples under `bookstore/tests/` that demonstrate various features of DbForms in the simplest form possible.

4.2.2. Tutorial example

The tutorial example is described in Chapter 7, *Tutorial: A sample application* including information on how to run it.

4.2.3. Other examples

Please note that the test suite and bugtracker examples include dependencies from earlier releases of DbForms. These dependencies should not be replaced with current dependencies.

i. Create the database by executing the SQL commands listed in `examples/*/WEB-INF/*.sql`.

ii. Configure a database connection pool for the new database tables using your favorite technique (either the connection pool built into your application server or codestudio's poolman or another database connection tool). Connection pools are an important technique for speeding up a web application. See Section 2.2.1.2.2, “Using arbitrary connection pool” for more information.

iii. Install the DbForms example applications by deploying the directory of the application into your servlet container, using the standard techniques supported by that container.
For Tomcat, this may be as simple as copying the directory to the $TOMCAT_HOME/webapps directory and restarting Tomcat.

iv. Take a look at /examples/README.txt or other README files in the examples directory!

v. Finally, using your web browser, go to the host and port and path where the examples should be.

4.3. Building your own DbForms-capable applications

Use the checklist below as a guide for setting up a DbForms application.

i. Copy the file `bin/dbforms.tld` from the DbForms distribution into the `WEB-INF` directory of your web application.

Note. The file `dbforms.tld` was named `taglib.tld` in some prior versions. The name `taglib.tld` was changed because it was too generic and the risk of naming conflicts too great.

ii. Copy the file `bin/dbformsX.Y.jar` (where X.Y is the release number) from the DbForms distribution into the `WEB-INF/lib` directory of your web application.

iii. Copy the jars in the `dependend` directory of the DbForms distribution into the `WEB-INF/lib` directory of your web application.

iv. Create or modify the J2EE deployment descriptor `WEB-INF/web.xml` file to include the following items. See Section 4.3.1, “Edit deployment descriptor `web.xml`” for more details.

- Three `servlet` elements to define the control, config, and file servlets and their URL mappings. Other servlets may also be needed for additional capabilities such as JasperReports, Excel output, and CSV output.
- A tag library declaration for the DbForms tag library.
- Security measures for your application.

v. Create a file `WEB-INF/dbforms-config.xml` that defines the tables, the fields, and the database connection to use. See Chapter 8, *Configuration Files* for details of the contents of this file. This can be generated AUTOMATICALLY (along with JSP pages) by using the DevGui application. See Chapter 5, *DevGui* for more information.

It is also possible to use more than one configuration file. Simply list them with commas between them in the `dbformsConfig` init param of the `ConfigServlet` in `web.xml` as shown in this fragment from the `dbforms-config.xml` file.

```
<init-param>
```

```
    <param-name>dbformsConfig</param-name>
    <param-value>
      /WEB-INF/dbforms-config.xml,/WEB-INF/custom/dbforms-config.xml
    </param-value>
  </init-param>
```

vi. Optionally, create a `WEB-INF/dbforms-errors.xml` file that defines error messages. See the `xmlErrors` tag in the separate DbForms Custom Tag Library document in the `docs/taglib` directory for more information.

vii. In each JSP page that will use the DbForms custom tags, add a line at the top of the page like this:

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
```

The prefix "`db`" followed by a colon is placed in front of each tag from the DbForms tag library. The above declaration gives the association between the tag library and the prefix. If you wish, you can use a prefix other that "`db`". This document, however, assumes a prefix of "`db`".

viii. Verify the structure of your deployment. See Section 4.3.2, “Typical deployment structure of applications using DbForms” for an example of a complete directory and file structure.

ix. Check that the database configuration defined in `dbforms-config.xml` is really working. Whether this is an easy task or not, depends heavily on the database server you are using. One way to do this is to use the database tab in DevGui (Section 5.4.2, “Database Tab”) to set up a simple, non-JNDI and non-connection pool connection. Common reasons for database-related problems are:

- JDBC drivers not in classpath (of servlet container).
- Connection pool classess not in classpath.
- Driver/Connection-pool resources (property-files) not in classpath.
- Misspelled connection URLs or JNDI keys.
- Database authentication problems (network versus direct connect, user id, passwords).

4.3.1. Edit deployment descriptor `web.xml`

4.3.1.1. Changes in web.xml to enable DbForms

The listing shown below is a complete deployment descriptor for a DbForms application. If the web application has additional servlets in addition to the servlets that comprise DbForms, they would be added to the descriptor in the normal manner.

The order of elements in `web.xml` is important so be sure to follow the DTD/Schema. For convenience, the top level of the root element `web-app` is shown below.

```
<!ELEMENT
    web-app (icon?, display-name?, description?, distributable?,
    context-param*, filter*, filter-mapping*, listener*,
    servlet*, servlet-mapping*, session-config?, mime-mapping*,
```

```
    welcome-file-list?, error-page*, taglib*,
    resource-env-ref*, resource-ref*, security-constraint*,
    login-config?, security-role*, env-entry*, ejb-ref*,
    ejb-local-ref*)>
```

Add the following `servlet`, `servlet-mapping`, `taglib`, and `context-param` (optional) elements to the `web-app` element in the `/WEB-INF/web.xml` file.

The sample listing below has portions keyed to remarks that follow.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <!--============== DBFORMS STYLING TEMPLATES ==============-->
  <context-param>
    <param-name>templateBase</param-name> ❶
    <param-value>templates</param-value>
  </context-param>

  <!--============ DbForms Configuration Servlet ============-->
  <servlet>
    <servlet-name>config</servlet-name>
    <servlet-class>org.dbforms.servlets.ConfigServlet</servlet-class>

    <init-param>
      <param-name>log4j.configuration</param-name> ❷
      <param-value>/WEB-INF/log4j.properties</param-value>
    </init-param>

    <init-param>
      <param-name>digesterDebugLevel</param-name>
      <param-value>0</param-value>
    </init-param>

    <init-param>
      <param-name>resourceBundle</param-name> ❸
      <param-value>ApplicationResources</param-value>
    </init-param>

    <init-param>
      <param-name>validation</param-name> ❹
      <param-value>/WEB-INF/validation.xml</param-value>
    </init-param>

    <!--More than one validation file is possible-->
    <!--init-param>
      <param-name>validation</param-name>
      <param-value>/WEB-INF/validation.xml</param-value>
    </init-param-->

    <init-param>
      <param-name>validator-rules</param-name>
      <param-value>/WEB-INF/validator-rules.xml</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
  </servlet>
```

```
    <!--============ DbForms Controller Servlet ==============-->
    <servlet>
      <servlet-name>control</servlet-name>
      <servlet-class>org.dbforms.servlets.Controller</servlet-class>

      <init-param>
        <param-name>maxUploadSize</param-name> ❺
        <param-value>80000</param-value>
      </init-param>
    </servlet>

    <!--============ DbForms FileServlet =====================-->
    <servlet>
      <servlet-name>file</servlet-name>
      <servlet-class>org.dbforms.servlets.FileServlet</servlet-class>
      <load-on-startup>2</load-on-startup>
    </servlet>

    <!--==== Controller Servlet and FileServlet Mappings========-->
    <servlet-mapping>
      <servlet-name>control</servlet-name>
      <url-pattern>/servlet/control</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
      <servlet-name>file</servlet-name>
      <url-pattern>/servlet/file</url-pattern>
    </servlet-mapping>

     <!--============ Session config ====================-->

     <session-config>
       <session-timeout>30</session-timeout>
     </session-config>

     <!--============ Welcome File List ====================-->

     <welcome-file-list>
       <welcome-file>index.jsp</welcome-file>
     </welcome-file-list>

     <!--============ DbForms Tag Library Descriptor ==========-->

     <taglib>
       <taglib-uri>/WEB-INF/dbforms.tld</taglib-uri>
       <taglib-location>/WEB-INF/dbforms.tld</taglib-location>
     </taglib>

     <!--============ Application Security Controls ==========-->

   </web-app>
```

Remarks:

❶ This is optional. See Section 11.3.4.1, “Template base directory ” for more information.

❷ The initialization parameter `log4j.configuration` should point to a valid log4j property file. If this parameter is not defined, the log4j standard configuration will be used for logging.

Now that log4j handles all console output, you would think that by setting log4j to display fatal errors only, your console would be virtually empty. Well, no! The current version of DbForms uses the struts digester class to parse xml files at startup. This class sends info to System.out. The ini-

tialization parameter `digesterDebugLevel` setting can be used to specify (to the digester) the amount of detail to display. Values can range from 0 (no debug) to 10 (Show all).

Sample log4j.properties file:

```
#begin log4j.props
#IMPORTANT -  Watch for trailing whitespaces after each statement!!!

#log4j.rootCategory=debug, stdout, logFile
#log4j.rootCategory=warn, stdout
log4j.rootCategory=error, stdout
log4j.category.org.dbforms=warn

#out to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

#out to file
log4j.appender.logFile=org.apache.log4j.RollingFileAppender

log4j.appender.logFile.File=c:\log4j\dbforms.txt
#log4j.appender.logFile.File=/var/tomcat/logs/dbforms.log
log4j.appender.logFile.MaxFileSize=100KB
log4j.appender.logFile.MaxBackupIndex=1
```

❸ DbForms v1.1 supports internationalization (i18n). Labels, messages, etc. can be stored in "resource bundle" files and retrieved using appropriate DbForm tags. The optional initialization parameter `resourceBundle` setting is used to specify the resource bundle root name. Refer to Section 16.2, “Defining Resource Bundles ” for more detail.

❹ Enhanced validation features are available because the Apache Commons-Validator framework has been integrated into DbForms. This framework requires the use of two distinct xml descriptor files. The optional initialization parameters `validator-rules` and `validation` are used to specify these files. Refer to Section 17.1, “Commons-Validator framework ” for more discussion.

❺ The initialization parameter `maxUploadSize` tells DbForms how big of a file upload it should accept. If no value is set, a default value will be used. Setting any kind of upload size is important to protect the system against DoS (Denial of Service) attacks.

4.3.1.2. Deploying a security aware application

DbForms inherits all of the security controls present in the servlet container, either an application server or a dedicated servlet engine such as Tomcat. Therefore, please refer to documentation for the Application Server or Servlet Engine being used for information on this topic. The Java Servlet specification also contains information about web application security.

A typical security-aware web application deployment descriptor has elements inside its `web-app` element as shown below.

```
<!-- ==================================================== -->
<!-- =            APPLICATION SECURITY                    = -->
<!-- ==================================================== -->

<!-- ================== authentication ================= -->
<!--
  We protect everything in directory /protected
  however, be sure NOT to protect login.jsp or logout.jsp since,
```

```
  in most containers, this would lead to endless recursion
  which ends with a crash of the Java Virtual Machine ("StackOverFlow")
-->

<security-constraint>
  <web-resource-collection>
    <web-resource-name>myapp</web-resource-name>
    <url-pattern>/protected/*</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <role-name>a_users</role-name>
    <role-name>b_users</role-name>
    <role-name>guests</role-name>
  </auth-constraint>
</security-constraint>

<!--==================== login config ====================-->
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Form-Based Authentication Area</realm-name>

  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/login.jsp</form-error-page>
  </form-login-config>
</login-config>

<!--<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Example Basic Authentication Area</realm-name>
</login-config>-->

<!--==================== roles ============================-->
<security-role>
  <description>Have read only access</description>
  <role-name>guests</role-name>
</security-role>

<security-role>
  <description>Registered users can do stuff, others not.</description>
  <role-name>a_users</role-name>
</security-role>

<security-role>
  <description>Registered users can do less then users_a</description>
  <role-name>b_users</role-name>
</security-role>
```

Remarks:

User authentication and mapping of users to roles are both performed by the servlet container. For Tomcat, this is controlled via the `Realm` element in the `conf/server.xml` file.

DbForms provide some additional security controls beyond the controls inherited through the container. See Chapter 9, *Security* for fine-grained definition of rights for data access and manipulation.

4.3.2. Typical deployment structure of applications using DbForms

Finally, here a short list of files to check when deploying a web application using DbForms:

Example 4.1. Structure of a typical DbForm application

Bold items are required.

```
/yourfile1.jsp
/yourfile2.jsp
/yourfile1.html
/yourfile2.html
/yourfile1.css

../templates/yourTemplateA/yourTemplateA_begin.jsp
../templates/yourTemplateA/yourTemplateA_end.jsp
../templates/templateB/templateB_begin.jsp
../templates/templateB/templateB_end.jsp

/WEB-INF/web.xml
/WEB-INF/dbforms-config.xml
/WEB-INF/dbforms.tld
/WEB-INF/log4j.properties
/WEB-INF/validation.xml
/WEB-INF/validator-rules.xml

/WEB-INF/lib/dbforms2.X.Y.jar
/WEB-INF/lib/cactus-13-1.7dev20041109.jar
/WEB-INF/lib/cewolf-0.10.1.jar
/WEB-INF/lib/commons-beanutils-1.6.jar
/WEB-INF/lib/commons-collections-2.1.jar
/WEB-INF/lib/commons-dbcp-1.1.jar
/WEB-INF/lib/commons-digester-1.5.jar
/WEB-INF/lib/commons-el-1.0.jar
/WEB-INF/lib/commons-fileupload-1.0.jar
/WEB-INF/lib/commons-httpclient-2.0.jar
/WEB-INF/lib/commons-lang-2.0.jar
/WEB-INF/lib/commons-logging-1.0.3.jar
/WEB-INF/lib/commons-pool-1.1.jar
/WEB-INF/lib/commons-validator-1.0.2.jar
/WEB-INF/lib/hsqldb-1.7.1.jar
/WEB-INF/lib/httpunit-1.5.4.jar
/WEB-INF/lib/itext-1.01.jar
/WEB-INF/lib/jasperreports-0.6.4.jar
/WEB-INF/lib/jcommon-0.9.6.jar
/WEB-INF/lib/jfreechart-0.9.21.jar
/WEB-INF/lib/junit-3.8.1.jar
/WEB-INF/lib/log4j-1.2.8.jar
/WEB-INF/lib/maxq-0.95dev.jar
/WEB-INF/lib/nekohtml-0.9.3.jar
/WEB-INF/lib/oro-2.0.8.jar
/WEB-INF/lib/poi-2.0-final-20040126.jar
/WEB-INF/lib/protomatter-1.1.8-pre5.jar
/WEB-INF/lib/rhino-1.5R4.1.jar
/WEB-INF/lib/servletapi2.3
/WEB-INF/lib/servletapi2.4
/WEB-INF/lib/xalan-2.5.1.jar
/WEB-INF/lib/xercesImpl-2.6.0.jar
/WEB-INF/lib/xmlParserAPIs-2.2.1.jar

/WEB-INF/classes/your_package/yourservlet1.class
/WEB-INF/classes/your_package/yourservlet2.class
```

```
/WEB-INF/classes/your_package/yourinterceptorI.class
/WEB-INF/classes/your_package/yourinterceptorII.class

/WEB-INF/classes/...
```

Remarks

The above list of required JAR files is dependent on the version of DbForms being used and also dependent on features of DbForms used in the application. The list is current as of version 2.5 but it is prudent to check the contents of directory `dependend` in the binary distribution since that directory defines the official set of dependencies.

The JDBC driver may, in some cases such as for simple connections, be placed in the `WEB-INF/lib` directory.

For Tomcat, the directory `common/lib` may also be used to place the DbForms JAR file and other dependencies in a single location that can be shared among multiple DbForms applications.

Other dependencies, such as JDBC drivers may be placed in Tomcat directory `server/lib` where they may be used by Tomcat to be shared among all web applications in the container.

In some cases, when there is a verson conflict between a DbForms dependency and the same library needed by the servlet container, a JAR file must be placed in the `server/lib` directory (or its equivalent in a container other than Tomcat) instead of the application's `WEB-INF/lib` directory.

4.4. Optional configuration for large tables

Some users find that, for tables with many rows, configuring `dbforms-config.xml` to use the `classic` navigation system to be advantageous. The newer, higher-performance default `datalist` navigation technique stores the result set in memory which may lead to memory issues. Of course, if filters are used so that the whole table is not being loaded in one page, then this may not be an issue even with very big tables.

More information on the navigation systems and how to configure the `classic` system can be found in Chapter 26, *Navigation*.

Chapter 5. DevGui

5.1. Introduction

DevGui provides tools for **automatic generation of the DbForms configuration XML file** `dbforms-config.xml` and for **automatic generation of JSP views**. It uses a convenient Swing-based application interface that can:

- Automatically create an initial `dbforms-config.xml` for your database by reading the meta data about tables and columns inside your database, and
- Create JSPs that use the DbForms tag library to create a running web application for your database without having written a single line of code.

5.2. Installation

DevGui is a simple Swing application, not a web application, so you do not need a Servlet or JSP container like Tomcat to run it. These containers, however, will be needed later to run a DbForms web application that is generated in part by DevGui. For DevGui, the following will be needed.

i. Complete the steps described in Section 4.1, “Prerequisites”.

ii. Set an environment variable named `DBFORMS_HOME` which points to the directory containing the dbforms distribution. For example,

```
DBFORMS_HOME=h:\dbforms
```

where the exact command to use depends on your operating system.

- On Win32, you can type

```
SET DBFORMS_HOME=h:\dbforms
```

or you can use the system control panel. Be sure there are no spaces in the path or you will get a `java.lang.NoClassDefFoundError`. For example, if the path includes a directory named "`Program Files`" or "`Documents and Settings`", **DbForms will not work**.

- On Unix, you can use the `setenv` or `export` commands, depending on your shell. For `csh`, and `tcsh` shells use something like the following.

```
setenv DBFORMS_HOME /home/user123/dbforms
```

For the `sh` shell, the above example would become the following.

```
DBFORMS_HOME=/home/user123/dbforms
export DBFORMS_HOME
```

For the `bash` and `ksh` shells, the above two commands can be combined into

```
export DBFORMS_HOME=/home/user123/dbforms
```

iii. A JDBC driver for your DBMS, listed in your `CLASSPATH`. This will also be needed later inside the running web application.

iv. Libraries for Xalan-Java XSLT processor and an XML parser like Xerces. **If your JDK is 1.4.x or later, you may skip this step.** The binary distributions may be downloaded from *Xalan distribution* [http://xml.apache.org/xalan-j/]. Include `xalan.jar` and `xercesImpl.jar` in your `CLASSPATH`.

You must include these libraries in your general `CLASSPATH` setting since DevGui does not run as a web application.

5.3. Running

Start the shell script in `$DBFORMS_HOME/dbforms/bin` appropriate for your operating system.

```
devgui.bat (win32)
devgui.sh  (unix,linux)
```

The Swing-based GUI should appear now. Documentation about using DevGui can be found in the next section.

Before using the generated application, you have to copy the stylesheet file `dbforms.css` and the `dbformslib` directory (both can be found within `misc` directory) to your web application. The `dbformslib` directory contains a subdirectory containing two icons and another subdirectory for the Javascript calendar application.

5.3.1. Alternative method of running DevGui via Ant

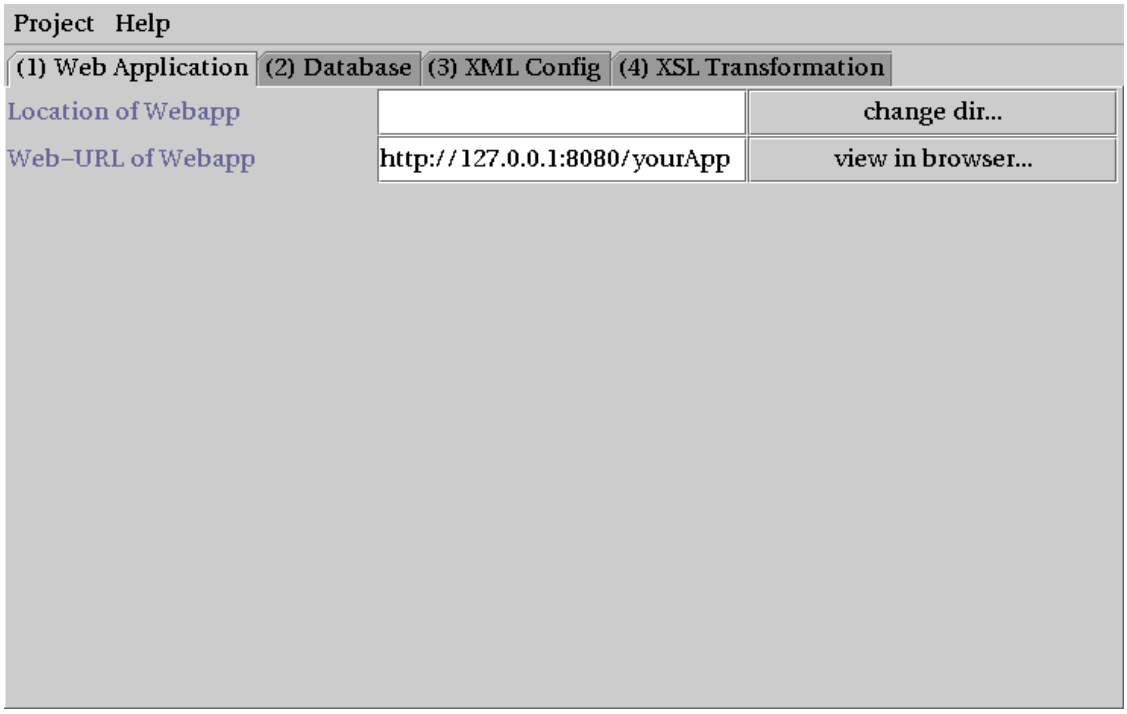
If you already have Ant installed, and are comfortable with tweaking `build.xml` files then you can use the <devgui> target. The `DBFORMS_HOME` directory should be set to your current location of DbForms. Make sure however that you have generated the DbForms jar by running the <jar_classes> target. You will also need to place your JDBC drivers in your `ant/lib` directory, or tweak the `build.xml` to include where your JDBC drivers are.

```
ant <jar_classes>
ant <devgui>
```

5.4. Use

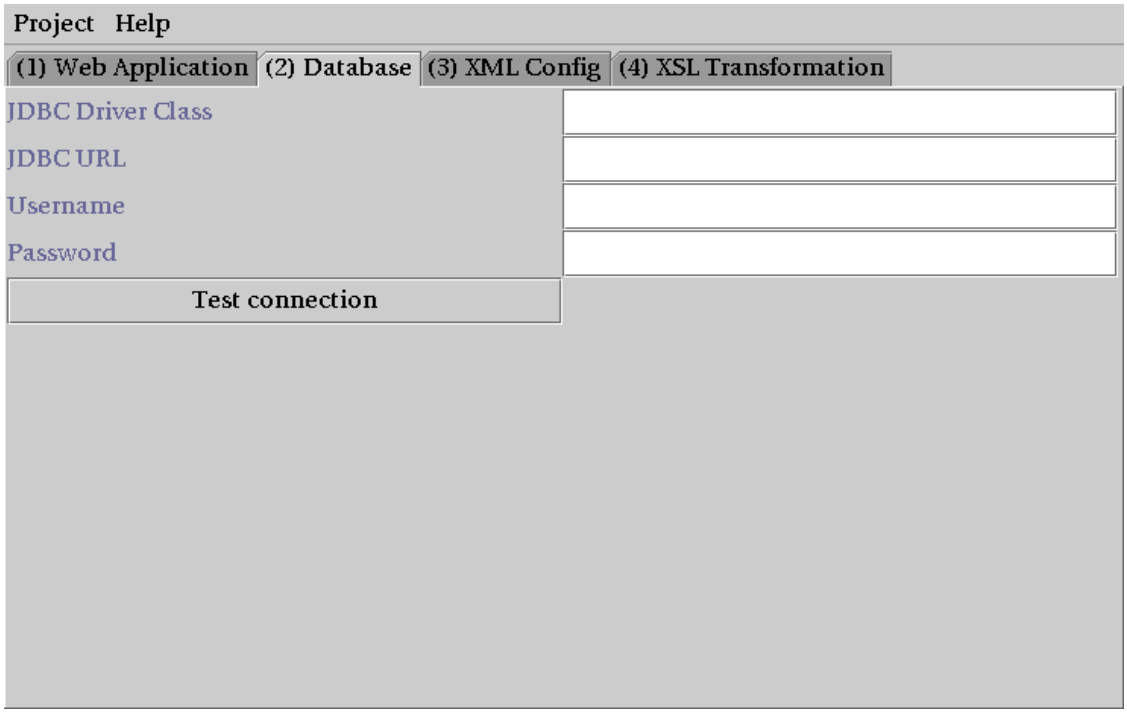
The following sections describe each tab in the DevGui tool.

5.4.1. Web Application Tab



- **Location of Webapp:** The JSP files generated by DevGui's XSL transformations will be written into this directory.
- **Web-URL of Webapp:** URL which can be used to access the application.

5.4.2. Database Tab



- **JDBC Driver Class:** The name of java class which implements `java.sql.Driver` and which will be used to connect to the database.

  Examples

  - `com.databasename.jdbc.Driver`
  - `org.freedb.jdbc.type4driver.MainDriver`

  See your driver's documentation.

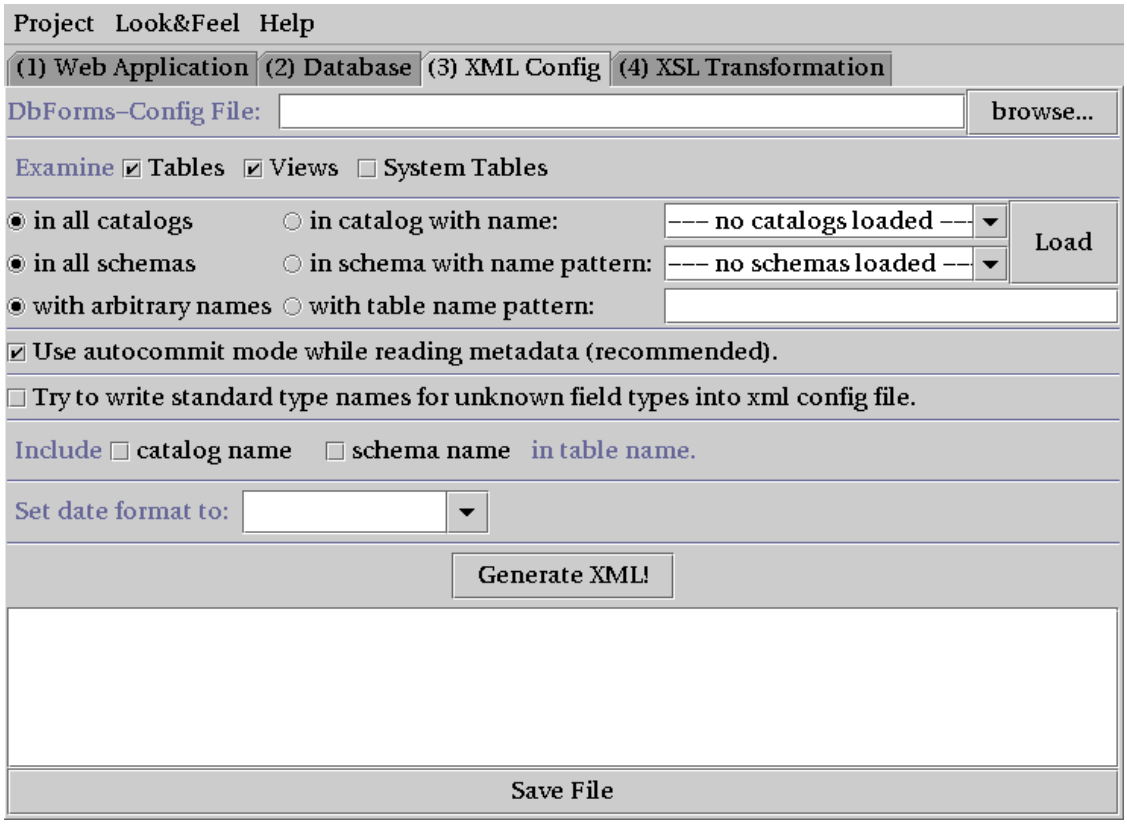
- **JDBC URL:** URL to connect to database. Refer to driver documentation.

  Examples:

  - `jdbc:foo:bar://host123:876/myveryimportantdb`
  - `jdbc:abcd://host2.dom1.dom2.dom3/database=db3;option1=yes`

- **Username:** Username for DB connection.
- **Password:** Password for DB connection.
- **Test Connection button:** Can (and should) be used to check if above entries are ok.

5.4.3. XML Config Tab



- **DbForms-Config File:** Name of XML file which contains config for dbforms. An initial version of this file can be automatically generated by DevGui by reading meta data from the database. This file is also used as input for XSL transfortations in tab 'XSL Transformation'.

  File name should be something like:

  ```
  <location-of-webapp>/WEB-INF/dbforms-config.xml
  ```

- **Examine tables, views, system tables checkboxes:** Select which table types and table descriptions should be generated into the configuration file.

- **Catalog name filter:** It depends on the database system, whether it supports catalogs and how it maps this general term to a database specific term. Often a catalog is mapped to a 'database'.

  The easiest thing is always to select 'all catalogs', but if needed, you may try to restrict the set of examined tables by entering a catalog name and selecting the corresponding radio button.

  Be aware that several database systems that map 'catalog' to 'database' do not allow for reading information about tables in a database other than that one currently in use.

- **Schema name filter**: It depends on the database system, whether it supports schemas and how it maps this general terms to a specific term. Often a schema is mapped to 'table owner'.

  The easiest thing is always to select radio button 'all schemas', but if needed, you may try to restrict the set of examined tables by entering a schema name pattern and selecting the corresponding radio button.

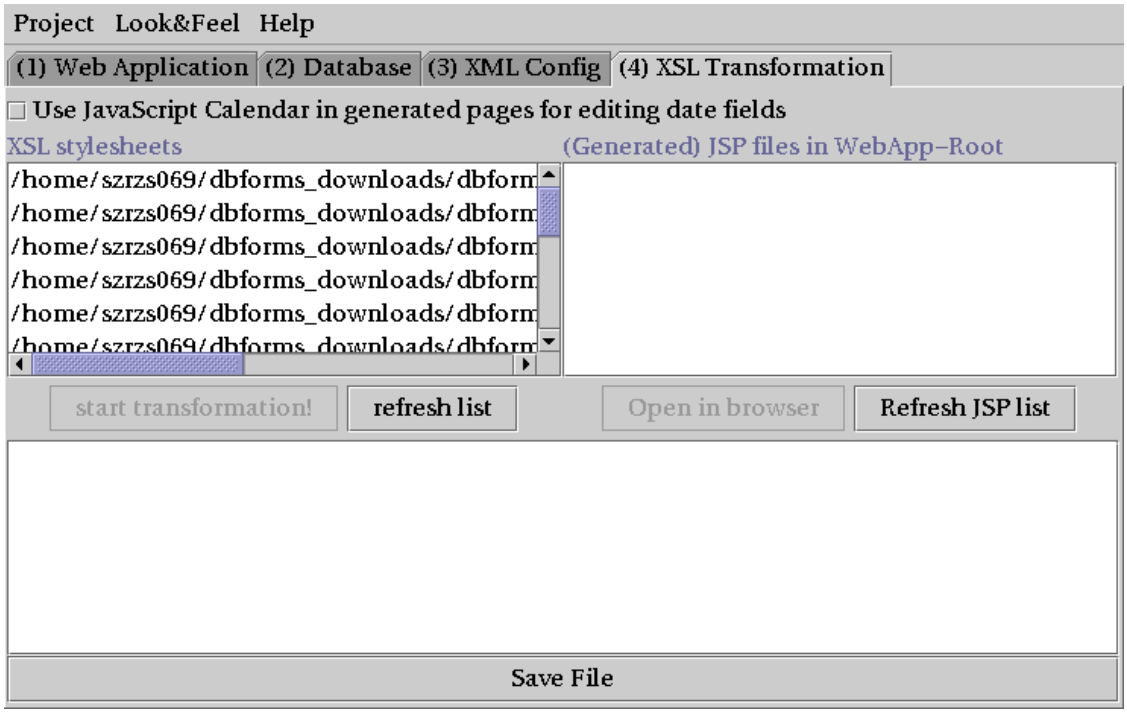
While you always have to enter a complete catalog name, the field for schema name may contain a schema pattern with '%' as pattern for an arbitrary (maybe empty) string.

Examples:

- 'foo%' would select all tables in a schema with a name that begins with substring 'foo'.
- '%nix%' would select all tables in a schema with a name that contains a substring 'nix'.

- '**Load' button:** You can try to load the set of available catalog and schema names into the corresponding combo boxes by clicking the 'Load' button. You must have completed the database tab before pressing this button.
- **Table name filter:** Select tables with arbitrary names or with a name that matches a certain name pattern. See comments about patterns above.
- **Use autocommit mode:** Should normally be checked, was inserted as a workaround for a bug of a special driver.
- **Write standard type names for unknown field types:** If not checked, DevGui will write the names for column types exactly as it gets them from the database. Sometimes this may lead to problems, if the DBMS uses a type that could be handled by DbForms but has a name unknown to DevGui. E.g. there are a lot of different names for various 'integer' types. If this option is checked, DevGui will try to write a standard name for all unknown types into the generated configuration file.
- **Foreign key detection:** Devgui can use meta data methods to determine foreign key relationships between tables and write corresponding tags into the generated config file. This can surely only work if the used database system as well as the JDBC driver supports this feature. You have two alternative method available:
  - Using method getImportedKeys() within a loop. This should always work and is the default
  - Using method getCrossReferences() once. This might not work with all databases but, if it does, it will probably be much faster than the first method above. Simply try it out!
- **Include catalog or schema name in table name:** If checked, DevGui tries to include catalog and/or schema name in the generated XML file like 'admin.table1' instead of just 'table1'.
- **Set date format to:** As of DbForms 2.5, this is obsoleted by the format attributes. Ignore this field.
- **Generate XML button:** After you've supplied all of the information described above, click this button. DevGui then tries to read database meta data and generate a DbForms configuration file in XML format. The result is shown in the editor pane.
- **Save File:** Save the configuration in the editor pane to a file.

5.4.4. XSL Transformation



This tab can be used to automatically generate Java Server Pages that use the DbForms tag library by applying XSL stylesheets to a DbForms configuration file. To do that, you must have generated (or loaded) a dbforms configuration file, then you select the name of a style sheet and select button 'start transformation'.

- **Stylesheet Directory:** Location of the directory that contains your stylesheets that you wish to use.

  Defaults to:

  ```
  <DBFORMS_HOME>/xsl-stylesheets/
  ```

- **Use JavaScript Calendar:** If checked, the generated pages will include a JavaScript calendar for editing of date fields. See Chapter 18, *JavaScript Calendar Application* for more info. Unfortunately this calendar only seems to work with Internet Explorer, so by default this option is not checked.

Generated pages are written to the web application directory.

5.5. Use from command line or within a Java program

Normally the property file will be generated once in a GUI session. It can later be used in command line sessions to regenerate the `dbforms-config.xml` file after changes have been made to the database. This can be done from the command line as well as from a running Java program. To do so, follow these steps.

i. Update the property file, `dbforms_devgui.config`, that was generated during a GUI session

as necessary.

ii. Invoke DevGui via the command line

```
java org.dbforms.devgui.DevGui createconfigfile <propertyfile>
```

-or-

```
java org.dbforms.devgui.DevGui createconfigfile <propertyfile>
                                                 <outputfile>
```

Command Line parameters are as shown below

```
[-propertyfile <filename>]
```

The properties can be collected into a property file and specified on the command line. An example is shown below.

```
java org.dbforms.devgui.DevGui -propertyfile ./dbforms_devgui.config
```

Example

```
java org.dbforms.devgui.DevGui createconfigfile dbex.props dbex.xml
```

If the outputfile is not specified in the command line, it will be taken from the property file.

It is also possible to call the `main()` method of class DevGui from within a running Java program as in the following example.

```
String[] pars = {"createconfigfile","/home/sweethome/dbex.props"};
org.dbforms.devgui.DevGui.main(pars);
```

Command Line parameters can't be used with the `createconfigfile` program argument.

The DevGui functionality XSL transformation cannot currently be called via command line or a Java program.

5.6. Additional information

More documentation about DevGui can be found in the following two articles in the `articles/about_dbforms` directory.

- DbForms_DevGui.pdf

- XSL_DbForms.pdf

Both articles are still worth reading because they descibe ideas and principles behind DevGui. However, included information about needed libraries or the user interface is partially out of date, so please rely on the current User's Guide to install, run, and use the application.

Although DevGui is a useful tool, it is still possible that something may not work correctly when the information generated by DevGui is included in a web application. Errors in the `dbforms-config.xml` file often cause errors in the execution of the DbForms web application.

Whenever a change is made to the configuration file, it is wise to check the servlet engine log files for any errors listed there. In particular, if an error occurs in parsing the configuration file, it is often the case that the database connection is not processed since it occurs near the end of the file. This means that when the application is executed, a database error will be reported (usually mentioning name=null) when the real error occurred earlier in the configuration file.

Chapter 6. Jump-start a MS SQL Server

6.1. Introduction

```
Jump-start Instructions for installing dbforms on Windows 2000 with a MS
SQL Server Database

Authors:  Richard Bondi (rbondi@ergito.com)
          Matthew Stein (mstein@ergito.com)
Created:  27 JAN 2003
Version:  beta 1
```

This document was written to help you get dbforms up and running, no more; hence the "jump-start" in the title.

Dbforms can be thought of *essentially* as an open source add-on for the Tomcat servlet server that allows you to update, delete, insert, search, and browse database tables via jsp pages, without having to write these jsp pages. It is available at http://jdbforms.sourceforge.net.

This file contains instructions for creating a dbforms installation for MS SQL Server, for a single table that has no foreign keys, and an identity column. We wrote it because the existing instructions are spread out among different files, are missing for at least one important step, and do not cover problems specific to Microsoft SQL Server.

We use a pre-release version of dbforms because in separate document, we will describe how to use tables with foreign keys, and our solution only works with the pre-release version.

In a number of places in this document, we confess our ignorance with the disclaimer "but it works for us." We would be glad to be enlightened by future readers, and to improve this document accordingly.

Please email any comments or enhancements to Richard Bondi (rbondi@ergito.com).

Richard Bondi Matthew Stein

6.2. Requirements

```
Download all of these files (many of them in zips) to a temporary
location; you will be moving them to permanent locations as part of the
dbforms installation.

- Tomcat 4.0.6
  - downloadable from http://jakarta.apache.org, Tomcat project
  - use LE version, because you will already have j2sdk 2.0 v.1.4
    installed.

- j2sdk 2.0 v.1.4
  - http://java.sun.com
```

```
- dbforms 1.1.3pr1
  - downloadable from http://jdbforms.sourceforge.net

- Microsoft JDBC driver for Microsoft SQL Server.
  - downloadable from http://www.microsoft.com/sql/downloads

- commons-beanutils.jar
  - ver. 1.0
  - downloadable from http://jdbforms.sourceforge.net,
    click "Resources" under "Download"
  - downloadable from  http://jakarta.apache.org,
    Commons project

-commons-collections.jar
  - ver. 1.0
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://jakarta.apache.org, Commons project

- commons-digester.jar
  - ver. 1.0
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://jakarta.apache.org, Commons project

- commons-lang.jar
  - ver. 1.0
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://jakarta.apache.org, Commons project

- commons-logging.jar
  - ver. 1.0
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://jakarta.apache.org, Commons project

- commons-validator.jar
  - ver. 1.0
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://jakarta.apache.org, Commons project

- jakarta-oro-2.0.6.jar
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://jakarta.apache.org, ORO project

- log4j.jar
  - Implementation-Version: 1.1b2
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  -downloadable from http://jakarta.apache.org, Log4j project

- cos.jar
  - ver. ? (manifest file doesn't list version.)
  - downloadable from http://jdbforms.sourceforge.net, click "Resources"
    under "Download"
  - downloadable from http://www.oreilly.com
```

Finally, you should also have enough knowledge of XML not to go "huh?" when we write about "attrib-

utes", "elements" or "tags".

6.3. Installation

Installation steps are specified in the following subsections.

6.3.1. j2sdk 2.0 v1.4

```
You must use this version for these dbforms instructions, because of its
xml/xsl classes.
```

6.3.2. Install MS JDBC Driver

```
Install the Microsoft JDBC driver for Microsoft SQL Server:

- run the setup.exe.

  - Currently, when you run the install, it identifies itself as a
    Service Pack 1. That's ok, it includes the original JDBC driver
    too.
```

6.3.3. Install Tomcat and create "dbf" webapp

```
- Just run the friendly install program.

- Note: we couldn't get the NT service option to work: we could never
  start the service, so we suggest you skip it.

- Read and follow the RUNNING.txt file, included with the download,
  but also availabe at http://jakarta.apache.org under the Tomcat
  project (make sure you read the one for version 4.0.6).

- Edit the %CATALINA_HOME%\conf\server.xml file:
  - Find the first Context element
  - Add the following element:

    >Context path="/dbf" docBase="dbf" debug="0" reloadable="true"/<

    - "path" is a relative virtual path on the server (e.g. localhost/dbf)
    - "docBase" is the directory where the files are located; this is relative
      off of %CATALINA_HOME%\webapps, but you can specify full paths such
      as c:\mydir\here.
    - For more information, read the Tomcat docs.

- Create the directory specified in the "docBase" above.
- Create a sub-directory, WEB-INF
- Create a sub-directory, WEB-INF\lib
```

6.3.4. Create dbforms web.xml file

```
- Create a file called web.xml with the contents shown below, and put it
  in the WEB-INF directory you just created. [1]

!! WARNING !!
(You must manually remove any line breaks that occur in any
of the elements below. Formatting this file made them inevitable.)

:::::::::::::::::::::::::::: snip ::::::::::::::::::::::::::::::::

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- ============ DbForms Configuration Servlet ============ -->
  <servlet>
    <servlet-name>org.dbforms.ConfigServlet</servlet-name>
    <servlet-class>org.dbforms.ConfigServlet</servlet-class>
    <init-param>
      <param-name>dbformsConfig</param-name>
      <param-value>/WEB-INF/dbforms-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>log4j.configuration</param-name>
      <param-value>/WEB-INF/log4j.properties</param-value>
    </init-param>
    <init-param>
      <param-name>resourceBundle</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <init-param>
      <param-name>validation</param-name>
      <param-value>/WEB-INF/validation.xml</param-value>
    </init-param>
    <init-param>
      <param-name>validator-rules</param-name>
      <param-value>/WEB-INF/validator-rules.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>control</servlet-name>
    <servlet-class>org.dbforms.Controller</servlet-class>
    <init-param>
      <param-name>maxUploadSize</param-name>
      <param-value>80000</param-value>
    </init-param>
  </servlet>
  <!-- optional? Seems to work without this. rb, 1/23/2003.-->
  <taglib>
    <taglib-uri>/WEB-INF/dbforms.tld</taglib-uri>
    <taglib-location>/WEB-INF/dbforms.tld</taglib-location>
  </taglib>
</web-app>
```

[1] Please note that you will not find this file's full contents in the documentation. Different parts of the dbforms documentation contain parts of, but not all of it; an essential part is missing altogether and only in the mailing list archive.

```
:::::::::::::::::::::::::: snip ::::::::::::::::::::::::::::::

- Create a file index.html that contains simply "hello world" and put
  it in %CATALINA_HOME%\webapps\dbf.

- Start Tomcat (run %CATALINA_HOME%\bin\startup.bat), and attempt to
  load this file by going to

    http://localhost:8080/dbf/index.html

- If there is a problem, it is a Tomcat problem. You'll have to solve
  that on your own using all available Tomcat support resources. (Start
  with the file RUNNING.txt.)
```

6.3.5. Get what web.xml needs

```
A number of files are referred to in web.xml. We list them here by their
<param-name> names, and explain what to do about them.

* <param-name>dbformsConfig</param-name>
  We create this file below, with devgui.

* <param-name>log4j.configuration</param-name>
  Create a file called log4j.properties and put it into WEB-INF. It should
  have the following contents:

!! WARNING !!
(You must manually remove any line breaks that occur in any
of the # comments below. Formatting this file made them inevitable.)

:::::::::::::::::::::::::: snip ::::::::::::::::::::::::::::::

# Log4j Configuation file
# Created by Matthew Stein, 1/14/2003
# Based out of text in
# http://www.javaworld.com/jw-11-2000/jw-1122-log4j-p2.html

# First, set the "rootCategory" so everything gets logged if necessary.

# (For multiple logs beyond those specified here, use , name format.
log4j.rootCategory=ALL, A2, A3


# Define A2
#   Define writer
#   Define layout
log4j.appender.A2=org.apache.log4j.ConsoleAppender
log4j.appender.A2.Target=System.out
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n


#Define A3 [a/k/a A2 written to a file ]
#   Define writer
#   Define layout
log4j.appender.A3=org.apache.log4j.RollingFileAppender
log4j.appender.A3.File=/RFA.log log4j.appender.A3.MaxBackupIndex=10
log4j.appender.A3.MaxFileSize=1MB log4j.appender.A3.Append=True
log4j.appender.A3.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.A3.layout.ConversionPattern=%d{DATE} %F %p: %C(%x) - %m%n

:::::::::::::::::::::::::: snip ::::::::::::::::::::::::::::::

* <param-name>validation</param-name>,
  <param-name>validator-rules</param-name>
  See the Validation_Framework chapter for more information on this.

* <taglib-uri>/WEB-INF/dbforms.tld</taglib-uri>
  Put %DBFORMS_HOME%\dist\*.tld into WEB-INF


Copy required jars

- Copy these jars, downloaded previously, into WEB-INF\lib:
  - commons-beanutils.jar
  - commons-collections.jar
  - commons-digester.jar
  - commons-logging.jar
  - commons-validator.jar
  - commons-lang.jar
  - cos.jar
  - jakarta-oro-2.0.6.jar
  - log4j.jar

- Find the three jars that come with the MS SQL Server JDBC driver,and
  copy them into WEB-INF\lib. Their names are:
  - mssqlserver.jar
  - msbase.jar
  - msutil.jar

- Copy %DBFORMS_HOME%\dist\dbforms_v1_1_3pr1.jar into WEB-INF\lib
```

6.3.6. Prepare MS SQL Server

```
In this document, we will assume the following names for MS SQL Server;
just substitute those of your actual MS SQL Server installation:

server name:    birdserver
database name:  parrotDb
table name:     PLUMAGE_TABLE

The table PLUMAGE_TABLE has the following properties:
- you are willing to modify its rows
- it has an identity column
- it has no foreign keys.

(In a different document, we describe how to use dbforms with tables
that have foreign keys.)
```

6.3.7. Create MS SQL Server user

```
In order for dbforms to point to parrotDb, and not some other default db
like "master", we create a user to whom we assign parrotDb as the
```

```
default. (This isn't strictly necessary, see footnote [2]; but we chose to do it tha

We will create:

user on parrotDb:         dbfuser
password:                 dbfpwd
user's default database:  parrotDb

- Open Sql Enterprise Manager (*ii*)

- Click on the database parrotDb
- Double-click the Users icon on the right

- Right-click on the right pane and select New database user...
- Name: dbfuser
- SQL Server authentication password: dbfpwd

- Database: parrotDb  - database
- Roles:
  - public (assigned automatically)
  - db_datareader
  - db_datawrite
```

6.3.8. Generate dbforms-config.xml

```
- (optional) When you installed the Microsoft JDBC driver above,  it
  installed 3 jars (in the \lib directory underneath the  installation
  directory). Copy the three Microsoft JDBC jars to a simple, top level
  directory (like c:\java\jars or something)  now to avoid a horribly
  illegible classpath.

- Put the three MS JDBC jars into your classpath

- Set DBFORMS_HOME to the directory you installed dbforms to, e.g.
  c:\java\dbforms or whatever. [3]

(These steps are described more generally in the Readme_DevGui (no
```

[2] There are two reasons why you might check this. -First, it is a way to ensure that you are referring to the db you want, in this case "parrotDb". In this example, we chose to do this by creating a Sql Server user whose default database we can specify; that way, when Java connects to the db via JDBC, we guarantee that "parrotDb" will be used. Instead, you could choose any user (with appropriate permissions), not specify its default db, and instead explicitly state the database and user by checking "Include Schema name in table name." Unfortunately, we are baffled by what a Schema name is -- the list of them in devgui doesn't make sense to us. But that's our problem, not devgui's or Sql Server's. Second, if you generate dbforms pages for more than one "catalog" (a MS SQL Server database), then of course you have to specify the catalog of each table. Otherwise neither JDBC nor SQL Server will know which db to look for the table in. We haven't tried this, and we hope never to have to!

[3] Shortcut: Instead of right-clicking on My Computer and selecting Properties, hold down the Windows key and press the Break key. Don't forget that after making changes to environment variables only affect new "cmd" dos windows, not any that are already open.

extension) in the %DBFORMS_HOME% directory. I have left out the xml/xsl jars because j2sdk 2.0 1.4 already has stuff like that.)

- Run %DBFORMS_HOME%\bin\devgui.bat

You will see four tabs. Fill them out as follows: (Note that you can save all the info we are about to type in into a reloadable file, via the Project menu.)

- WebApps tab:
  - Location of Webapp: If for example %CATALINA_HOME% is c:\tc401, then this would be c:\tc401\webapps\dbf
  - Web-URL of Webapp: for the installation described in this document, this would be http://127.0.0.1:8080/dbf.

- Database tab:
    - JDBC driver class: this what is inside the Java call class.forName. You can find it inside the MS JDBC documentation. It is:

      com.microsoft.jdbc.sqlserver.SQLServerDriver

    - JDBC URL: Again, this is in the help file. It is a parameter of the Java JDBC DriverManager.getConnection call. It is:

      jdbc:microsoft:sqlserver://yourServerName:1433

      or in the example we are using:

      jdbc:microsoft:sqlserver://birdserver:1433

      (If your server is on another machine, you might want to use the full DNS name, e.g. birdserver.internal.pets4all.com, not just birdserver.)

  - Username and password:
    In our example, these are "dbfuser" and "dbfpwd".

    - Click Test connection.
    - If unsuccessful, double-check the above values, and that the MS JDBC jars are really in your classpath.

- XML Config tab
  This tab is for creating the dbforms-config.xml. [4]

  - DbForms config file:
    Specify the path and file name for the docBase you specified \WEB-INF\dbforms-config.xml If you don't have one yet (and you shouldn't, if your following these instructions for the first time)
      - click Browse
      - drill down to the directory specified in docBase\WEB-INF
      - type in "dbforms-config.xml" in the File Name box
      - click Open
      - ignore the error message.

    Alternatively, you can just type in the exact path and file name manually.

  - Examine:
    This is self-explanatory. Here you say whether you want to use dbforms web pages to update/insert/delete tables, and/or views, and/or system tables.

[4] You can actually call it whatever you want, since you specify the file's name in your web.xml file, in the <param-name> dbformsConfig</param-name>.

```
    - choose Tables.
    - Click the LOAD button.
  - Catalogs:
    In MS SQL Server, these are databases.
    - Click "in catalog with name"
    - select e.g. PLUMAGE_TABLE
  - Schemas:
    (Honestly, we don't know what these correspond to in MS SQL Server.
    Just do what we did:)
    - Choose dbo.
    - Check "in schema with name pattern."
  - Table names:
    (You can use the LIKE wild cards % and _. For this example:)
    - type PLUMAGE_TABLE for the table name
    - Check "with table name pattern."
  - Autocommit
    - Check "use autocommit". (We don't know why, but it works.)
  - Try to write standard types...
    - Leave this unchecked. (We don't know why, but it works.)

  - Foreign key detection
    - For this example, check "deactivated".

  - Include catalog/schema name in table name
    - Leave unchecked. (*v*)
  - Set Date Format to:
    - Umm... We don't know whether this is just for how dates are
      displayed, or for something else. We've treated it as a display-
      only choice; so choose your favorite.

  - Click Generate Xml.
  - Expand the window to see the generated xml.
  - Edit the generated xml as follows:
    - Find the
        fieldType="int identity"
      attribute. Change this to:
        fieldType="int"
    - If the same tag doesn't have an autoInc attribute, add:
        autoInc="true"
  - Delete any columns you don't want to display/edit via dbforms.
  - Click Save.
    This saves the xml to the xml into the file you specified
    at the top of the dialog, as
      the specified docBase\WEB-INF\dbforms-config.xml

- XSL Transformation tab
  In devgui, click the fourth tab, XSL Transformation. From this dialog,
  DevGui simply transforms the dbforms-config.xml file with each of the
  xsl style sheets listed. Each style sheet produces a different jsp or
  set of jsps. [5]
  That's the beauty of dbforms: if you have 20 tables, you don't have to
  lose your mind generating custom jsps for each one; instead, you come
  to this dialog and generate them all.

  - Stylesheet directory:
    This will already be set to where the xsls are in the dbforms
    directory.
  - Use JavaScript Calendar
    May or may not work on your browser. Allows you to select a date
```

[5] If you know xsl, feel free to edit them to your heart's delight, and store your edited ones in a different directory: just select that directory here before you start transforming.

```
    from a pop-up calendar instead of typing in the date. Check if
    you're feeling moderately adventurous.
  - For each xsl stylesheet listed:
    - click on it to highlight it
    - click "start transformation!"

  - Do *not* edit and/or save the xsl stylesheets displayed at the
    bottom of the dialog.
```

6.4. Use it

```
- Stop and start tomcat
- Point your browser to:

  http://localhost:8080/dbf/menu.jsp

- If you get a crash, go to the DOS window that opens whenever you
  launch  Tomcat. Scrutinize all the errors and other logging in it.
  Your log4j.properties file is maximizing the logging information.

  This is always the place to start when trying to debug dbforms.

- Click View Normal button

  Note that the next page will take a long time to load. This will be
  true  of all first pages while Tomcat compiles them for the first
  time.

  You should see all rows of the table PLUMAGE_TABLE.

- Click all the buttons on menu.jsp; you should be able to figure out
  what they do from there.
```

6.5. Conclusion

This document was written to help you get dbforms up and running, no more; hence the "jump-start" in the title. For example,if you want to know how to display more rows at a time in Edit Range, we refer you to the dbforms documentation. Or to configure log4j to customize logging, see the log4j documentation.

Once again, please email all suggestions and improvements for this document to rbondi@ergito.com.

Part III. Basic Use Tutorial

Table of Contents

Chapter 7. Tutorial: A sample application

7.1. Application requirements

Let's imagine we're supposed to write a little Customer Care application for a company which sells services to customers. The application is used mainly by sales people.

The users of this application should be able to perform the following tasks:

- Add and/or modify services that are sold to customers
- Add and/or modify customer information
- Input customer orders
- Register customer complaints
- Assign a 'priority level' to a customer complaint

The application should be capable of supporting concurrent users and should be web-based. Also, let's imagine our boss wants this application done in say, two days! No problem! As you'll see, we will complete this application in less than two hours. What to do with the rest of the time allocated to this project? Take a mini vacation.

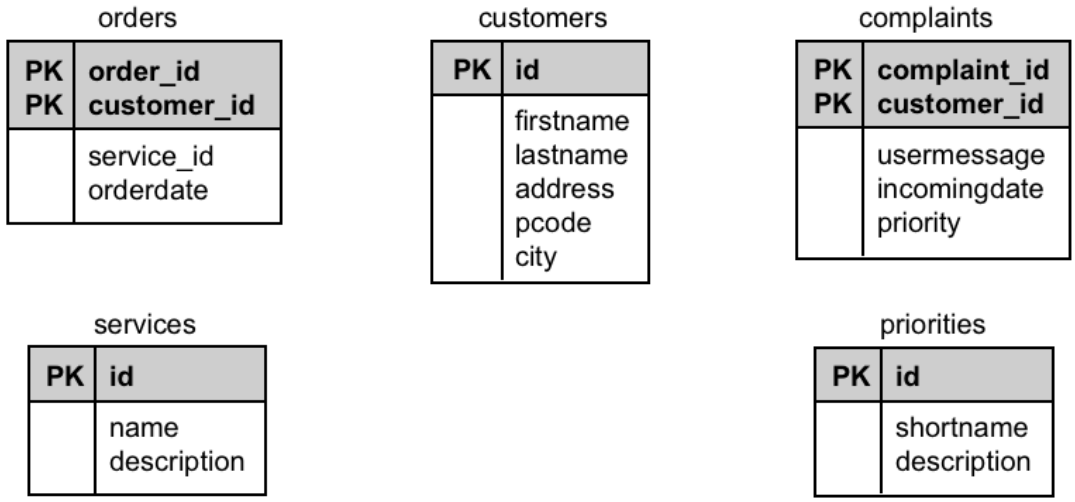
7.2. Defining the database model and database connection

First of all, we will design the database model. (If a database model already existed, then we could skip this task and simply re-use the existing model). DbForms has no special requirements for the database model. Here is a simple model which addresses the requirements defined above.

7.2.1. Conceptual design

The following database model will be used in this tutorial.

Figure 7.1. Conceptual diagram of the database



7.2.2. Physical table creation

Here is the SQL code for creating the database model shown in the preceeding section. The example is for a MySQL database.

Example 7.1. SQL code for creating the database tables

```
CREATE TABLE services
(
  id          int         NOT NULL PRIMARY KEY,
  name        char (30)   NULL,
  description char (255) NULL
);

CREATE TABLE customers
(
  id        int       NOT NULL PRIMARY KEY,
  firstname char (50) NULL,
  lastname  char (50) NULL,
  address   char (30) NULL,
  pcode     char (10) NULL,
  city      char (40) NULL
);

CREATE TABLE orders
(
  order_id     int       AUTO_INCREMENT,
  customer_id  int       NOT NULL,
  service_id   int       NULL,
  orderdate    char (20) NULL,
  PRIMARY KEY ( order_id, customer_id )
);

CREATE TABLE complaints
(
  complaint_id int         AUTO_INCREMENT,
  customer_id  int         NOT NULL,
  usermessage  char (255) NULL,
```

```
    incomingdate char (20)  NULL,
    priority     int        NULL,
    PRIMARY KEY ( complaint_id, customer_id )
  );

  CREATE TABLE priorities
  (
    id          int        NOT NULL PRIMARY KEY,
    shortname   char (12)  NULL,
    description char (100) NULL
  );
```

Remarks

It would be desirable to add referential integrity, by adding foreign keys and/or triggers) but to keep things simple for this tutorial on DbForms capabilities, we have not included them.

7.2.3. Defining a database description for DbForms

The next step is to create the XML-based description of this model, as follows.

Example 7.2. Defining tables in `dbforms-config.xml`

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dbforms-config>

  <table name="services">
    <field name="id"          fieldType="int" isKey="true" />
    <field name="name"        fieldType="char" />
    <field name="description" fieldType="char" />
  </table>

  <table name="customers">
    <field name="id"        fieldType="int" isKey="true" />
    <field name="firstname" fieldType="char" />
    <field name="lastname"  fieldType="char" />
    <field name="address"   fieldType="char" />
    <field name="pcode"     fieldType="char" />
    <field name="city"      fieldType="char" />
  </table>

  <table name="orders">
    <field name="order_id"    fieldType="int" isKey="true"
                              autoInc="true" />
    <field name="customer_id" fieldType="int" isKey="true" />
    <field name="service_id"  fieldType="char" sortable="true"/>
    <field name="orderdate"   fieldType="char" sortable="true"/>
  </table>

  <table name="complaints">
    <field name="complaint_id" fieldType="int" isKey="true"
                               autoInc="true" />
    <field name="customer_id"  fieldType="int" isKey="true" />
    <field name="usermessage"  fieldType="char" />
    <field name="incomingdate" fieldType="char" sortable="true"/>
    <field name="priority"     fieldType="int"  sortable="true"/>
  </table>
```

```
  <table name="priorities">
    <field name="id"          fieldType="int" isKey="true" />
    <field name="shortname"   fieldType="char" />
    <field name="description" fieldType="char" />
  </table>

<!--
  <dbconnection
    name = "java:comp/env/jdbc/tutorial"
    isJndi = "true"
  />
-->
   <dbconnection
     isPow2 = "true"
     connectionProviderClass =
       "org.dbforms.conprovider.SingleConnectionProvider"
     name =
       "jdbc:hsqldb:$(SERVLETCONTEXT_REALPATH)/WEB-INF/db_hsql/tutorial"
     isJndi = "false"
     conClass = "org.hsqldb.jdbcDriver"
     username = "sa"
     password = ""
   />

</dbforms-config>
```

Remarks

There exists a DbForms tool which does all the painful work of transferring a database schema into XML format. See Chapter 5, *DevGui* for more information.

This example includes samples of two **`db:dbconnection`** elements. Refer to Chapter 2, *DbForms Concepts* for a discussion of the dbconnection element. The example that is not commented out uses an in-memory HSQL database that requires no setup time to run.

7.3. Structure and screen flow of the application

To fulfill the requirements, we need to implement at minimum the following forms.

Table 7.1. Core requirements for our sample application

Edit services table	Maintain an inventory of the services the company offers.
Customer orders	Maintain a record of customer orders.
Customer complaints	Maintain a record of customer complaints.

In addition to developing a page for each of the core requirements, we'll implement the following pages for user convenience.

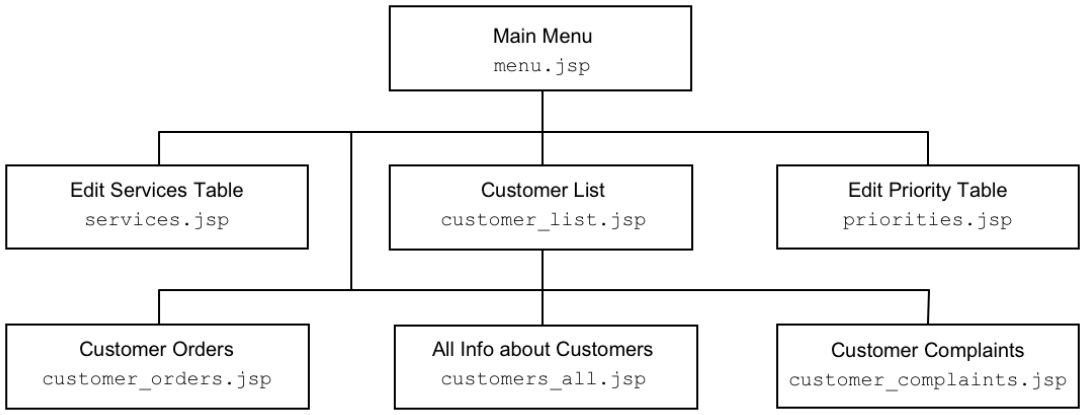
Table 7.2. Additional features for our sample application

Main Menu	The point of entry for the application that links to all other pages.
Edit priority table	Every complaint must be assigned a priority level. This page allows the user to edit the valid priority codes that may be used in the com-

	plaints page.
Customer list	This page provides a listing (overview) of all customers and enables the user to select a customer and jump to the appropriate input page.
All information about customers	This page shows all orders and all complaints for a given customer on one page.

The user should be able to navigate from page to page as shown in the screen flow diagram below.

Figure 7.2. Screen flow of our application



7.4. Implementing the forms

The following sections will demonstrate how to create our sample application using some basic and some more advanced elements of DbForms.

For each form we are going to discuss here, we have included the JSP source code containing the DbForms elements followed by a representative screenshot of the resulting HTML page.

We suggest installing this tutorial application from the `examples/tutorial` directory in the distribution and taking a detailed look at it. You will quickly find out how the elements behave and which pattern of combined DbForms elements is best for each purpose. Instructions for quickly installing the tutorial are described in Section 7.5, “Installing the tutorial application” .

For more detailed information about each DbForms custom element, please refer to the DbForms Custom Tag Library document in the `docs/taglib` directory. This document contains a complete description of all DbForms tags and their attributes.

7.4.1. Main Menu

The main menu's only purpose is to link to the other pages. Of course, this could be done by simply coding html hyperlinks, but this section demonstrates how to use an empty element and how to use gotoButton elements.

Example 7.3. Code for `menu.jsp`

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<html>
<head>
  <db:base/>
</head>

<body>

  <h1>Menu</h1>

  <db:dbform followUp="/menu.jsp">
  <center>
  <p>
    <db:gotoButton caption="Edit Services Table"
                   destination="/services.jsp"/>
  </p>
  <p>
    <db:gotoButton caption="Edit Priority Table"
                   destination="/priorities.jsp"/>
  </p>
  <p>
    <db:gotoButton caption="Customer List"
                   destination="/customer_list.jsp"/>
  </p>
  <p>
    <db:gotoButton caption="Browse Customer Orders"
                   destination="/customer_orders.jsp"/>
  </p>
  <p>
    <db:gotoButton caption="Browse Customer Complaints"
                   destination="/customer_complaints.jsp"/>
  </p>
  <p>
    <db:gotoButton caption="All Info about Customers"
                   destination="/customer_all.jsp"/>
  </p>
  </center>
  </db:dbform>

</body>
</html>
```

Remarks

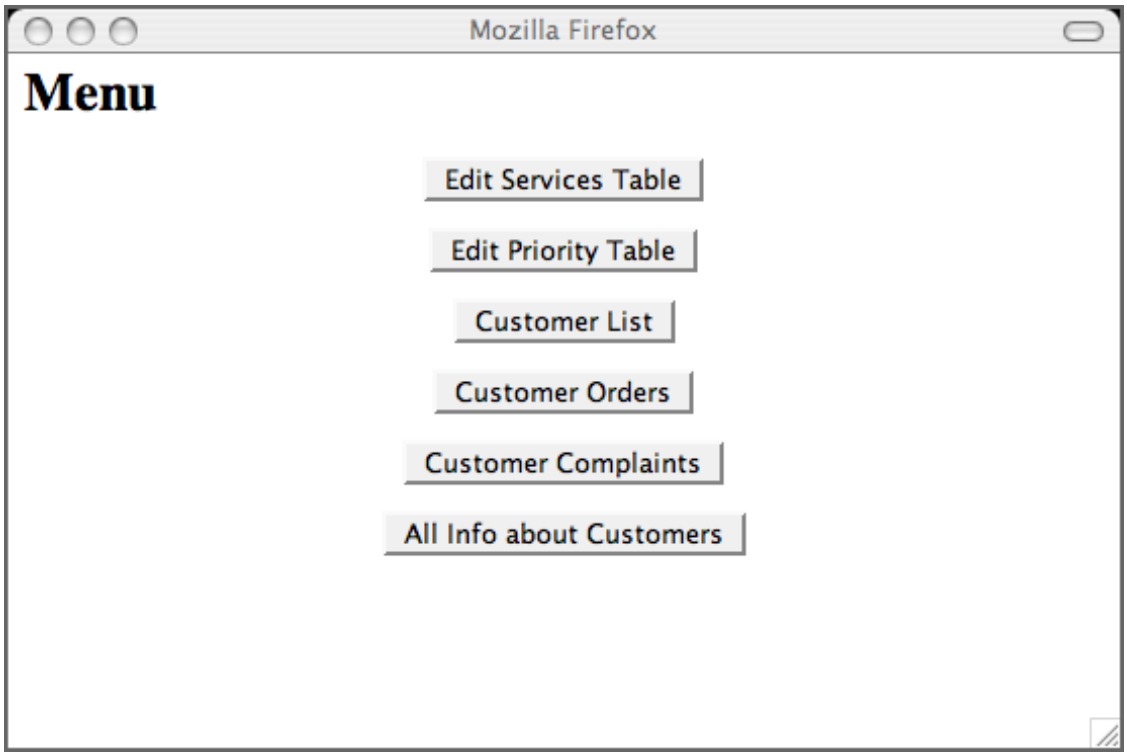
The **<%@taglib uri="/WEB-INF/dbForms.tld" prefix="db" %>** element refers to the Taglib Descriptor (tld) of DbForms and defines the prefix used to identify the elements of the tag library. (In our sample application we are using **db**, but you could use a different prefix.)

We used the following attributes of **db:gotoButton**:

- **caption**: the caption of the button
- **destination**: the URL of the page to jump to

The **db:base** element should be included in every JSP that contains DbForms elements. It ensures that images, Cascading Style Sheets and relative links to other pages are found.

Figure 7.3. Main menu (menu.jsp)



7.4.2. Services

This page enables the user to administer an inventory of the services the company sells to its customers. The user will get a list of all the existing services along with text fields and buttons to update and delete rows. Finally an empty input form for inserting new services is also presented.

Example 7.4. Code for `services.jsp`

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<html>
<head>
  <db:base/>
</head>

<body>
  <db:errors/>

  <db:dbform tableName="services" maxRows="*"
             followUp="/services.jsp">

    <db:header>
      <db:gotoButton caption="Menu" destination="/menu.jsp" />
      <h1>Services We Provide</h1>
      <center>
        <h3>Existing Service Definitions</h3>
```

```
    </center>

    <table border="5" width="60%" align="CENTER">
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Description</th>
      <th>Actions</th>
    </tr>
  </db:header>

  <db:body>
  <tr>
    <td>
      <db:textField fieldName="id" size="5"/>
    </td>
    <td>
      <db:textField fieldName="name" size="20"
                    maxlength="30"/>
    </td>
    <td>
      <db:textField fieldName="description" size="24"
                    maxlength="255"/>
    </td>
    <td>
      <db:updateButton caption="Update"/>
      <db:deleteButton caption="Delete"/>
    </td>
  </tr>
  </db:body>

  <db:footer>
  </table>

  <center><h3>Enter New Service Definition</h3></center>

  <table align="center" border="3">
  <tr>
    <td>Id</td>
    <td>
      <db:textField size="5" fieldName="id"/>
    </td>
  </tr>
  <tr>
    <td>Name</td>
    <td>
      <db:textField size="20" maxlength="30" fieldName="name"/>
    </td>
  </tr>
  <tr>
    <td>Description</td>
    <td>
      <db:textArea rows="4" cols="20" wrap="virtual"
                   fieldName="description"/>
    </td>
  </tr>
  </table>
  <br>
  <center>
    <db:insertButton caption="Store New Service Definition"
                     showAlways="true"/>
  </center>
  </db:footer>
</db:dbform>
```

```
</body>
</html>
```

Remarks

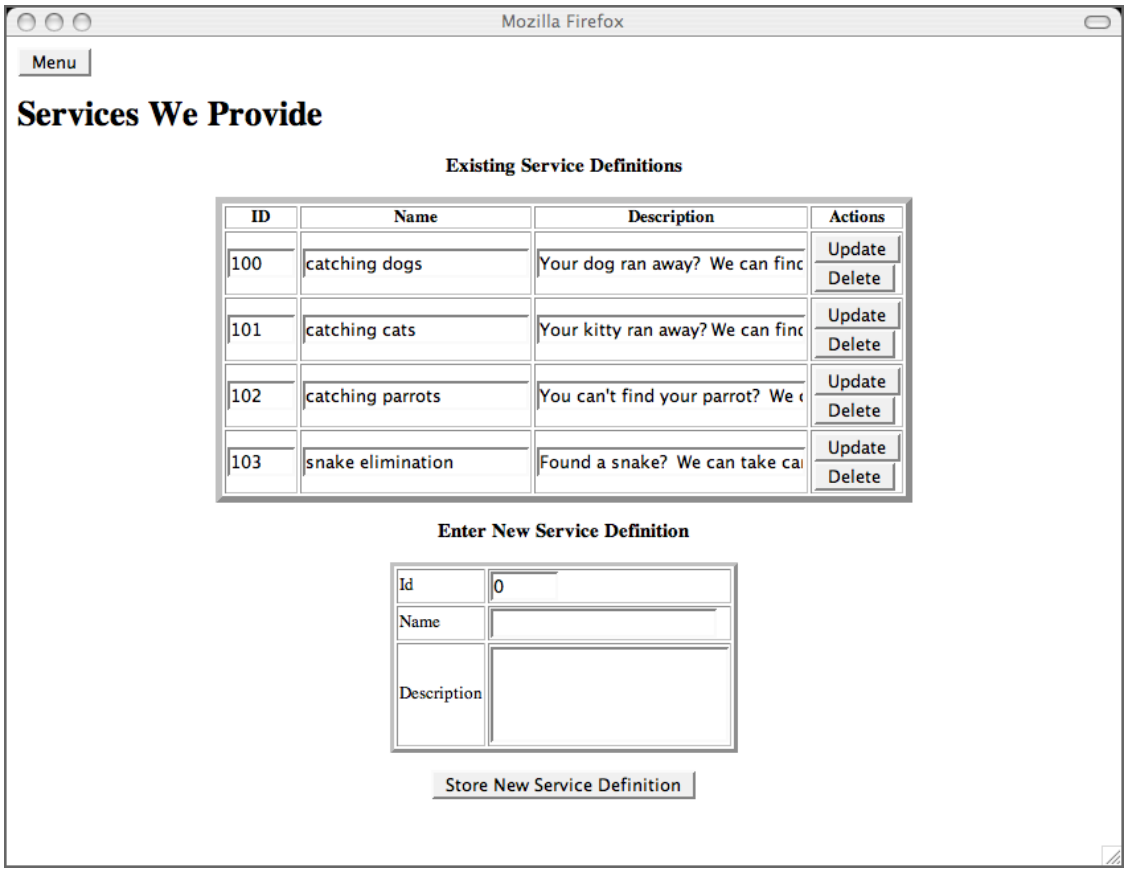
There is one HTML table for headings and rows of data. The **td** elements that define the column headings are in the **db:header** element because the header is rendered only once on the page. The **td** elements that define the rows of data are in the **db:body** element becase the row will be rendered many times if multiple rows of data are retrieved from the database.

maxRows is set to ***** which is equivalent to 'get all rows'. If the data requested contains a great number of rows, we might consider setting maxRows to 10, 20 or another limited number. If so, we would instantiate navigation buttons for scrolling between the pages. We will use that pattern later on.

The **db:errors** element shows a list of errors, if any occurred (i.e. duplicate key error, etc.). The placement of this element determines where the messages will be displayed.

The **db:updateButton** and **db:deleteButton** elements are placed in the body and are therefore rendered for each row.

Figure 7.4. Managing services (services.jsp)



7.4.3. Priorities

This page enables the user to manage a list of priorities we will use this data later in the `complaints` page where we will prompt the user to select the appropriate priority for a user's complaint.

Example 7.5. Code for `priorities.jsp`

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<html>
<head>
  <db:base/>
</head>

<body>
<db:errors/>

<db:dbform tableName="priorities" maxRows="*"
  followUp="/priorities.jsp" autoUpdate="true">

  <db:header>
    <db:gotoButton caption="Menu" destination="/menu.jsp" />
    <h1>Priority Definitions</h1>
    <center><h3>Existing Priority Definitions</h3></center>
    <table border="5" width="60%" align="CENTER">
    <tr>
      <th>Id</th>
      <th>Short Name</th>
      <th>Description</th>
      <th>Actions</th>
    </tr>
  </db:header>

  <db:body>
    <tr>
      <td><db:label fieldName="id"/></td>
      <td><db:textField fieldName="shortname"/></td>
      <td><db:textField fieldName="description"/></td>
      <td>
        <db:updateButton caption="Update"/>
        <db:deleteButton caption="Delete"/>
      </td>
    </tr>
  </db:body>

  <db:footer>
    </table>

    <center><h3>Enter New Priority Definition</h3></center>

    <table border="3" align="center">
    <tr>
      <td>Id</td>
      <td><db:textField size="3" fieldName="id"/></td>
    </tr>
    <tr>
      <td>Short-Name</td>
      <td><db:textField fieldName="shortname"/></td>
    </tr>
    <tr>
      <td>Description</td>
      <td><db:textArea rows="3" cols="20"
                       fieldName="description"/></td>
```

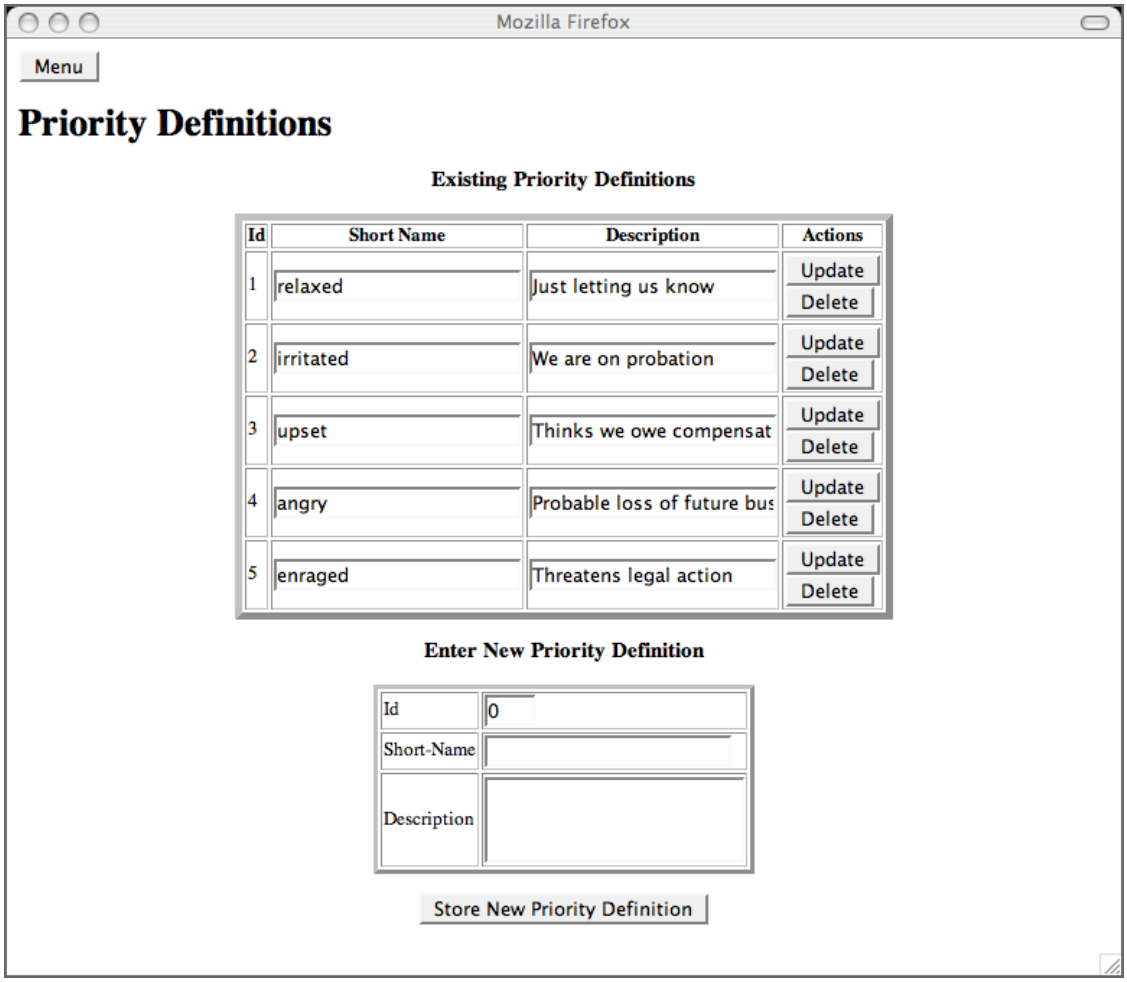
```
    </tr>
    </table>

    <br>
    <center>
      <db:insertButton caption="Store New Priority Definition"
                       showAlways="true"/>
    </center>
  </db:footer>

</db:dbform>
</body>
</html>
```

Remarks

As you may have noticed, we have used a pattern for this page that is similar to the pattern for `services.jsp`.

The only real difference can be found in the **`autoUpdate`** attribute in the **`db:form`** element, which is set to **`true`** in `priorities.jsp`. This means, that all rows will be updated if their text fields (and other data-sensitive elements such as pick lists) have been edited by the user. For example, if the user edits data in priorities numbered 1, 2 and 3 and then clicks a single update (or any other action button), **all** changes will be stored **automatically** in the database! (These automatically triggered events are called *implicit*, or *secondary* events.)x

Figure 7.5. Defining priority levels (priorities.jsp)



7.4.4. Customer list

This convenience page provides the user with a listing of all customers. The user is able to select a customer from the list and perform a given operation (edit orders, edit complaints submitted by the customer). The user will be able to delete existing customers, as well.

Example 7.6. Code for `customer_list.jsp`

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<html>
<head>
  <db:base/>
</head>
<body>
  <db:errors/>

  <db:dbform tableName="customers" maxRows="*"
             followUp="/customer_list.jsp"
             autoUpdate="false">

    <db:header>
```

```
    <db:gotoButton caption="Menu" destination="/menu.jsp" />
      <h1>Customer List</h1>
      <table align="center" cellspacing="6">
      <tr>
        <td><b></b></td>
        <td><b>First Name</b></td>
        <td><b>Last Name</b></td>
        <td><b>Address</b></td>
        <td><b>P-Code</b></td>
        <td><b>City</b></td>
        <td><b>Action</b></td>
      </tr>
    </db:header>

    <db:body allowNew="false">
      <tr>
        <td><db:associatedRadio name="r_customerkey" /></td>
        <td><db:label fieldName="firstname"/></td>
        <td><db:label fieldName="lastname"/></td>
        <td><db:label fieldName="address"/></td>
        <td><db:label fieldName="pcode"/></td>
        <td><db:label fieldName="city"/></td>
        <td><db:deleteButton caption="delete"/></td>
      </tr>
    </db:body>

    <db:footer>
      </table>
      <p>
      <center>

      Show

      <db:gotoButton caption="Orders"
                     destination="/customer_orders.jsp"
                     destTable="customers"
                     keyToKeyToDestPos="r_customerkey"/>

      <db:gotoButton caption="Complaints"
        destination="/customer_complaints.jsp"
        destTable="customers"
        keyToKeyToDestPos="r_customerkey"/>

      <db:gotoButton caption="All Information"
                     destination="/customer_all.jsp"
                     destTable="customers"
                     keyToKeyToDestPos="r_customerkey"/>

      of the selected customer!

      </center>
      </p>
    </db:footer>

  </db:dbform>
</body>
</html>
```

Remarks

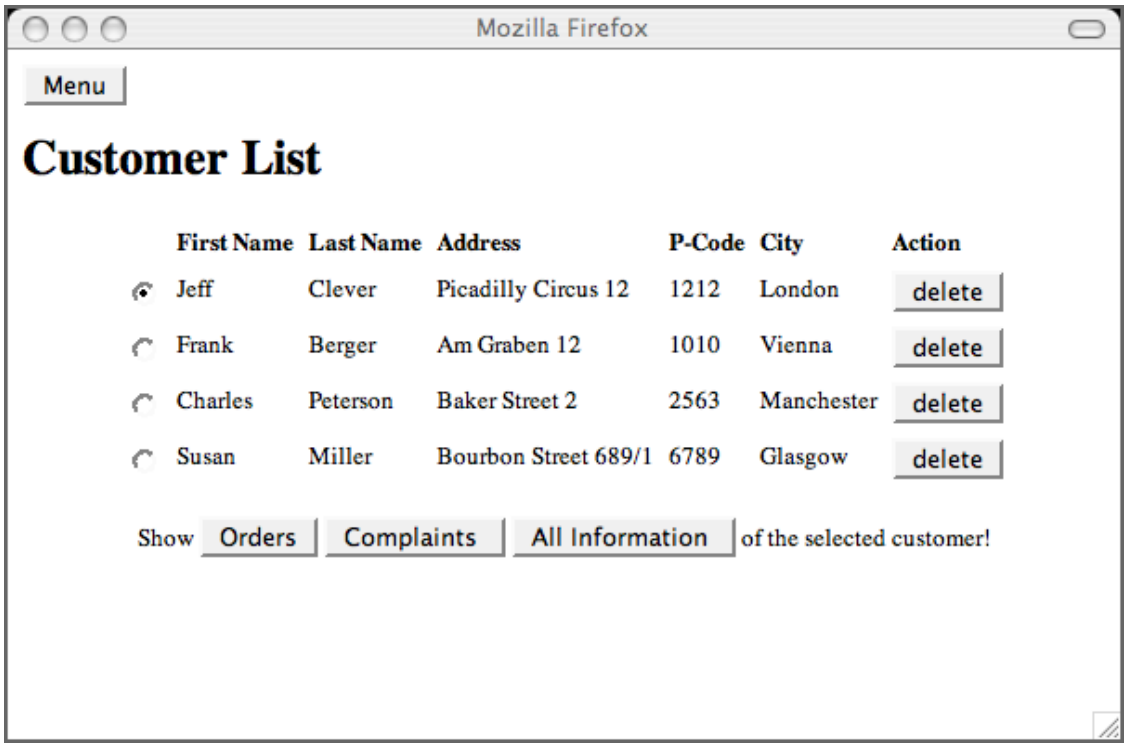
In this page, we introduce a new pattern, the use of **`db:associatedRadio`** elements. To demonstrate the difference between this new pattern and the old pattern, we have used *both* patterns in the customer list page. The delete buttons follow the old pattern and the buttons for orders, complaints and all inform-

ation are implemented using the new pattern. Using **db:associatedRadio** elements to mark a row for certain actions (in our case navigating to orders, complaints and all information) saves a lot of space and makes for a cleaner interface. If we had to include a button for all possible actions, the page would not look very user-friendly.

You may have noticed that the **db:gotoButton** element has a strange sounding attribute **keyToKeyToDestPos**. The value of this attribute is generated by DbForms and links the row in the page that is associated with the radio button with the corresponding row in the Customers table. It refers to the primary key in the Customers table and is used to identify the row the system will jump to when the user clicks an action button.

Another interesting item is that the attribute **allowNew** in the db:body element is set to a value of **false**. This has the effect, in the case of an empty result set (no customers), of bypassing the body element altogether.

Figure 7.6. List of customers (customer_list.jsp)



7.4.5. Orders

This page provides the user with the ability to manage new orders for a given customer. The user is able to edit customer orders, as well as customer data. The user should not be required to 'struggle' with item numbers, but instead, should be able to simply select (from a drop down select box) the services the customer has indicated they wish to purchase.

Figure 7.7. Underlying data for customer_orders.jsp

Example 7.7. Code for `customer_orders.jsp`

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<html>
<head>
  <db:base/>
</head>

<body>

  <!-- show any database errors here -->
  <db:errors/>

  <db:dbform tableName="customers" maxRows="1"
             followUp="/customer_orders.jsp" autoUpdate="false">

  <db:header>
    <db:gotoButton caption="Menu" destination="/menu.jsp" />
    <h1>Customer Orders</h1>
  </db:header>

  <db:body>
    <table align="center">
    <tr>
      <td>Id</td>
      <td>
        <db:textField fieldName="id" size="4"/>
      </td>
    </tr>
    <tr>
      <td>First Name</td>
      <td>
        <db:textField fieldName="firstname" size="18"/>
      </td>
    </tr>
    <tr>
      <td>Last Name</td>
      <td>
        <db:textField fieldName="lastname" size="18"/>
      </td>
    </tr>
    <tr>
      <td>Address:</td>
      <td>
        <db:textField fieldName="address" size="25" />
      </td>
    </tr>
    <tr>
      <td>Postal Code - City</td>
      <td>
        <db:textField fieldName="pcode" size="6"/> -
        <db:textField fieldName="city" size="16"/>
      </td>
    </tr>
    </table>
    <br>

    <!-- table embedding the sub form -->
    <table align="center" border="1">
    <tr>
      <td>
```

```
<center><p><b>Orders</b></p></center>

<!-------- sub form begin ------->
<db:dbform tableName="orders" maxRows="2"
           parentField="id" childField="customer_id"
           followUp="/customer_orders.jsp"
           autoUpdate="false">

  <db:header>
    <!-- Show existing orders of services for the customer -->
    <table width="100%">
    <tr>
      <td width="40"></td>
      <td>Service</td>
      <td>Order Date</td>
    </tr>
  </db:header>

  <db:body allowNew="true">
    <tr>
      <td width="40">
        <db:associatedRadio name="radio_order"/>
      </td>
      <td>
        <db:select fieldName="service_id">
          <db:tableData
            name          = "our_services"
            foreignTable  = "services"
            visibleFields = "name"
            storeField    = "id"
          />
        </db:select>
      </td>
      <td>
        <db:textField fieldName="orderdate" size="14"/>
      </td>
    </tr>
  </db:body>

  <db:footer>
    <tr>
      <td colspan="3" align="CENTER">
        <db:updateButton caption="Update Order"
                         associatedRadio="radio_order"/>
        <db:deleteButton caption="Delete Order"
                         associatedRadio="radio_order"/>
        <db:insertButton caption="Store New Order"
                         showAlways="false" />
        <db:navNewButton caption="New Order"
                         showAlwaysInFooter="false" />
      </td>
    </tr>
    </table>

    <br>
    <center>
      <db:navFirstButton caption="&lt;&lt; First" />
      <db:navPrevButton caption="&lt; Previous" />
      <db:navNextButton caption="Next &gt;" />
      <db:navLastButton caption="Last &gt;&gt;" />
    </center>

  </db:footer>
```

```
        </db:dbform>
        <!-------- sub form end -------->

      </td>
    </tr>
    </table>
    <!-- end of table embedding the sub form -->

    <br>
    <center>
      <db:insertButton caption="Store New Customer" />
      <db:updateButton caption="Update Customer" />
      <db:deleteButton caption="Delete Customer" />
      <db:navNewButton caption="New Customer"
                       showAlwaysInFooter="false" />
    </center>
  </db:body>

  <db:footer>
    <br>
    <center>
      <db:navFirstButton caption="&lt;&lt; First" />
      <db:navPrevButton caption="&lt; Previous" />
      <db:navNextButton caption="Next &gt;"  />
      <db:navLastButton caption="Last &gt;&gt;" />
    </center>
  </db:footer>

  </db:dbform>
</body>
</html>
```

Remarks

In this page, we have introduced several new DbForm features.

Nested Forms: The structure of this page is similar to the structure shown in Section 2.2.2.1, “The structure of a DbForms View” where a main form has a subform in its body. The subform is linked to its parent by the equality of one or more data fields defined in the child form's **`parentField`** and **`childField`** attributes. If there is more than one field to define for correct mapping, a field list may be provided, with each field separated from the other by commas or semicolons.

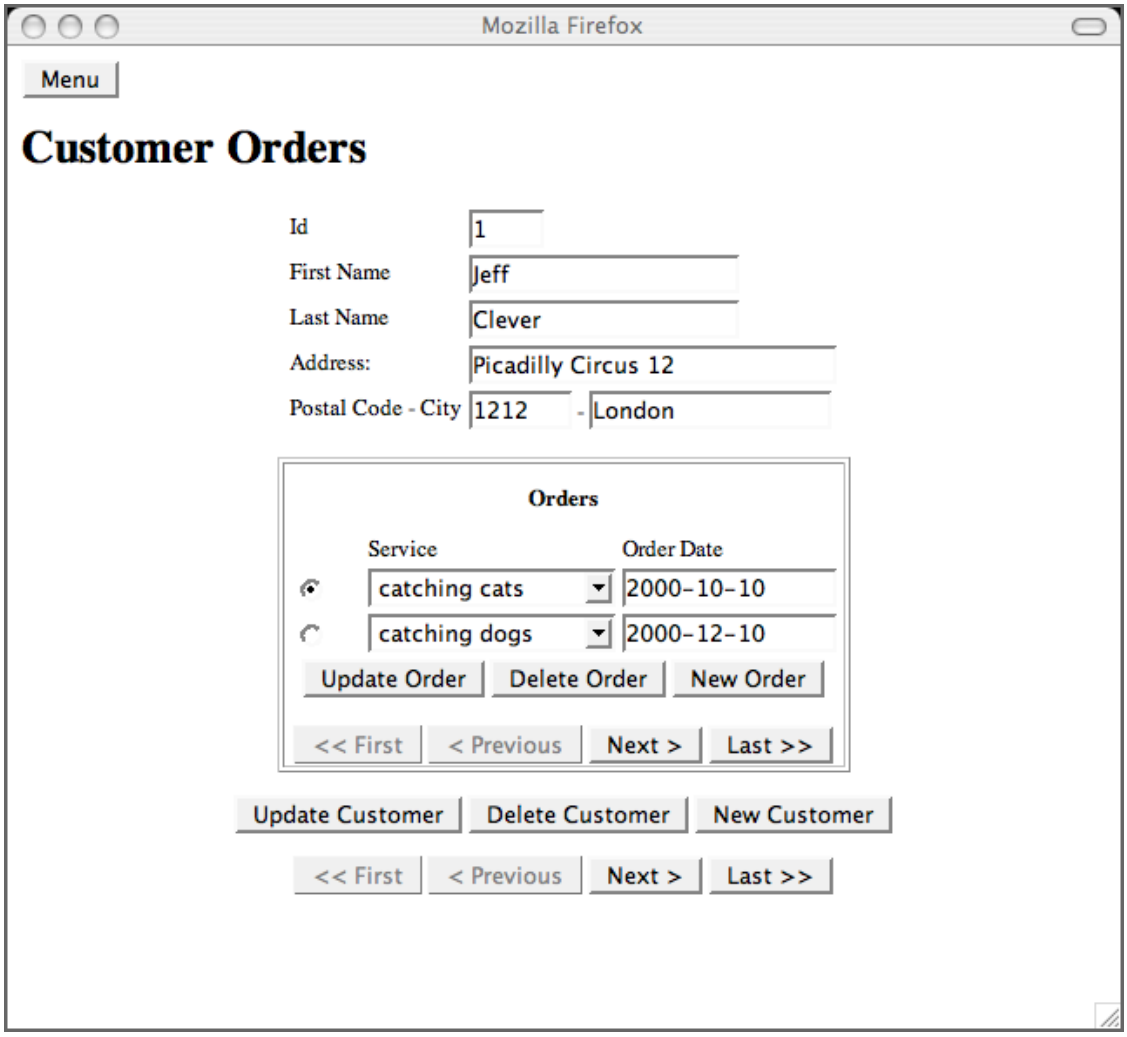
Select Element: In addition to **`db:textField:`** and **`db:textArea`** elements, more complex elements like **`db:select`**, **`db:radio`** and **`db:checkbox`** can be used for data visualization and manipulation. The **`db:select`** element to allows the user to choose the type of services available from a pulldown list.

External data fetched by a **`db:tableData`** element: This element provides external data for **`db:radio`**, **`db:checkbox`** or **`db:select`** elements. It may be used for cross references to other tables. In our case, we initialized the select box with external data from the service table. Be aware that you have to distinguish between the field(s) to be *shown* to the user and the field to be *stored* in the associated field in the table. In our case we *showed* the field `service.name` and *stored* the value `service.id` in the associated field `orders.service_id`! The name **our_services** was defined to enable internal caching of the data, which increases performance.

Navigation Buttons: Because only *one* customer is visible at once, the user needs a means of navigating between records. This functionality is provided by the **`db:navFirstButton`**, **`db:navLastButton`**, **`db:navPrevButton`**, and **`db:navNextButton`** elements. In addition, using the second set of navigation buttons, the user can navigate through a list of orders if there are more than the maxRows that are displayed in the subform.

The **db:navNewButton** element navigates the user to an empty form. This form is automatically created by DbForms. It is the same as this page, but the fields are not populated and the insert button that we defined in this JSP is present but the update and delete buttons are not present.

Figure 7.8. Managing customers and their orders on a single page (customer_orders.jsp)



7.4.6. Complaints

This page is conceptually similar to the previous JSP (`customer_orders.jsp`). It provides the user with the ability to manage customer complaints. The user is, once again, able to edit customer complaints as well as customer data, all from the same page.

To establish an order of importance, every reported complaint must be associated with a priority level. To make life easier for our users, they are able to simply pick a priority level from a maintainable list of predefined priority levels.

Figure 7.9. Underlying data for `customer_complaints.jsp`

Example 7.8. Code for `customer_complaints.jsp`

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<html>
<head>
  <db:base/>
</head>

<body>

  <!-- show any database errors here -->
  <db:errors/>

  <db:dbform tableName="customers" maxRows="1"
    followUp="/customer_complaints.jsp" autoUpdate="false">

  <db:header>
    <db:gotoButton caption="Menu" destination="/menu.jsp" />
    <h1>Customer Complaints</h1>
  </db:header>

  <db:body>
    <table align="center">
      <tr>
        <td>Id</td>
        <td>
          <db:textField fieldName="id" size="4"/>
        </td>
      </tr>
    <tr>
      <td>First Name</td>
      <td><db:textField fieldName="firstname" size="18"/></td>
    </tr>
    <tr>
      <td>Last Name</td>
      <td><db:textField fieldName="lastname" size="18"/></td>
    </tr>
    <tr>
      <td>Address:</td>
      <td><db:textField fieldName="address" size="25" /></td>
    </tr>
    <tr>
      <td>Postal Code - City</td>
      <td>
        <db:textField fieldName="pcode" size="6"/> -
        <db:textField fieldName="city" size="16"/>
      </td>
    </tr>
    </table>
    <br>

    <!-- table embedding the sub form -->
    <table align="center" border="1">
    <tr>
      <td>
```

```
<center><p><b>Complaints</b></p></center>

<!-------- sub form begin ------->
<db:dbform tableName="complaints" maxRows="2"
           parentField="id" childField="customer_id"
           followUp="/customer_complaints.jsp"
           autoUpdate="false">

<db:header>
  <!-- Show existing complaints of the customer -->
  <table>
  <tr>
    <td width="40"></td>
    <td valign="top">User's Message</td>
    <td>Priority<br/><db:sort fieldName="priority"/></td>
    <td>Incoming Date<br/><db:sort
                           fieldName="incomingdate"/></td>
  </tr>
</db:header>

<db:body allowNew="true">
  <tr>
    <td width="40" valign="top">
      <db:associatedRadio name="radio_complaint" />
    </td>
    <td valign="top">
      <db:textArea fieldName="usermessage"
                   cols="32" rows="3" wrap="virtual"/>
    </td>
    <td valign="top">
      <db:select fieldName="priority">
        <db:tableData
          name          = "some_priorities"
          foreignTable  = "priorities"
          visibleFields = "shortname"
          storeField    = "id"
        />
      </db:select>
    </td>
    <td valign="top">
      <db:textField fieldName="incomingdate" size="14"/>
    </td>
  </tr>
</db:body>

<db:footer>
  <tr>
    <td colspan="4" align="CENTER">
      <db:updateButton caption="Update Complaint"
                       associatedRadio="radio_complaint"/>
      <db:deleteButton caption="Delete Complaint"
                       associatedRadio="radio_complaint "/>
      <db:insertButton caption="Store New Complaint"
                       showAlways="false" />
      <db:navNewButton caption="New Complaint"
                       showAlwaysInFooter="false" />
    </td>
  </tr>
  </table>

  <br/>
  <center>
    <db:navFirstButton caption="&lt;&lt; First" />
    <db:navPrevButton caption="&lt; Previous" />
```

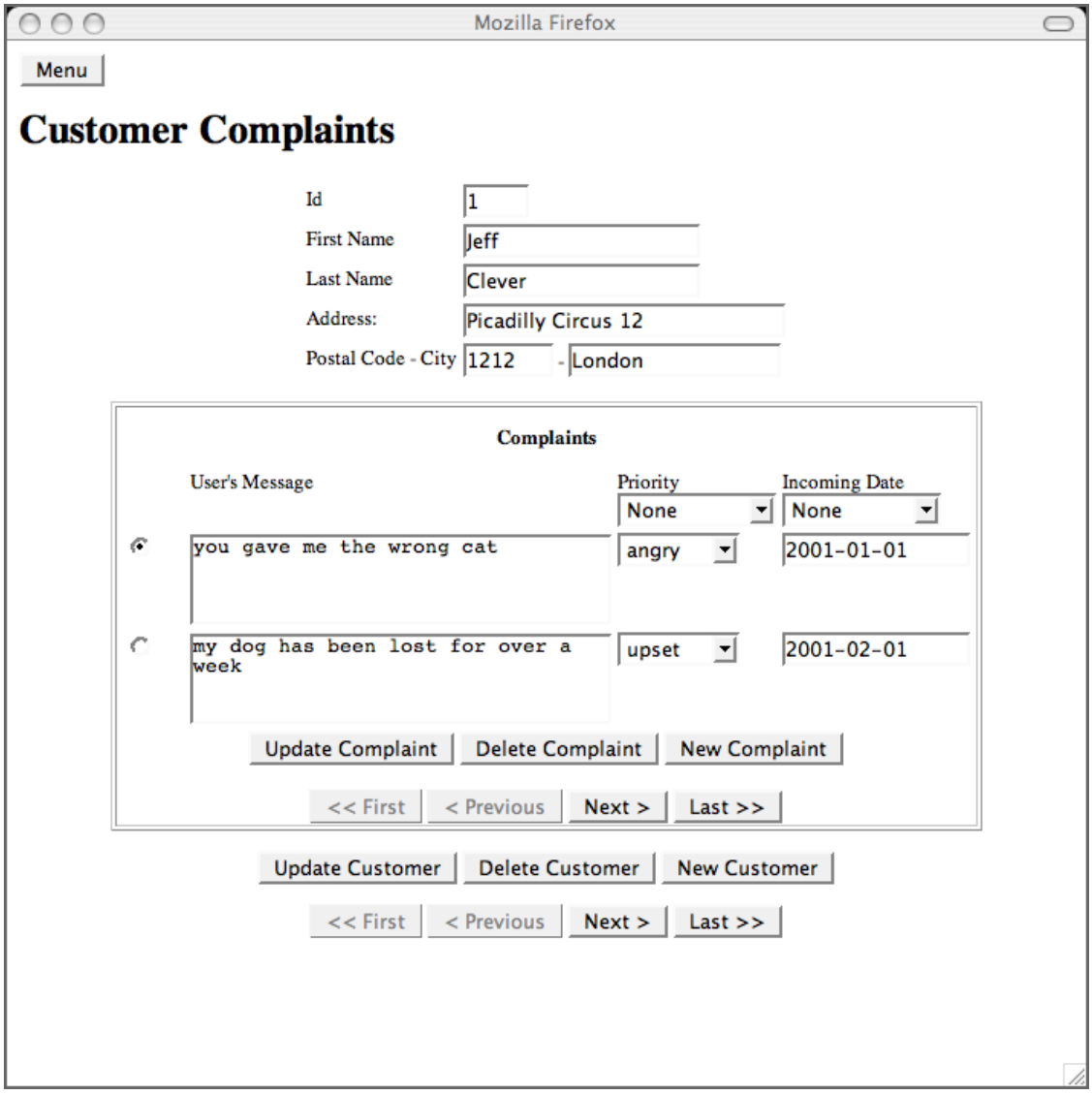
```
            <db:navNextButton caption="Next &gt;" />
            <db:navLastButton caption="Last &gt;&gt;" />
          </center>
        </db:footer>

        </db:dbform>
        <!-------- sub form end -------->

      </td>
    </tr>
    </table>
    <!-- end of table embedding the sub form -->

    <br>
    <center>
      <db:insertButton caption="Store New Customer"  />
      <db:updateButton caption="Update Customer"  />
      <db:deleteButton caption="Delete Customer"  />
      <db:navNewButton caption="New Customer"
                       showAlwaysInFooter="false"/>
    </center>
  </db:body>

  <db:footer>
    <br>
    <center>
      <db:navFirstButton caption="&lt;&lt; First" />
      <db:navPrevButton caption="&lt; Previous" />
      <db:navNextButton caption="Next &gt;"  />
      <db:navLastButton caption="Last &gt;&gt;" />
    </center>
  </db:footer>

  </db:dbform>
</body>
</html>
```

Remarks

In this page, the user can sort the complaints by either priority or by date. The sort control boxes are placed under the column headings using the **`db:sort`** element. To use this feature on a field, the field must be defined as **`isKey`** or **`sortable`** in the `dbforms-config.xml` file.

The `usermessage` field is very long, 255 characters. The **`db:textArea`** element is used to display this field. In our example, up to 96 characters is displayed on three lines.

Figure 7.10. Managing customers and their complaints on a single page (customer_complaints.jsp)



7.4.7. Customer information

This page was created by merging `customer_orders.jsp` and `customer_complaints.jsp` into one file and then removing the update capabilities (radio buttons and update, etc. buttons).

A new technique was used in this form. Only excerpts of the code are shown in the examples below. The complete JSP source code is available in the tutorial application in the `examples/tutorial` directory of the distribution.

Example 7.9. Defining a query with data from more than one table in `dbforms-config.xml`

```
<query name="orders_and_servicenames"
       from="orders,services"
       where="orders.service_id=services.id"
```

```
        orderBy="orderdate">
    <field name="customer_id"  fieldType="int"/>
    <field name="name"         fieldType="char"/>
    <field name="orderdate"    fieldType="char"/>
  </query>
```

Example 7.10. Displaying data from more than one table in a single form (`customers_all.jsp`)

```
<!-------- first sub form begin ------->
<db:dbform tableName="orders_and_servicenames"
           maxRows="2" parentField="id"
           childField="customer_id"
           followUp="/customer_all.jsp">

  <db:header>
    <!-- Show existing orders of services for the customer -->
    <table width="100%">
    <tr>
      <td>Service</td>
      <td>Order Date</td>
    </tr>
  </db:header>

  <db:body allowNew="false">
    <tr>
      <td>
        <db:textField fieldName="name"/>
      </td>
      <td>
        <db:textField fieldName="orderdate" size="14"/>
      </td>
    </tr>
  </db:body>

  <db:footer>
    </table>
```

Remarks

In a previous example, we used the `db:select` element along with a `db:radioButton` element to get the names for the types of services from the Services table and display them in a pulldown list. This displays the name of the current value of the service `id` and allows the user to change the value. But in `customers_all.jsp` we only want to display data rather than provide a capability to update it. We cannot use the `db:select` and `db:radioButton` elements. We need to use a different technique to get and display the names of the types of services.

This technique requires an addition in the `dbforms-config.xml` file. The **`query`** element called **`orders_and_servicename`**. contains an SQL query that joins the Orders and Services tables. We get the data from the Orders table and the service names from the Services table. Then, in the JSP, we use `query` in the same way that we have been using `table`. We reference this query in the **`db:dbform`** element and the field `name` in the **`db:textField`** element.

7.5. Installing the tutorial application

A skeleton of this tutorial is located in the `examples/tutorial` directory of the distribution. The example, however, is not ready to run. This section gives the steps needed to make the tutorial functional. Begin by checking that all of the prerequisites listed in Section 4.1, "Prerequisites" have been accomplished.

7.5.1. Setting up the servlet context

The tutorial needs to be included in the servlet engine's configuration file. For Tomcat, a `Context` element has to be added to the `conf/server.xml` file.

7.5.2. Creating database tables and loading sample data

Next, the tables must be defined in the RDBMS and a user account created. The tutorial has three subdirectories under `examples/tutorial/WEB-INF` called `db_hsql`, `db_pgsql`, and `db_mysql` that provide scripts for the HSQL, PostgreSQL, and MySQL databases. The HSQL database is provided as a shortcut for executing the tutorial. By specifying that database (already specified in the tutorial `dbforms-config.xml` file), the tutorial application can be run without using any other database system.

To use the PostgreSQL or MySQL RDBMSs, create a tutorial database and a tutorial user using database tools. Then, execute the appropriate `tutorial.script` file to define tables and populate some initial data for the application. Changes will probably be required to use another RDBMS because the `orders` and `complaints` tables use generated keys which are non-standard.

Once the database tables are created and populated, the `dbconnection` element needs to be updated to use the selected RDBMS. Using the DevGui program is a good way to configure and debug this portion. Once the database is reachable via DevGui, let DevGui write a configuration file. The database connection element can then be extracted and copied to the tutorial configration file. Otherwise, changes to the `dbforms-config.xml` can be made by hand.

7.5.3. Copying DbForms tag library and dependencies

The final step is to copy the DbForms tag library and dependencies to `WEB-INF` directory. The file `bin/dbforms.tld` must be copied to the `WEB-INF` directory itself. The file `bin/dbforms` X.Y `.jar` (where X.Y is the release number) must be copied to the `WEB-INF/lib` directory. The contents of the `dependend` directory must also be copied to the `WEB-INF/lib` directory.

The application should now be ready to run. As a check, review the typical directory structure in Section 4.3.2, "Typical deployment structure of applications using DbForms" and compare it to the structure of the tutorial as you've modified it.

7.5.4. Final remarks

Errors in the `dbforms-config.xml` file often cause errors in the execution of the DbForms web application.

Whenever a change is made to the configuration file, it is wise to check the servlet engine log files for any errors listed there. In particular, if an error occurs in parsing the configuration file, it is often the case that the database connection is not processed since it occurs near the end of the configuration file. This means that when the application is executed, a database error will be reported (usually mentioning name=null) when the real error occurred earlier in the configuration file.

Once there are no parse errors in the configuration file, try the DbForms application. The most common type of problem at this point is a non-working database connection. The error reported, however, will be different than those described in the previous paragraph.

If the database connection is not working, check the servlet engine log. For Tomcat, this will be the `catalina.out` file. In the case of Tomcat, there will usually be two tracebacks in the log. The first traceback is immediately preceded by another error message that indicates the source of the problem such as an authentication error or a class-not-found error.

When the configuration file is error free and the database connection is working, the application should come together very quickly.

Part IV. Advanced Use and Extensions

Table of Contents

Chapter 8. Configuration Files

8.1. Overview

DbForms application JSPs contain tags for building the application functionality. These tags are defined in the tag library document, not in this section.

The application web deployment descriptor file `web.xml` must contain specific content for DbForms. See Section 4.3, "Building your own DbForms-capable applications" and, in particular, Section 4.3.1, "Edit deployment descriptor `web.xml`" for more information about the requirements for `web.xml`.

The other key application configuration file is `dbforms-config.xml` which is the subject of the remainder of this chapter. Errors in this file often cause errors in the execution of a DbForms web application. Whenever a change is made to this file, it is wise to check the servlet engine log files for any errors listed there.

The `dbforms-config.xml` is read when the servlet container is started or when the web application is reloaded. If there are any errors in this file, the DbForms parser will abort with an exception. The web application will then be left in an indeterminate state. Typically, the servlet engine (such as Tomcat) will continue to service requests for the web application but these will encounter a error such as a Java "No DbConnection object configured with name 'null'" exception.

Setting up the configuration file `dbforms-config.xml` is an important part of creating a DbForms application. This file defines the set of database tables to be used, defines the database connectivity needed to perform operations, and supplies information and parameters for more advanced capabilities such as interceptors and defined queries. The DevGui tool described in Chapter 5, *DevGui* provides an easy way to create an initial version of `dbforms-config.xml`. DevGui uses JDBC to connect with your database and automatically extract table definitions needed for DbForms.

The top level structure of the `dbforms-config.xml` file is shown below. The root element is `<dbforms-config>` and this, of course, occurs only one time. The seven subelements shown inside of the root element must appear in the order shown.

```
<dbforms-config>
  <table></table>
  <query></query>
  <events></events>
  <dbconnection></dbconnection>
  <DOMFactoryClass></DOMFactoryClass>
  <DefaultEscaperClass></DefaultEscaperClass>
  <interceptors></interceptors>
</dbforms-config>
```

The first four subelements, with tags `<table>` through `<dbconnection>`, can appear zero or more times. The remaining subelements, tags `<DOMFactoryClass>` through `<interceptors>`, may occur zero or one time. Since all subelements are optional, an empty root tag is valid although not useful.

8.2. Element `dbforms-config`

The root element surrounds the other elements. Only one `dbforms-config` element can appear in a file. XML namespaces are not used. The root element may be preceded by an XML declaration such as

```
<?xml version="1.0" encoding="UTF-8" ?>
```

or an XML comment.

If you have a source distrtibution of DbForms and wish to validate your `dbforms-config.xml` file, the XML schema is located at `src/org/dbforms/resources/dbforms-config.xsd` relative to the root directory of the source distribution. The JEdit editor (www.jedit.org) with the XML plugin may be used for this purpose. The Apache Xerces project (xml.apache.org/xerces2-j) also provides capabilities to validate an XML document against a schema. Two attributes may be needed in the `dbforms-config` element tag to allow the validation to occur.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="(your path to)/dbforms-config.xsd"
```

8.3. Element `table`

The `table` element is perhaps the most complex element. It contains the majority of the information needed by DbForms to render the DbForms JSP tags. A `table` element is needed for for each table to be used with DbForms.

A starting set of `table` elements as well as a basic `dbconnection` element can be generated using the DevGui tool. This tool, which is described in Chapter 5, *DevGui* , verifies database connectivity and then uses JDBC capabilities to query the metadata for tables in the database. This can be a signficant time-saving step.

The nesting and ordering of subelements within each `table` tag is shown below.

```
<table>
  <field/>
  <calc/>
  <foreign-key/>
  <events>
    <event/>
  </events>
  <interceptor>
    <param/>
  </interceptor>
  <granted-privileges/>
</table>
```

The top level subelements, `field` through `granted-privileges` may each occur zero or more times except `events` which is optional but may appear a maximum of one time.

There is no element content in the any of the elements. All information is supplied in the attributes of these elements and the attributes of their subelements.

The `table` element itself has one required attribute, `name`, giving the name of the table and a number of optional attributes. All of the attributes are listed below.

`name`	The name of the table. Required.
`orderBy`	The name of a column by which the table should be sorted. The order defined in this element is not changeable by the user and appears in the application to be the natural order of rows in the table. If omitted or set to false, the order is determined by the data-

base system. Optional.

`dataAccessClass` — This is the fully-qualified name of a Java class that implements the data access functionality for the table, replacing JDBC. Currently, this is used for XML as discussed in Chapter 25, *Using XML for Data input* but developers may define other data access classes. All data access classses must extend the DbForms class `org.dbforms.event.datalist.dao.DataSource`. Optional.

`escaperClass` — This is the fully-qualified name of a Java class that implements functionality to place escape characters in column data so the HTML display is as desired and to place escape characters in a field value before it is stored in the database. The Java class must implement DbForms interface `org.dforms.util.IEscaper`. Optional.

`alias` — This attribute is used for XML data sources. It provides a URL to access data for an XML "table". The URL includes a query portion (following the '?' in the URL) that is used as an XPath expression to obtain the data. See Chapter 25, *Using XML for Data input* for more details. Optional.

`blobHandling` — If this attribute is set to "`classic`", DbForms will use its original method of handling BLOBs which involved storing BLOB data in individual disk files. When this attribute is omitted or has any other value, the database management system is used to store BLOBs in appropriate table columns. When classic BLOB handling is used, the `directory` and `encoding` attributes of `field` subelements may be used to control where and how BLOB files are handled. Optional.

`defaultVisibleFields-Format` — This attribute sets the default for the `foreign-key` attribute `format` as a convenience when fields from the table are referenced as foreign keys. See Chapter 19, *Foreign Key support within DbForms* for more information. Optional.

`defaultVisibleFields` — This attribute sets the default for the `foreign-key` attribute `visibleFields` as a convenience when fields from the table are referenced as foreign keys. See Chapter 19, *Foreign Key support within DbForms* for more information. Optional.

The `field` element describes a column in the table. There are two required attributes, `name`, giving the name of the column in the table, and `fieldType`, giving the data type of the column. There are also many optional attributes. All of the attributes are listed below.

`name` — This is the name of the field (column) as known in the database. Required.

`fieldType` — This is the type of the field. The list of valid values is shown below. Required. The comparison within DbForms uses the Java method `String.startsWith()` so, for example, Oracle's `varchar2` matches the `varchar` type listed below. Use of an unsupported column type will result in a Java exception while reading the `db-forms-config.xml` file which will generally render the application unusable.

Character Types	DbForms field types `char`, `varchar`, `longchar`, `long varchar`, and `text`.

	Integer Types	DbForms field types `int`, `smallint`, `long`, and `tinyint`.
	Numeric Types	DbForms field types `numeric`, `number`, and `decimal`.
	Date and Time Types	DbForms `date`, `timestamp`, and `time`.
	Floating Point Types	DbForms field types `double`, `float`, and `real`.
	Large Object Types	DbForms `image`, `blob`, and `diskblob`.
	Boolean Types	DbForms field type `bool`.

`size` This attribute gives the size of the field in the database. Optional.

`isKey` This attribute should be set to true if the field is a key field. If omitted, the field is not a key. Optional.

`sortable` This attribute should be set to true if the field will be made available for sorting. If omitted, the field is not sortable unless it is a key field. Optional.

`autoInc` This attribute should be set to true if the field will be populated automatically by the database. Optional.

`directory` This attribute is used with classic BLOB handling where DbForms stores BLOB data in individual files. The value of this attribute indicates the directory where the BLOB file will be stored. This directory must exist and be must be writeable by the servlet engine. Optional.

`expression` This attribute is used for XML data access. The value is taken to be a relative XPath expression that is attached to the URL given by the `alias` attribute in the enclosing `table` element. Optional.

`defaultValue` This value is used to supply a default for the given field when a row is inserted into the database. Optional.

`alias` Not currently used.

`escaperClass` This is the fully-qualified name of a Java class that implements functionality to place escape characters in column data so the HTML display is as desired and to place escape characters in a field value before it is stored in the database. The Java class must implement DbForms interface `org.dforms.util.IEscaper`. Optional.

`encoding` This attribute is used with classic BLOB handling where DbForms stores BLOB data in individual files in the directory given by the `directory` attribute. When `encoding` is set to true, file names for BLOB objects are generated by DbForms so there is no possibility of one BLOB object overwriting another. If `encoding` is false or is omitted, the remote file name of an uploaded BLOB file is used by DbForms. This means that files of the same name will overwrite each other. Optional.

The `calc` element is similar to the `field` element except that the `encoding` attribute is not allowed. A field defined within a `calc` element is not populated by the user. Rather, the value is generated by an interceptor.

Columns in a table that are foreign keys in the SQL sense should be described in using the element

`foreign-key` rather than the `field` element. This allows DbForms to handle the field differently. There are limitations, however, on DbForms foreign key handling. See Chapter 19, *Foreign Key support within DbForms* for more details.

Every table uses, by default, the standard event configuration established in the `events` subelement nested directly within the root `dbforms-config` element. A developer can choose, however, to override the event class associated with a certain type of event, by adding and configuring the `events` element within a `table` element.

The `events` element itself is merely a container for one or more `event` subelements within it. Element `events` has no attributes. Each `event` element may have a number of attributes and may enclose nested `param` subelements. See Chapter 22, *Pluggable events* and especially Section 22.5, "Override the event classes on a table-by-table basis " for more details.

The `interceptor` element is used for declaring interceptor classes locally for a particular table. See the discussion later in this chapter on the `interceptor` element for details of the attributes and the `param` subelement. As mentioned in Chapter 15, *Application hook-ups (Interceptors)* , it is also possible to declare interceptors globally to apply to all tables. Globally declared iterceptors are invoked for all tables but interceptors declared for specific tables are invoked only for that table. If both table-specific and global interceptors are defined, the table-specific iterceptor is invoked first and, if it signals success, the global interceptor is then invoked.

By default, all users of the DbForms application have full access to read and write all of the tables defined in `dbforms-config.xml`.The `granted-privileges` element has four optional attributes that can establish limits for specific user roles. The attributes are `select`, `insert`, `update`, and `delete`. User roles are those defined in the application `web.xml` file. See Chapter 9, *Security* for more details.

8.4. Element `query`

A `query` is a defined SQL SELECT such as a SQL join that can be referenced as if it were a table. A query is defined within a `query` element. The `query` element has the same structure (attributes and subelements) as a `table` element but with some additional subelements and additional attributes on the `query` element itself to define the query by supplying portions of the SQL SELECT statement.

There is an additional repeating subelement `search` which occurs after the other subelements as listed in the preceding section on the `table` element. The `search` subelement allows the query to contain dynamic parameters. The `search` subelement has no element content but does have three attributes. The <`db:search`> tag in a JSP page provides the link between a user-entered value and the query. It also provides the comparison operator used between the column or expression and the user-entered data value. The attributes are listed below.

`name` This is the name of a column in the query. Optional.

`fieldType` This attribute gives the field type. Optional.

`expression` This attribute allows an expression to be used rather than a column name. The expression will be given the name provided in the `name` attribute using the SQL AS operator. Optional.

See Chapter 13, *Query Support* for more information.

8.5. Element `events`

The `events` element contains information about advanced handling for database and page navigation.

It is optional but may occur as many times as desired.

See Chapter 22, *Pluggable events* for a more complete explanation of the use of this element.

The nesting of subelements within the `events` tag is shown below.

```
<events>
  <database-events>
    <database-event/>
  </database-events>
  <navigation-events>
    <navigation-event/>
  </navigation-events>
</events>
```

The `database-events` element must occur at least once but may appear as many times as necessary. The `<database-events>` tag encloses one or more `database-element` elements. There is no element content in the `database-event` element. The element may have the otional attribute `id` and must have the required attributes `type` and `className`.

The `navigation-events` element must occur at least once but may appear as many times as necessary. The `<navigation-events>` tag encloses one or more `navigation-element` elements. There is no element content in the `navigation-event` element. The element may have the otional attribute `id` and must have the required attributes `type` and `className`.

8.6. Element `dbconnection`

This element is used to specify a connection that DbForms will use to access the database. It is possible to define more than one connection. There are attributes for some DbForms tags that allow a specific connection to be specified. If there are multiple connections, one should be designated as the default. The default connection is used unless a specific connection is specified in a DbForms JSP tag.

The nesting of subelements within the `dbconnection` tag is shown below.

```
<dbconnection>
  <property/>
  <pool-property/>
</dbconnection>
```

The `property` element may occur zero or more times. Element `pool-property` may also occur zero or times. Neither of these elements can contain element content. Instead, all information is supplied in required attributes. These attributes are `name` and `value`. The attribute values are simple strings.

The `dbconnection` element itself has a number of attributes. All of the attributes except `name` are optional.

See Chapter 2, *DbForms Concepts*, Chapter 20, *Connection Support*, and Chapter 21, *Multiple Database Connections* for more detail about this element and for examples of its use.

8.7. Element `DOMFactoryClass`

This element contains the fully-qualified class name to be used as a DOM factory object. The element content between the start and end tags gives the class name. There are no attributes for the element.

8.8. Element `DefaultEscaperClass`

This element contains the fully-qualified class name to be used to escape database and other HTML content. Escaping is necessary to make sure that characters such as '<' and '>' in displayed data do not compromise the HTML structure of the DbForms page. A replacement escaper class might be needed if the DbForms default escaper class is not sufficient.

The element content between the start and end tags gives the class name. There are no attributes for the element.

8.9. Element `interceptors`

See Chapter 15, *Application hook-ups (Interceptors)* for a more complete explanation of the use of this element. This element is used for declaring interceptor classes globally. It is also possible to declare interceptors within each table. See the `table` element earlier in this chapter. Globally declared interceptors are invoked for all tables but interceptors declared for specific tables are invoked only for that table. If both table-specific and global interceptors are defined, the table-specific iterceptor is invoked first and, if it signals success, the global interceptor is then invoked.

The nesting of subelements within the `interceptors` tag is shown below.

```
<interceptors>
  <interceptor>
    <param/>
  </interceptor>
</interceptors>
```

The `interceptors` subelement is optional but, if present, can only occur once. The `<interceptors>` tag encloses one or more `interceptor` elements. Within an `interceptor` element there may be zero or more `param` elements that provide fixed parameters to the instances of the interceptor class. None of these elements contain element content. Instead, all information is supplied in attributes.

The `interceptors` element has no attributes.

The `interceptor` element has a single required attribute `className` that gives the fully-qualified Java class name of the interceptor class. This class must implement the `org.dbforms.config.DbEventInterceptor` interface.

The `param` element has two required attributes, `name` and `value`. Attribute `name` gives the name of the parameter and the `value` attribute gives a string value for the parameter. These parameters are accessible via the servlet context which is made available to the interceptor class via the `config` parameter of type `DbFormsConfig`.

Chapter 9. Security

9.1. Introduction

DbForm's security features are fully compliant with SUN's Java Servlet API Specifications, version 2.2. Chapter 11 of the Java Servlet Specification 2.2 describes the mechanisms containers must provide in order to enable developers and deployers to ensure the following characteristics of security:

- Authentication
- Access control for resources
- Data Integrity
- Confidentiality of Data Privacy

DbForms, built on top of JSP, naturally builds on top of JSPs security concepts. Building a proprietary security system for DbForms (from scratch) would not make much sense. Building on top of implies that DbForms indeed provides an **additional** level of security handling.

In short, DbForms takes away much work and worries from the *developer*, but not from the *deployer*. The deployer is still concerned with defining user-groups, choosing UserManager-factory-classes, eventually setting up database tables for user name/password, etc. Many of these issues (i.e. choosing UserManager-factories) are dependent of Application Servers (which is another reason why DbForms does not address these issues).# For more information about the role of the deployer, refer to paragraph 2.2.2 of the Java Servlet Specification (Roles/Deployer)

As we have elaborated, the deployer establishes the *basic infrastructure* for application security. Would that be sufficient for fulfilling an application security needs? Well, the deployer could indeed provide a certain level of security, but building an application using declarative security only, will not work. Why?

Consider the following example:

We have a view called customer.jsp, accessing a table customer, and we have three users-groups A, B and C. (The terms group and role are used synonymously in this document.)

> Members of A may read, insert, update and delete customers, members of B may read and update customers and members of C have no access at all.

Using declarative security a deployer could define security constraints in web.xml that allow members of A and B to access customer.jsp and deny access for C. But how should the distinction be made between A and B? One way would be to split customer.jsp into 2 files -one with a delete button and one without. But how do we prevent malicious users from simulating a delete button using telnet?

This is where DbForms security concept steps in:

9.2. Security concept of DbForms

DbForms provides fine-grained definition of rights for data-access and manipulation. DbForms security works directly on *tables*, not on JSPs. All database operations done by DbForms run through a security check before they get executed.

DbForms allows attaching security constraints to each table defined in the xml-configuration. Listing 5 demonstrates how the problem described above could be solved using DbFormss built-in-security concept:

```
<dbforms-config>
  <table name="customer" >
    <field name="id" fieldType="int" isKey="true" />
    <field name="firstname" fieldType="char" />
    <field name="lastname" fieldType="char" />
    <field name="address" fieldType="char" />

    <granted-privileges
      select = "A,B"
      insert = "A"
      update = "A,B"
      delete = "A"
    />
  </table>
</dbforms-config>
```

The attributes of the <granted-privileges> element tell DbForms:

- Members of A may select, insert, update and delete customers
- Members of B may only read and update customers
- All other groups (which include C) may not access this table at all.

This is exactly what we needed to solve the problem!

Nota bene: not all attributes must be specified explicitly. If, for instance, all users may have *read* access to a table, the *select* attribute needs not to be declared. A missing attribute does not generate an error but implicates access for all groups (which may be useful if there are dozen's of roles)

If a `<table>`-element does not contain a `<granted-privileges>`-element, then all users have full read write - access to this table.

If a user tries to execute an operation beyond his/her access rights, an SQL-Error will be triggered and (if a `<db:errors/>` tag is placed on the view) displayed to the user.

Note: current action buttons are not aware of security (i.e. a delete button will get rendered independently of the users right to delete data from the table)

Chapter 10. File uploads

10.1. Introduction

In the past, most database applications dealt with data rows consisting of simple alphanumerical fields. But today, there is an increasing demand to be able to store and manipulate *complex* data. Complex data is imminent not only in multimedia applications for the Web, but also in specialized application domains such as document- and workflow management.

Most relational DBMS vendors have added the capabilities to manipulate binary large objects (BLOBs) in the database. However, in many cases BLOBs are still not a viable option:

- If the application uses a RDBMS or JDBC-driver without BLOB-support,
- If BLOB-support is too slow or even buggy, or
- If the files should be accessible without using a database layer. For instance, imagine a web content management system placing files (images, pdf's, applets, etc) directly into the Http-Root of the server.

DbForms addresses *all of these situations:* On the one hand, it supports the upload (and download) of files into classical BLOB fields, and on the other, it supports a file-system approach, allowing database-driven management of files outside of the database. (Depending on your RDBMS, a binary large object is called a BLOB, IMAGE, etc.)

10.2. BLOB-fields

10.2.1. Defining BLOBs in the model definition

Managing BLOB Fields using DbForms is a trivial task. First, you have to tell DbForms about BLOB-Fields in the xml-configuration file:

NOTE: after version 2.2, interceptor based BLOB handling was introduced. The interceptor `org.dbforms.event.BlobInterceptor` is used to store the filename of an uploaded file in a specified column. In previous versions, the BLOB was stored as a serialized Java object in the database along with the file meta data. However, the disadvantage of this solution was:

- the lack of long term storage (i.e. a change in the wrapper class would lead to ClassCastExceptions)
- the BLOB is not readable by other (non-DbForms) applications because those apps can't de-serialize the java object

To switch back to the old way of handling BLOBs, you can set the optional attribute `"blobHandling"` to `"classic"` in the table definition in your `dbforms-config.xml` file.

More documentation on this issue will follow; until then this section is a bit outdated. meanwhile, please refer to the API doc for class `org.dbforms.event.BlobInterceptor` to see how to configure the BLOB interceptor the new way. The following example is for the "classic" way.

```
<dbforms-config>
```

```
            <table name="pets" blobHandling="classic">
                <field name="pet_id" fieldType="int"
                       isKey="true" autoInc="true" />
                <field name="name" fieldType ="char" />
                <field name="portrait_pic"
    fieldType ="blob" />
                <field name="story"
    fieldType ="blob" />

                              </table>
        </dbforms-config>
```

The configuration-code-snippet shown above tells DbForms that the fields portrait_pic and story are of type BLOB. As you can see, DbForms allows more than one field defined in a table to be of type BLOB.

10.2.2. Uploading files into BLOBs

After defining our BLOB-powered table, we want to manage the BLOB fields. Of course data-input elements like textField or select cannot be applied to BLOBs. For this purpose a new Data-Element is introduced:

```
<db:file
fieldName=portrait_pic>
```

The attribute `fieldName` refers to the name of the field, the file needs to be uploaded into. There are additional attributes available for this element (accept, maxLength, etc.), please see chapter DbForms Custom Tag Library for additional information)

This Custom Tag gets rendered as an HTML <file> - tag, as shown next:

This HTML-element enables multipart-enabled browsers to submit files to the server. Be aware that not all browsers support this functionality.

Important: the multipart-attribute of the <dbforms> element must be set to `true` to support multipart requests!

It is possible to save the MIME type and file size of a BLOB. This is done by providing column names for the `BlobInterceptor` to use. The example shown below shows how this is done.

```
<interceptor className="org.dbforms.event.BlobInterceptor">
   <param name="blob-column" value="BODY"/>
   <param name="name-column" value="FILE_NAME"/>
   <param name="mime-column" value="MIME_TYPE"/>
   <param name="size-column" value="BODY_SIZE"/>
</interceptor>
```

The above code would be added to a `table` element in `dbforms-config.xml`.

10.2.3. Updating and deleting BLOBs

Regarding update and delete operations, BLOB fields may be treated similar to common (alphanumerical) fields:

if a user hits the DELETE-button, the BLOB fields of the selected row will be deleted. (this is done automatically by the RDBMS)

If a user hits the UPDATE and no new file is uploaded, the existing BLOB will NOT be modified. If a user has chosen a new file, (using the Browse- Button or by entering a valid file-path into the text-field left to the button) any existing BLOB at the selected row will be overwritten with the new value.

10.2.4. Retrieving BLOBs

Managing BLOBs as described above may be sufficient in most cases. But sometimes, we may want to reverse the process: we may want to *retrieve* BLOBs. (for instance, to provide users the ability of visualizing uploaded images, etc.)

DbForms provides a simple way of retrieving BLOBs from the database. You are, however not required to use it. Again, using dbForms is meant to help you increase your productivity, if another tool or techniques is better for a given task include it!

For retrieving a BLOB using DbForms BLOB-retrieving facility, you need to use the following tag:

```
<db:blobURL fieldName="portrait_pic"/>
```

This tag renders an URL pointing to a servlet built into DbForms. This servlet retrieves the specified field from the database, tries to figure out the MIME type of the file, assigns it to the response and finally returns a stream of bytes containing the data of the BLOB.

To retrieve and render an image, the following JSP-code would need to be specified:

```
<img src=<db:blobURL fieldName="portrait_pic"/>
     width=100 height=80 border=0>
```

To establish a link to a BLOB (for example to a word or pdf document) you could write:

```
<a href=<db:blobURL fieldName="portrait_pic"/>>
  Click to download</a>
```

Dependent of the installed plug-ins and other browser-and system properties, either the MIME-matching plug-in (or helper application) will be opened, a save-as dialog will pop up or a download manager like *Gozilla!* will be started.

It is important to mention that the <blobURL> element is valid only if embedded in a <body> - element. The <file> tag works in <header>, <body> and <footer> elements. Within the <body> element, both - inserts and updates - are possible, within the other tags, only inserts are possible.

10.3. File-system approach

DbForms also supports the storage of files on the file-system. This is completely **transparent** to the JSP (view) developer! For uploading and retrieving file-system-stored objects, **the same tags and attributes** are used as for uploading and retrieving BLOBs.

The difference lies only in the definition of the model; instead of fieldType blob, we define a type called diskblob. I invented this term because I think it reflects what it represents a binary large object stored to disk (instead of to the database)

10.3.1. Defining DISKBLOBs in the model definition

As the following example and table show/explain, there are additional attributes available for defining a diskblob-field:

```
<dbforms-config>
        <table name="pets">
            <field name="pet_id" fieldType="int"
                  isKey="true" autoInc="true" />
            <field name="name" fieldType ="char" />
            <field name="portrait_pic"
                   fieldType ="diskblob"
                   directory="c:\uploads\pets"
                   encoding="true" />
            <field name="story"
                   fieldType ="diskblob"
                   directory="x:\stories" />
        </table>
</dbforms-config>
```

Table 10.1. define a table capable of Blobs (depends on SQL dialect implemented by RDBM vendor)

Directory [this attribute is required]	Holds the directory uploaded files should be stored to. This directory must exist and b) must be write-able by the web-server
Encoding [this attribute is optional]	Holds either #true# or #false#. If "true", then files will be renamed to ensure that every file is unique and no file overwrites another. If this attribute is not specified, the default "false" is applied (keeping original names and overwriting equal-named files)

With the definition shown in the example preceeding the table, we can expect the following behavior:

If a portrait_pic (lets say dog.jpg) is uploaded, it will be stored in directory c:\upload\pets under a unique name like 34834893483_748734.jpg)

If a story (lets say dog.pdf) is uploaded, it will be stored in directory x:\stories (which may be a virtual directory pointing to another computer-box) under its original name.

10.3.2. What goes on behind the scenes

You might ask: How does DbForms remember where it stores the files? The answer is simple: the refer-

ences to the actual physical files are stored in the specified DISKBLOB field, which turns out to be an ordinary alphanumerical (CHAR, VARCHAR, or similar) field.

Following our example, the pets table would now contain a row such as the one displayed below:

Figure 10.1. Physical representation of our example

pet_id	name	portrait_pic	Story
...			
101	Snoopy	34834893483_748734.jpg	dog.pdf
...			

Both files are stored in their respective directories (c:\upload\pets x:\stories). The directory names are not included in the references for two reasons (please email me any arguments to the contrary):

It is not necessary because this information is managed by the XML-configuration

The value may be retrieved using a simply db:label or db:textField tag. It is not a good idea to expose server internals (directory structure) to the clients.

10.4. A working demo

The following demo (and others) can be viewed *online here* [http://www.dbforms.org/demo.html]

Let's imagine we want a little application for managing information about pets: their name, a portrait-pic and a funny story about them. We will use our example-table defined earlier in this chapter.

Using mySQL, this table could have been created as follows: (check your RDBMS-SQL-manual if you are not sure how your database handles auto incremental fields or BLOBs)

```
CREATE TABLE pets (
   pet_id int AUTO_INCREMENT NOT NULL PRIMARY KEY,
   name char (30) NULL,
   portrait_pic BLOB NULL,
   story BLOB NULL,
 );
```

Here is the source code we need to create the application:

```
<%@ taglib uri="/WEB-INF/dbForms.tld" prefix="db" %>
 <html>
  <head>

      <db:base/>
    <link rel="stylesheet" href="dbforms.css">

    <SCRIPT>
        <!-- function openWin(s) {
        window.open(s,"littlewin","width=450,height=350,resizable=yes");
        }-->
```

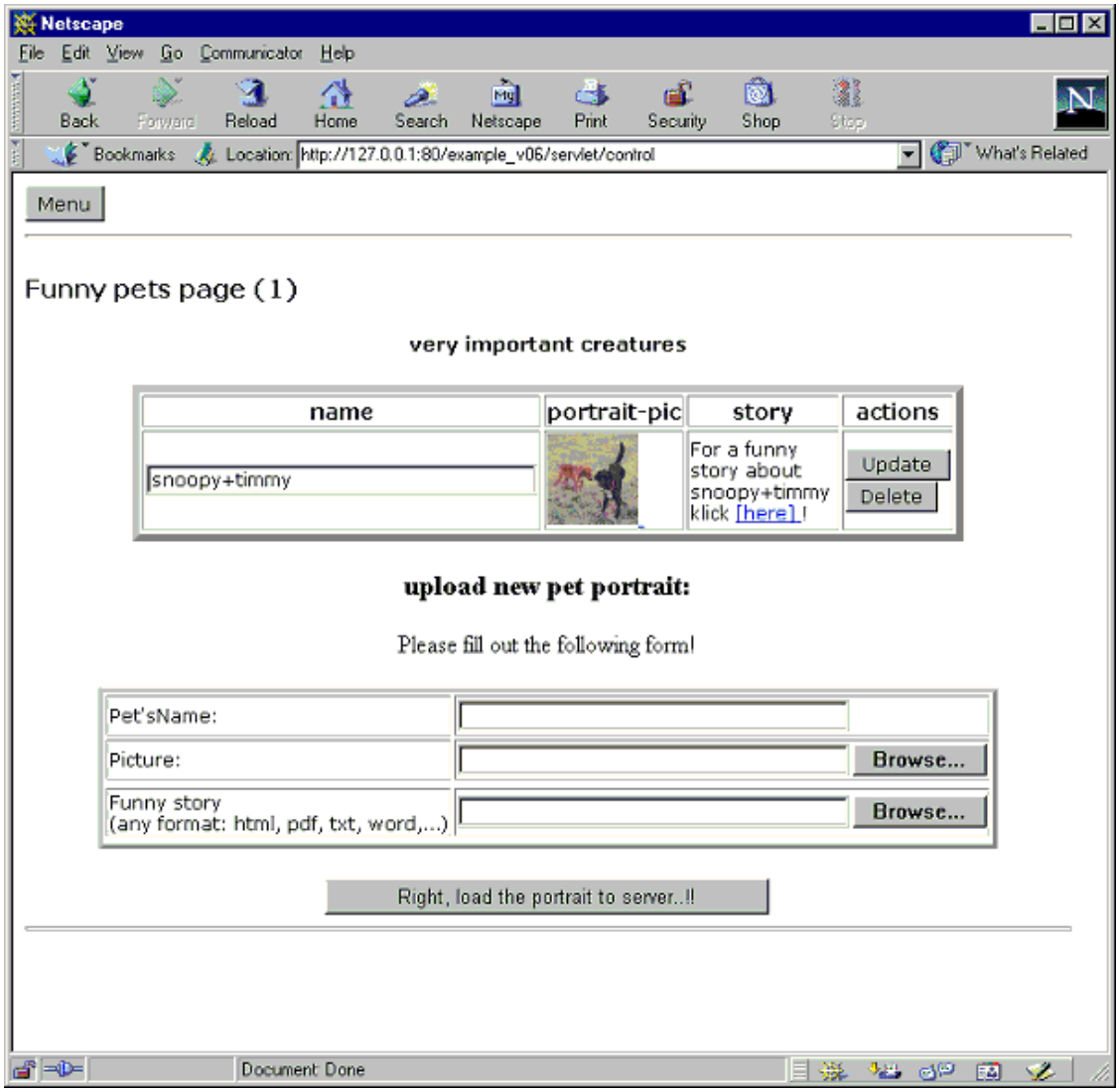
```
  </SCRIPT>
</head>

<body>

    <db:errors/>
    <db:dbform tableName="pets" maxRows="*"
               followUp="/petportraits.jsp" multipart=true>
      <db:header>

    <db:gotoButton caption="Menu" destination="/menu.jsp" />

          <hr>
          <h1>Funny pets page (1)</h1>
          <center><h3>very important creatures</h3></center>

          <table border="5" width="60%" align="CENTER">
              <tr>
                  <th>name</th>
                  <th>portrait-pic</th>
                  <th>story</th>
                  <th>actions</th>
              </tr>

    </db:header>
    <db:body allowNew="false">
              <tr>
                  <td>
    <db:textField fieldName="name" size="20" maxlength="30"/>
                  </td>
                  <td>
                      <a href="javascript:openWin('/example_v06/
    <db:blobURL
                           fieldName="portrait_pic"/>')">
                           <img src="
    <db:blobURL fieldName="portrait_pic"/>" border="0"
               width="60" height="60" alt="pet portrait (click to see full si
                      </a>
                      <br>
                  </td>
                  <td>
                      For a funny story about
    <db:label fieldName="name"/> click
                      <a href="javascript:openWin('/example_v06/
    <db:blobURL fieldName="story"/>')">
                      [here]
                  </td>
                  <td>

    <db:updateButton caption="Update"/>

    <db:deleteButton caption="Delete"/>
                  </td>
              </tr>

    </db:body>
    <db:footer>
          </table>

          <center><h3>upload new pet portrait:</h3></center>
          <p>
          <center>Please fill out the following form!</center>
          </p>

           <table align="center" border="3">
```

```
                <tr>
                    <td>Pet's Name:</td>
                    <td>

        <db:textField size="20" maxlength="30" fieldName="name"/>
                    </td>
                </tr>
                <tr>
                    <td>Picture:</td>
                    <td>

        <db:file fieldName="portrait_pic" accept="image/*" />
                    </td>
                <tr>
                <tr>
                    <td>
                        Funny story<br>(any format: html, pdf, txt, word,...)
                    </td>
                    <td>

        <db:file fieldName="story" />
                    </td>
                <tr>
            </table>

            <br>
            <center>

        <db:insertButton caption="Right, load the portrait to server..!!"/>
            </center>

            <hr>

        </db:footer>
        </db:dbform>

    </body>
    </html>
```

You have now seen `<blobURL>` and `<file>` in action! As you can see, it is pretty easy to handle files with DbForms. Most of the code is about laying out the html-view and playing around with JavaScripts to create pop-up windows for showing the pet-stories and portraits in an attractive way.

The final result of our efforts looks like this:

Figure 10.2. our little BLOB-App in action


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://127.0.0.1:80/example_v06/servlet/control
What's Related
Menu
Funny pets page (1)
very important creatures
name
portrait-pic
story
actions
snoopy+timmy
For a funny story about snoopy+timmy klick [here] !
Update
Delete
upload new pet portrait:
Please fill out the following form!
Pet'sName:
Picture:
Browse...
Funny story
(any format: html, pdf, txt, word,...)
Browse...
Right, load the portrait to server..!!
Document Done

Chapter 11. Styling

11.1. First level: basic HTML and JSP - tags

As you may have noticed, the sample applications used lots of table/tr/td/br/hr tags as well as cascade stylesheets to build an average interface.

We could use other HTML/JSP code to build a better interface: For instance, we could use much more JavaScript or DHTML to create eye catching effects, or we could even include headers and footers using JSP:include elements, etc.

DbForms does not restrict you in any way.

11.1.1. Cascading Style Sheet Usage

A lot of attributes like color, background color, fonts etc. can now be easily changed at a central place by using the stylesheet **dbforms.css** to make generated JSPs to your liking in terms of style. The stylesheet dbforms.css can be found inside the **misc** directory in cvs and within the directory **dist** inside a DbForms distribution..

If you don't like the appearance of generated pages, it is now easy to change it.

11.2. Second level: <dbForms> - tags

Most visible tags of the dbForms Custom Tag Library provide attributes for styling, just as their HTML-pendants do (please refer the chapter DbForms Custom-Tag-Library for more information about the attributes for embellishing tags).

A special issue of styling is the **action-buttons**. These buttons provide additional functionality for a better styling, as the following examples demonstrate:

Example 11.1. Instantiating a standard button


```
<db:gotoButton caption="Edit bears" destination="/bears.jsp"
/>
```

Example 11.2. Instantiating an image button

```
<db:gotoButton destination="/bears.jsp"

flavor="image" src="bear.gif" alt="click to edit bears"
/>
```

Example 11.3. Instantiating a HTML4 button


```
    <db:gotoButton destination="/bears.jsp"
 flavor="modern" style="width:220"
>
        <b>Bears!</b>

        <p>

        <img src="bear.gif" border="0">
</p>
<b>click if you want to view them</b><br>
(but be careful and don't feed...)

    </db:gotoButton>
```

Nota bene: you should be aware that the image and modern flavored buttons are built on HTML standards not supported by all browsers. For example the modern button shown in Listing 15 wont work on Netscape 4.

11.3. Third Level: DbForms - templates

Please note: this topic is currently under construction. It is possible that the styling approach described

in this chapter will be replaced by some other approach in subsequent versions. (Further comments at the end of the chapter)

11.3.1. Introduction

DbForms comes with a simple yet powerful method of template-processing. This mechanism was designed to ease and speed up development. In summary, it is based on the following requirements:

- It should tidy up the HTML/JSP code. In other words, it should act as a kind of macro-engine replacing complex code by simple statements. This makes the HTML/JSP-code easier to read, understand and maintain.
- It should be easy to use. Application-designers using predefined templates should not need to do any complicated coding. They should just specify which template to use and eventually specify attributes for customizing the templates this leads to the next point:
- Templates should be dynamic, not static. Templates should be able to retrieve and process attributes provided by the application -designer. Furthermore, templates should be JSP pages themselves, to provide maximum flexibility.
- New templates should be easy to create. Transforming a good-looking HTML/JSP code into a valid and working template for use by DbForms-applications should be a matter of a few minutes, not hours! This should ensure growing repositories of powerful DbForms-templates which are available to other DbForms-developers (whether inside a company or even better as a contribution to all developers)

11.3.2. How to use DbForms templates

Including existing templates into the JSP view is according to the goals described above an easy task. There is only one custom tag, `db:style tag`, that needs to be facilitated for this purpose:

```
<db:style
template="templateName"

paramList="param1='value1',param2='value2',. . .,paramN='valueN'"

part= "begin"  /
>
```

This tag may or may not include a body. Its attributes are:

- **template** [required]: this refers to the name of the template to be used
- **paramList**: a parameter-list used to specify rendering properties for the template.
- **part**: the part of the template (i.e. begin, end)

11.3.3. Example of *applying* templates

Imagine a simple login box. The code for such a form could resemble the following:

```
    <html>
        <body>

<db:style template="center" part="begin" />


<db:style template="sourceforge" paramList="width='300'
            bg='steel3.jpg'" part="begin"/>

                <form method="POST">

                    <table align="center">
<tr>
                            <td>
                                <b>Username: </b>
                                <br>
                                <input name="j_username" type="text">
                                <br>
                            </td>
</tr>
                        <tr>
                            <td>
                                <b>Password: </b>
                                <br>
                                <input name="j_password" type="password">
                                <br>
                            </td>
                        </tr>
                        <tr>
                            <td align="center">
                                <br><input type="submit" value="Log in!">
                            </td>
                        </tr>
                    </table>

                </form>


<db:style part="end"/>


<db:style part="end"/>

        </body>

    </html>
```

Remarks

Note the bold printed <style> tags. The use of these tags make up the difference between the screenshot shown in Figure 17 and the other one in Figure 18

As you may notice, one template (sourceforge) is nested into the other (center).

Another thing you may notice is that the sourceforge-template (which I have taken from the popular open-source development site sourceforge.net) defines a parameter-list while the other template does *not*.The parameters declared in the parameter list are written in JavaScript-style using *single quotes*, but the quotes may be left out (they are tolerated to make parameter lists easier to read) The important thing however, is to separate each param-value-pair by a *comma*.

Figure 11.1. The login box without styling templates

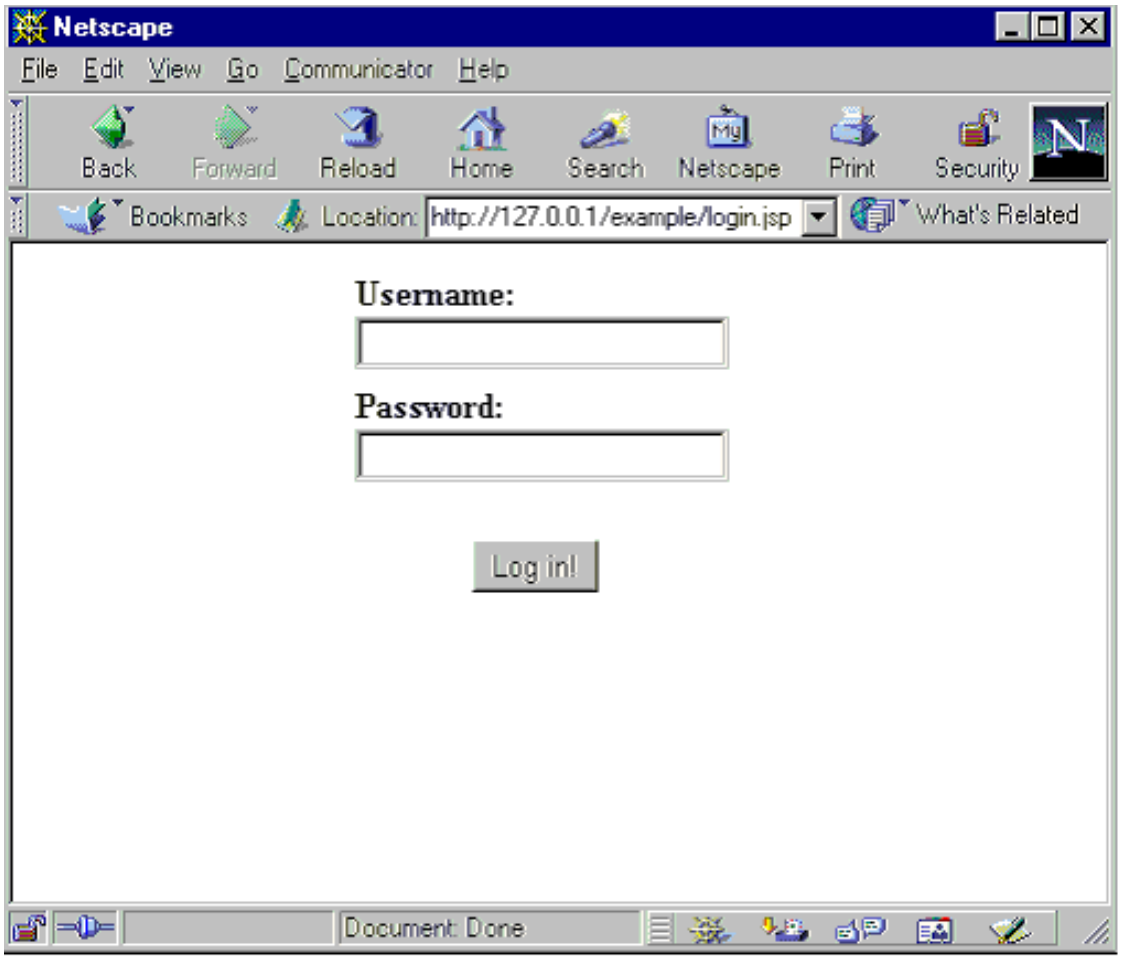
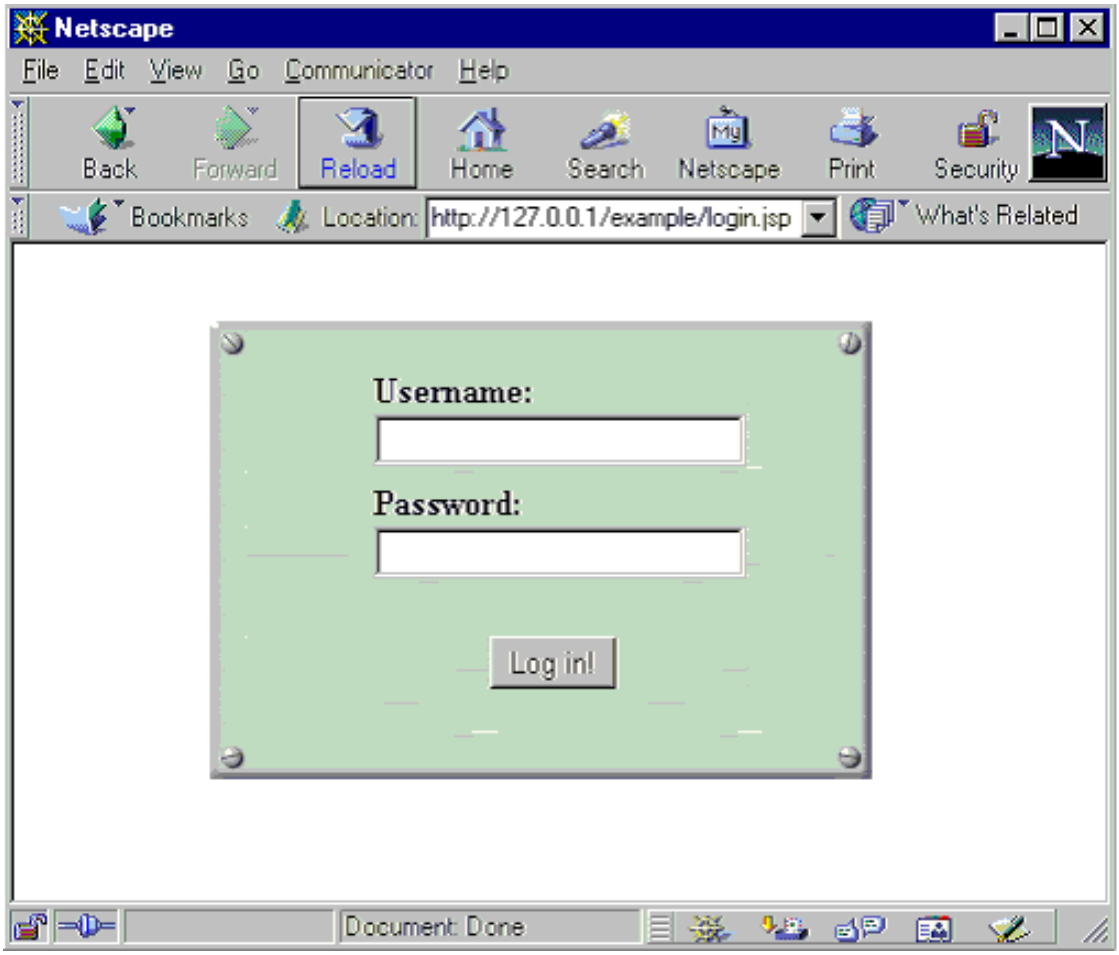


Figure 11.2. The same login box using 2 styling templates



11.3.4. Example of *creating* templates

11.3.4.1. Template base directory

There exists (optionally) one template-directory for each web-application. This directory should be dedicated exclusively to templates. The location of this directory is specified in the web application deployment descriptor (web.xml) using the context-parameter templateBase (see following example)

```
<context-param>

      <param-name>
templateBase</param-name>

      <param-value>mytemplates</param-value>

  </context-param>
```

If there is no such entry in web.xml, then the default directory `templates` will be used.

The directory entry is relative to the root of the web-application (absolute directories are not allowed)

This base-directory contains all templates accessible by the application. Each template has its own sub-directory. In the example above, the subdirectories sourceforge and center would exist.

11.3.4.2. The structure of templates

A template usually consists of two files: One file to be included/rendered when the start of the style-tag is evaluated, and another file to be included/rendered when the end of the style-tag is evaluated. The starting file has to have the suffix _begin.jsp, the ending file has to have the suffix _end. The names of both files start with the name of the template (and the directory they are contained in)

In the example above, the following structure exists:

```
            ourWebApp/templates/
            ourWebApp/templates/sourceforge/
ourWebApp/templates/sourceforge/sourceforge_begin.jsp
ourWebApp/templates/sourceforge/sourceforge_end.jsp
ourWebApp/templates/sourceforge/images/
ourWebApp/templates/sourceforge/images/steel.jpg
ourWebApp/templates/sourceforge/images/nail.jpg
ourWebApp/templates/center/
ourWebApp/templates/center/center_begin.jsp
ourWebApp/templates/center/center_end.jsp
```

As you may have noticed in Listing 18, there can also exist other resource types (image-/jsp-/html-files) in addition to the mandatory `*_begin.jsp` and `*end.jsp` files. These files may be statically or dynamically included by the templates-stubs.

11.3.4.3. Coding templates

11.3.4.3.1. Static templates:

center_begin.jsp

```
<table width="100%" height="100%">
    <tr>
        <td>
        </td>
    </tr>
    <tr>
        <td>
<center>
```

center_begin.jsp

```
                </center>
            </td>
        </tr>
        <tr>
            <td>
            </td>
  </tr>
    </table>
```

As most readers will agree, there is not much to tell about the code above. In this example, a table gets rendered which centers all elements it embeds (for instance the login-form).

11.3.4.3.2. Dynamic templates:

a few snippets from sourceforge_begin.jsp

```
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="db" %>[...]

    <table cellpadding="0" cellspacing="0" border="0"
    width
            ="<db:templateParam name="width" defaultValue="99%" />">
     [...]

        <td background
            ="<db:templateBasedir/>images/tbar1.png" width="1%" height="17">
         [...]


            <db:templateParam name="bg" defaultValue="steel.jpg" dir="images" />"
             [...]
```

This example shows the usage of two custom tags supporting templates.

- <templateParam> parses a specified parameter from the paramsList-attribute of the calling JSP.
- <templateBasedir> supplies the base directory to use. This is important to make sure all resources (ie: images) are found.

More information about these tags can be found in the description of the custom tag library and in on-line examples.

11.4. Why is the style-taglib still under construction?

Because it has one serious limitation: <style> tags may not be nested inside any other custom tag. Many servers, including Suns reference implementation Apache Tomcat 3.2, are firing an exception like illegal flushing within custom tag. This stems from the way response- and writer-objects are handled in the JSP

framework.

Other servers like Orion server tolerate db:style tags within other tags, but of course DbForms is obliged to the JSP specification and its reference implementation.

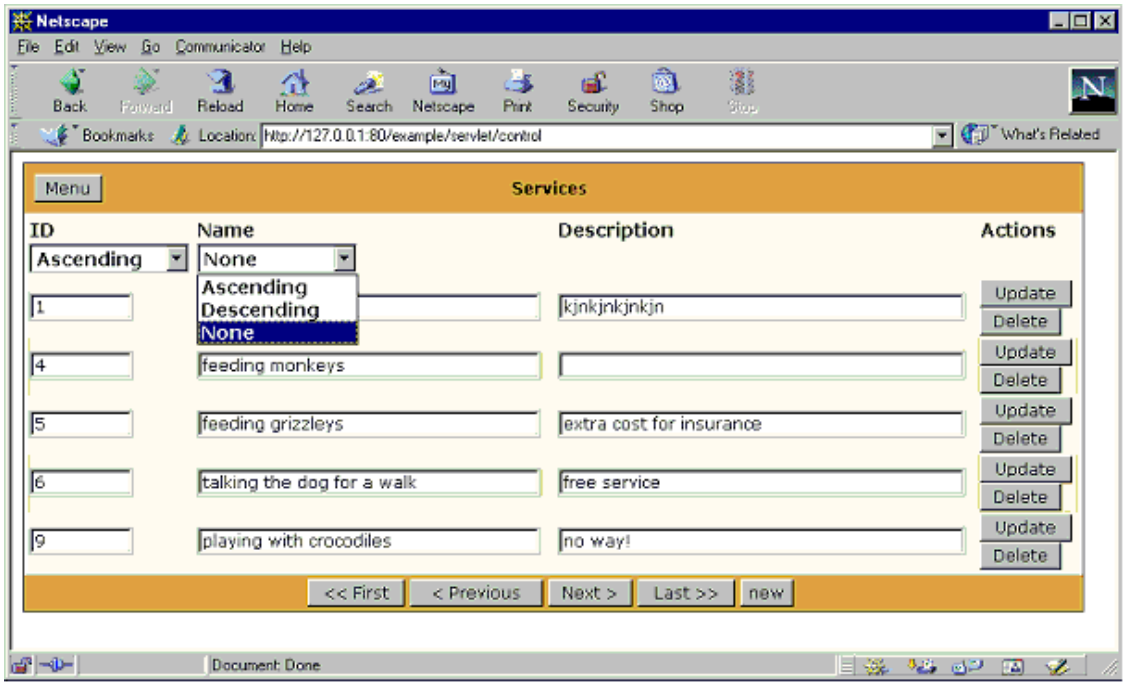
This is the reason why I am currently researching how to bypass the problem. Any hint from the reader would be highly appreciated!

Chapter 12. Sorting, Filtering and Searching.

12.1. Sorting

Sorting is an important, even essential, issue for many database applications. For example, look at the application shown in Figure 19 - it is an enhanced version of the service-JSP taken from the sample application (demonstrated in an earlier chapter)

Figure 12.1. DbForms built in sorting mechanism in action



The user is able to define the sorting of the table by choosing a value Ascending, Descending or None from the sorting-select-boxes below ID or below NAME

If the user wants, for instance, the table to be ordered by NAME instead of ID then he/she has to select ASCENDING or DESCENDING from the NAME drop-down select box.

12.1.1. How to use DbForms sorting facility

The custom tag enabling this functionality is very easy to use:

Example 12.1. Instantiating a sort - tag

```
<db:sort fieldName="id" />
```

Basically, this is all you need to write! This functionality even works within nested forms. But there is one important configuration issue you should be aware of: **all non-key-fields** you want to be sortable should be declared sortable in the XML-config file:

Example 12.2. Preparing tables for sorting: define sortable fields

```
<table name="service">
      <field name="id" fieldType="int" isKey="true"/>

<field name="name" fieldType="char"
sortable="true"/>
<field name="description" fieldType="char" />
  </table>
```

The default sorting behavior is false. You should use **sortable=true** sparingly, especially fields with big field size should not be declared as sortable if it's not really necessary. The reason for this is the amount of data to be transferred for enabling sorting. The more sortable fields, the more data traffic!

Please note: sorting is one of those issues which are still under development. The sorting-boxes described in this tag are already very powerful, but they are still only a beginning. A limitation of the current implementation is the melting of view and controller for the search widgets. Future versions of DbForms may be expected to provide more on sorting.

12.1.2. Alternative method to order query results

An alternative to sorting is to use a `orderBy`attribute in the dbform tag of your jsp page.

```
<db:dbform table="TABLE1" followUp="this.jsp" orderBy="id"/>
```

Similar to sorting, if you use a non-primary key with orderBy, you MUST specify `sortable="true"` within your dbforms-config.xml file.

If you use orderBy with a non primary key and you don't specify that in the config file, then navNext/navPrev buttons will function like navLast/navFirst buttons and you will skip over results.

See the *TagLib* [../../taglib/DbFormsTags_Frame.html]for more information.

12.2. Filter Tag

There now exist three main ways to filter information

i. Filter tag (from 1.1.4pr1)

The filter tag can be thought of as the next generation filtering method and can effectively replace the following `filter` and `whereClause` attributes, as well as providing an improved alternative to variable searchFields.

It has these advantages:

- Allows for more complex sql conditions than the `filter` attribute allows
- Navigation works with both `classic` and `dataList` navigation sytems (navigation with a `whereClause` only works with `dataList`).
- Simplifies searching.

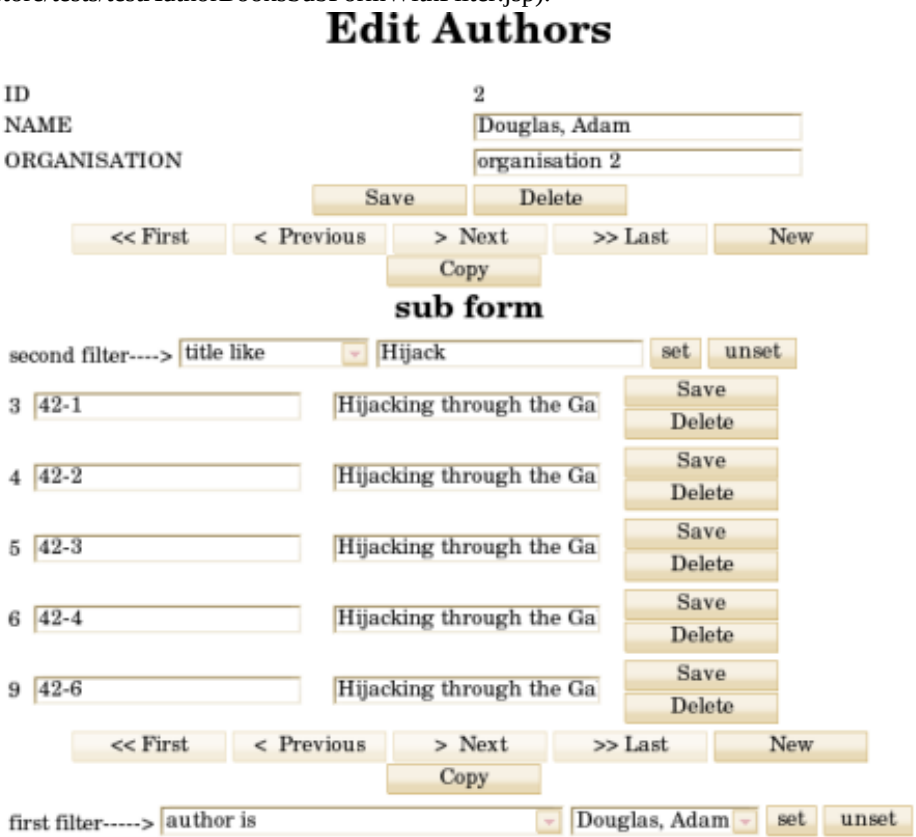
ii. `filter` attribute in dbforms tag

iii. `whereClause` attribute in dbforms tag

12.2.1. The Filter Tag in Use

The `filter` tag allows for easy use of user's input as well as fixed sql conditions.

It will render a select box for the user to choose which filter they would like to use. Once they have chosen, it will (if desired) render a way to input condition values. Finally buttons to "set" or "unset" the filter will be generated. This can be seen twice in the following example (bookstore/tests/testAuthorBooksSubFormWithFilter.jsp).



In the first filter [for the parent form -- here placed in its footer], the user chooses from a select box pop-

ulated with data obtained via a `queryData` tag. Here it is used to select which author is to be viewed.

```
<db:filter>

  <db:filterCondition label="author is">
    NAME = ?
    <db:filterValue type="select">
      <db:queryData name="q1" query="select distinct name as n1, name as n2 from a
    </db:filterValue>
  </db:filterCondition>

  ...(other filterConditions can be included for the user to select from)...
</db:filter>
```

In the second filter [for the subform -- here placed in its the header], the user inputs filter critera into a text box. Here it is used to select titles from the author chosen in the first filter.

```
<db:filter>
  <db:filterCondition label="title like">
    TITLE LIKE %?%
     <db:filterValue/>
  </db:filterCondition>

  ...(other filterConditions can be included for the user to select from)...
</db:filter>
```

See the *`filter`* [../../taglib/DbFormsTags_Frame.html]tag for more information and usage details.

12.3. The filter attribute (in the dbform tag)

(Note: from 1.1.4pr1 there is a new Filter tag that you may want to use instead of using a `filter` attribute.

Just like sorting and searching, filtering of data rows is an important, even essential, issue for many database applications.

Filtering a table-selection is the most typical case of filtering. In its simplest case, the Filter is nothing more than a part of the WHERE clause of the SQL SELECT statement used to retrieve the rows.

This kind of filtering is very easy applied in DbForms. The filter criteria may be passed in the filter attribute of the dbform- root tag.

Example 12.3. An example for static filtering

```
      <db:dbform tableName="employee" maxRows="*" followUp="/employees.jsp"
 autoUpdate="false"
      filter="deptno=101,salary>=30000,name~Peter%">
  ...

</db: dbform>
```

The example above restricts the result set to employees of department 101 with salaries higher or equal 30000 and a name *like* Peter (i.e. Peter Berger, Peter Smith, ...)

The following is a list of the operators that are currently supported by the filter attribute:

Table 12.1. filter operators

Operator	Meaning	Example
=	Equals	aField=value
<>	Not Equal	aField<>value
<	Smaller then	aField<100
>	Greater then	aField>0
<=	Smaller then or equal	aField<=100
>=	Greater then or equal	aField>=0
~	Like	aField~%P%

By default, filter elements are interpreted by dbforms and ANDed together to build a final where Clause. In certain situations, a logical OR is required between individual elements. This is achieved by using the pipe(|) symbol:

aField<10,aField=1,|aField=3

will render the following where Clause

WHERE aField<10 AND aField=1 OR aField=3

Dynamic filtering is possible as well, as the filter-attribute gets dynamically evaluated by the jsp container.

Example 12.4. Example for dynamic filtering

```
<%
// scriptlet determines filter

StringBuffer myFilter = new StringBuffer();

myFilter.append("aField=");

myFilter.append(request.getParameter("SomeVal"));

%>


      <db:dbform tableName="employee" maxRows="*"
followUp="/employees.jsp" autoUpdate="false"
      filter="<%= myFilter.toString() %>"
      >
...

</db: dbform>
```

12.3.1. Filtering (restricting) rows to be manipulated

There are many different reasons why filtering of data is needed. However, in many cases the filtering should not only be applied to data-selection, but also to data manipulating operations such as *insert*s, *update*s or *delete*s.

Scenario:

For example, we want to define a filter to guarantee that certain users may only manipulate data belonging to certain departments: User A may only edit data for employees of department 100 and 200.

The detailed constraints should be defined as follows:

- it is forbidden for the user to insert any new employees with a dept.no. other than 100 or 200.
- it is forbidden for the user to update employee data where the dept_no field has an assigned value other then 100 or 200.
- it is forbidden for the user to delete employees where the dept_no field has a value other then 100 or 200.

We could easily carry out these rules by applying the methods and patterns we learned in the previous chapters:

- set a filter attribute on the <dbform> element
- use a select box or radio buttons for the dept_no field => so that the user has to choose from a set of valid values and has no chance to enter invalid values.

In many cases, this would be sufficient, but to make things really secure and to prevent malicious users from emulating form request using telnet, etc. we need to define these rules in a more secure way.

This approach is described in the chapter on Application Hookups.

12.3.2. Using the whereClause to restrict queries

It is also possible to use a `whereClause` within a dbform tag for situations where you need more complexity than filtering can handle.

However, the whereClause attribute **can not be used with** `orderBy` or `filter` attributes. They will be ignored. Order by should be used within the whereClause instead:

See the *Tag Library* [../../taglib/DbFormsTags_Frame.html] from more info.

12.4. Searching

Searching is another must-have functionality for a database application framework like DbForms.

(Note: from 1.1.4pr1 there is a new Filter tag that you may want to use instead of using search-forms).

DbForms allows you to create search-forms very quickly. The number of fields to be searched, the kind of input widgets (text field, textArea, select box, etc.), the search (compare) algorithms and the search criteria combinations are easily defined by the developer.

All you have to do is:

- decide which fields you want to make searchable.
- decide if you want the user to choose the criteria-combination-mode or if you want to use a (hidden) default.
- decide if you want the user to choose the kind of search algorithm to be used or if you want to use a (hidden) default.
- define a kind of search-button which will submit the search fields to the controller.

12.4.1. Search Criteria

The fields holding search-criteria, search-criteria-combing-mode and search-algorithms **must follow a special naming convention** (in order to allow searching in nested forms, etc). This makes the whole thing look a bit complicated, but it really isnt, as you will see below. This naming convention is prepared for you in hashtable objects accessible as scripting variables. (refer to the chapter on Scripting Variables)

Table 12.2. search naming system

Name of scripting variable Hashtable	Description
searchFieldNames_tableName	Contains the encoded names used for selecting the search criteria fields (firstname, lastname, etc.) Example: searchFieldNames_cust.get("firstname");
searchFieldModeNames_tableName	Contains the encoded names used for specifying the search modes (and, or) of each search criteria Example: searchFieldModeNames_cust.get("firstname")
searchFieldAgorithmNames_tableName	Contains the encoded names used for specifying the algorithm type (weak, sharp) of each search criteria Example: searchFieldAlgorithmNames_cust.get("firstname")

12.4.2. Search Algorithms

The behaviour of searching is selected by different algorithm types:

Table 12.3. search behavior selection

possible values of search field algorithm:	Description - Searching is done with:
sharp	= this is the default behavior
sharpLT	<
sharpLE	<=
sharpGT	>
sharpGE	>=
sharpNE	<>
sharpNULL	is null
sharpNOTNULL	Not is null
weak	like
weakStart	like, % is added in front of value
weakEnd	like, % is appended to value
weakStartEnd	like, % is added in front of value and % is appended to value
sharpExtended	In this mode the search value is parsed for special chars -- see list below. The default search operator is taken from the list above.
sharpLTExtended	
sharpLEExtended	
sharpGTExtended	
sharpGEExtended	
sharpNEExtended	
sharpNULLExtended	
sharpNOTNULLExtended	
weakExtended	
weakStartExtended	
weakEndExtended	
weakStartEndExtended	

If extended mode is set, the values will be parsed for special chars at the beginning:

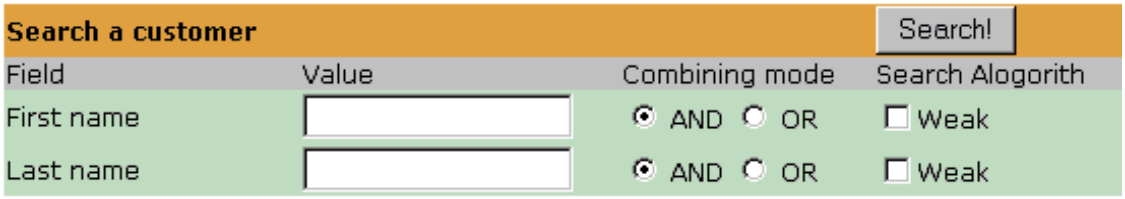
Table 12.4. extended mode searches

Special chars in search values	Description
=	Search operator is sharp
<>	Search operator is sharpNE
!=	Search operator is sharpNE
>=	Search operator is sharpGE
>	Search operator is sharpGT
<=	Search operator is sharpLE
<	Search operator is sharpLT
[NULL]	Search operator is sharpNULL
[!NULL]	Search operator is sharpNOTNULL

value1-value2	Results in fieldname <= value1 and fieldname >= value2
value1-	Results in fieldname <= value1
-value2	Results in fieldname >= value2

12.4.3. Search Example

This is a fully functional search panel. It is included in the example application included in the DbForms distribution. You may also test it on the running live samples at the DbForms Website.



This is the code responsible for creating the panel above. In the context of this example, the use of the hashtable should be clear. As mentioned above, this mechanism may be applied on any other database fields, using any html widgets for data input.

```
<table cellspacing="0" cellpadding="1" width="550">
          <tr bgcolor="#F7A629">
  <td bgcolor="#F7A629"><b>Search a customer</b></td>
  <td colspan="2" bgcolor="#F7A629">nbsp;</td>
  <td bgcolor="#F7A629">
                <input value="Search!"
                           type="button"
                           onClick="javascript:document.dbform.submit()">
              </td>
          </tr>
          <tr bgcolor="#CCBBCC">
  <td>Field</td>
  <td>Value</td>
  <td>Combining mode</td>
  <td>Search Algorithm</td>
          </tr>
          <tr bgcolor="#BBCCBB">
  <td>First name</td>
  <td>
                <input type="text"
                           name="<%=
      searchFieldNames_customer.get("firstname") %>"
                           size="17"
                />
             </td>
  <td>
                <input type="radio" checked
                           name="<%=
      searchFieldModeNames_customer.get("firstname") %>"
                           value="
      and"> AND
                <input type="radio"
                           name="<%=
      searchFieldModeNames_customer.get("firstname") %>
                           value="
      or"> OR
  </td>
  <td>
```

```
                <input type="checkbox"
                                name="<%=
    searchFieldAlgorithmNames_customer.get("firstname") %>"
                                size="30"
                                value="
    weak">Weak
</td> .
        </tr>
        <tr bgcolor="#BBCCBB">
            <td>Last name</td>
<td>
            <input type="text"
                           name="<%=
    searchFieldNames_customer.get("lastname") %>"
                           size="17"/>
            </td>
<td>
                <input type="radio" checked
                                name="<%=
    searchFieldModeNames_customer.get("lastname") %>"
                                value="
    and"
                > AND
                <input type="radio"
                                name="<%=
    searchFieldModeNames_customer.get("lastname") %>
                                value="
    or"
                > OR
</td>
<td>
                <input type="checkbox"
                                name="<%=
    searchFieldAlgorithmNames_customer.get("lastname") %>"
                                size="30"
                                value="
    weak"
                >Weak
</td>
        </tr>
```

Chapter 13. Query Support

To make the usage of sql queries more flexible in dbforms, you can use the query element which can be used to:

- Create *group by* with flexible *where* part.
- Create joined sql with a flexible *join* statement
- Create an alias of an existing table
- Make dynamic changes of the query which prevents the use of a view in the database.

13.1. Examples:

13.1.1. group by

In a table with:

```
create table
            values
    (cno integer not null primary key,
        type char(20) not null,
        date timestamp not null,
        value number not null
   )
```

You need a query with:

```
Select type, sum(value) as sumValue where date < ?
group by type having sumValue > ?
```

Because the date should be selected by the user, you can not use a fixed view in the database. However, this is possible using the **query** element in dbforms_config.xml:

```
            <dbforms-config>
...
            <query name="valuesGroup" from="values" groupBy=type>
    <field="type" type="char">
    <field="sumValue" expression=sum(value) type="number">
```

```
    <search="date" type="timestamp">
</query>
```

13.1.2. join

```
            <dbforms-config>
...
            <query name="viewname" from="table1 left join table2 on table1.i
    <field1 type=char>
</query>
```

13.1.3. Simple alias

```
            <dbforms-config>
...
            <query name="aliasname" from="table" />
```

- can be used to have the same table more than once on a page.
- inherits all field definitions from the table.
- is updateable!
- may be another query.

13.2. Use within dbforms-config.xml

The following example is taken from `src\org\dbforms\resources\dbforms-config.xsd` It has been changed here from schema style to DTD sytle for those more familiar with the latter. However, no DTD for dbforms.xml actually exists.

Table 13.1. Excerpts in DTD style of dbforms-config.xml

<!ELEMENT query (field*, search*)>	
<!ATTLIST query	
name CDATA #REQUIRED	Name of the table, same meaning as in table attribute
from CDATA #IMPLIED	Name of the parent table. This table must be

	defined within dbforms-config.xml. If you leave out everything else, query is used as an updatable alias for the prior defined table! If missing,
groupBy CDATA #IMPLIED	Group by clause for select
where CDATA #IMPLIED	Starting where part of select
followAfterWhere CDATA #IMPLIED	Must be OR or AND, will be appended to the starting where part if dbforms have build an own where part: Where + followAfterWhere + where_from_dbforms
orderWithPos CDATA #IMPLIED	If true, the order part will build with fieldnumber instead of fieldname
distinct CDATA #IMPLIED	If true, select will be build as select distinct
>	
<!ATTLIST field	
name CDATA #REQUIRED	Name of field in table
type CDATA #IMPLIED	Type of field
autoInc CDATA #IMPLIED	stores if the field is AUTOINCremental
key CDATA #IMPLIED	=TRUE if field is key field
sortable CDATA #IMPLIED	all non-key-fields you want to be sortable should be declared #sortable# in the XML-config file
expression CDATA #IMPLIED	Expression of field. Results in expression + AS + name
>	
<!ELEMENT search EMPTY>	All elements of type search can be used as normal fields in dbforms. While construction the sql statement these fields will not be included in the from part. They will only be used to generate the where part of the query. If the query has a groupBy statement, these fields will be used in the where part of the query if needed, not in the having part!
<!ATTLIST search	
name CDATA #REQUIRED	Name of field in table or alias
expression CDATA #IMPLIED	Expression of field. Results in expression + AS + name
>	

Chapter 14. Scripting Variables

Various DbForms tags such as <dbform> and <body>, make internal information available to the embedded JSP code via **scripting variables.**

This mechanism used for passing these values over is called **Tag Extra Info** and is part of the Java Server Pages specification (refer to spec version 1.1, chapter 5.5)

Please note:

The list of scripting values is still growing and some names may be changed in the near future.

14.1. tableName known at compile time

Table 14.1. List of scripting variables

Scripting variables	Type	Description
searchFieldNames_tableName	java.util.Hashtable	Contains the encoded names used for selecting the search criteria fields (firstname, lastname, etc.) Scope: inside the respective DbForms- element
searchFieldModeNames_ tableName	java.util.Hashtable	Contains the encoded names used for specifying the search modes (and, or) of each search criteria. Scope: inside the respective DbForms- element
searchFieldAgorithmNames_tableName	java.util.Hashtable	Contains the encoded names used for specifying the algorithm type (weak, sharp) of each search criteria. Scope: inside the respective DbForms-element
currentRow_tableName	java.util.Hashtable	Contains the field-values of the current row. This construct is similar to #associated Arrays used in many Perl/PHP modules Example: String email = (String) curren-tRow_customers.get("email"); Scope: inside the respective <body>-tag
position_tableName	java.lang.String	Contains the encoded key-fieldvalues of the current row. Scope: inside the respective <body>-element
rsv_tableName	org.dbforms.util.ResultSetVector	Refers to the ResultSetVector containing the data to display. Use the methods getField() get-

		FieldAsObject to retrieve data from the vector. Example: if we want to print out a string representation of the current value of the field "reporter" in table "bugs", then we would write: <%= rsv_bugs.getField("reporter")%> we also could write: <%= rsv_bugs.getFieldAsObject("reporter") %> what is the difference between getField(String) and getFieldAsObject(String): getField always returns a String (as long as the fieldname is correct), even it the underlying data in the database has the value NULL. getFieldAsObject returns the Object value as provided by the ResultSet.getObject(int) method. This means NULL-values will be returned as null and not as empty ""-Strings.

14.2. tableName determined at runtime

If the tableName is determined at runtime, the well known dbforms variables, e.g. `currentRow_tablename` will not work. So there is a new variable named `dbforms` which is a hashtable that holds a list of all known dbforms and subforms in the page. Key of the map is the new `db:dbforms` attribute `name`. Objects of this list are of the type `DbFormContext`, which will give the access to all of the known fields:

```
Public class DbFormsContext {
   public String getPosition();
   public ResultSetVector getRsv();
   public Map getSearchFieldAlgorithmNames();
   public Map getSearchFieldModeNames();
   public Map getSearchFieldNames();
   public Map getCurrentRow();
   public void setCurrentRow(Map map);
   public void setPosition(String string);
}
```

Any form with a subform on it would be ok too.

See `DbFormTagTEI` class and the test case (in the bookstore example) `testAuthorBooksSubFormWithContextVar.jsp` for details!

For example:

```
<db:dbform   ...
               tableName='<%=determined_at_runtime%>'
               name="AUTHOR"
     >
...
```

```
 <db:body>
...
    <%=((DbFormContext)dbforms.get("AUTHOR")).getCurrentRow().get("AUTHOR_ID")%>
...
   </db:dbform
```

Chapter 15. Application hook-ups (Interceptors)

15.1. Introduction

It would be neither possible nor useful to create a system which has a solution, for every problem or requirement which could appear during the development-process (and life-cycle) of a database application.

Because it is impossible to foresee all eventual use cases and user needs, an 'all-in-one' product would most likely restrict the application developer sooner or later.

In order to handle the all-to-familiar unknown, systems must offer a kind of programming facility or Application Programmer Interface. DbForms offers the following:

15.2. Interface DbEventInterceptor

DbForms provides the `DbEventInterceptor` interface, which is capable of intercepting database operations (before and after they are executed).

This interface provides the following methods

Example 15.1. Methods defined in interface DbEventInterceptor

```
public int preInsert(HttpServletRequest request, Table table, FieldValues fiel
            DbFormsConfig config, Connection con)
            throws ValidationException;


public void postInsert(HttpServletRequest request, DbFormsConfig config, Conne


public int preUpdate(HttpServletRequest request, Table table, FieldValues fiel
            DbFormsConfig config, Connection con)
            throws ValidationException;


public void postUpdate(HttpServletRequest request, DbFormsConfig config, Conne


public int preDelete(HttpServletRequest request, Table table, FieldValues fiel
            DbFormsConfig config, Connection  con)
            throws ValidationException;


public void
    postDelete(HttpServletRequest request, DbFormsConfig config, Connection co


public int
    preSelect(HttpServletRequest request, DbFormsConfig config, Connection con
            throws ValidationException;
```

```
public void
    postSelect(HttpServletRequest request, DbFormsConfig config, Connection co


    (Note: the following pre/postAddRow methods are used to make a custom filt
    sum values, or just merge some columns which can not be merged in sql.  Se
    below.  Also, they are useful for counting rows -- see howtoAddRowCountSup

public int preAddRow(DbEventInterceptorData data)
            throws ValidationException, MultipleValidationException;


public void postAddRow(DbEventInterceptorData data);
```

As the names indicate:

- The **preXxx() methods** get called BEFORE the respective database operation is performed, and **return a value** indicating if the operation should be performed or not

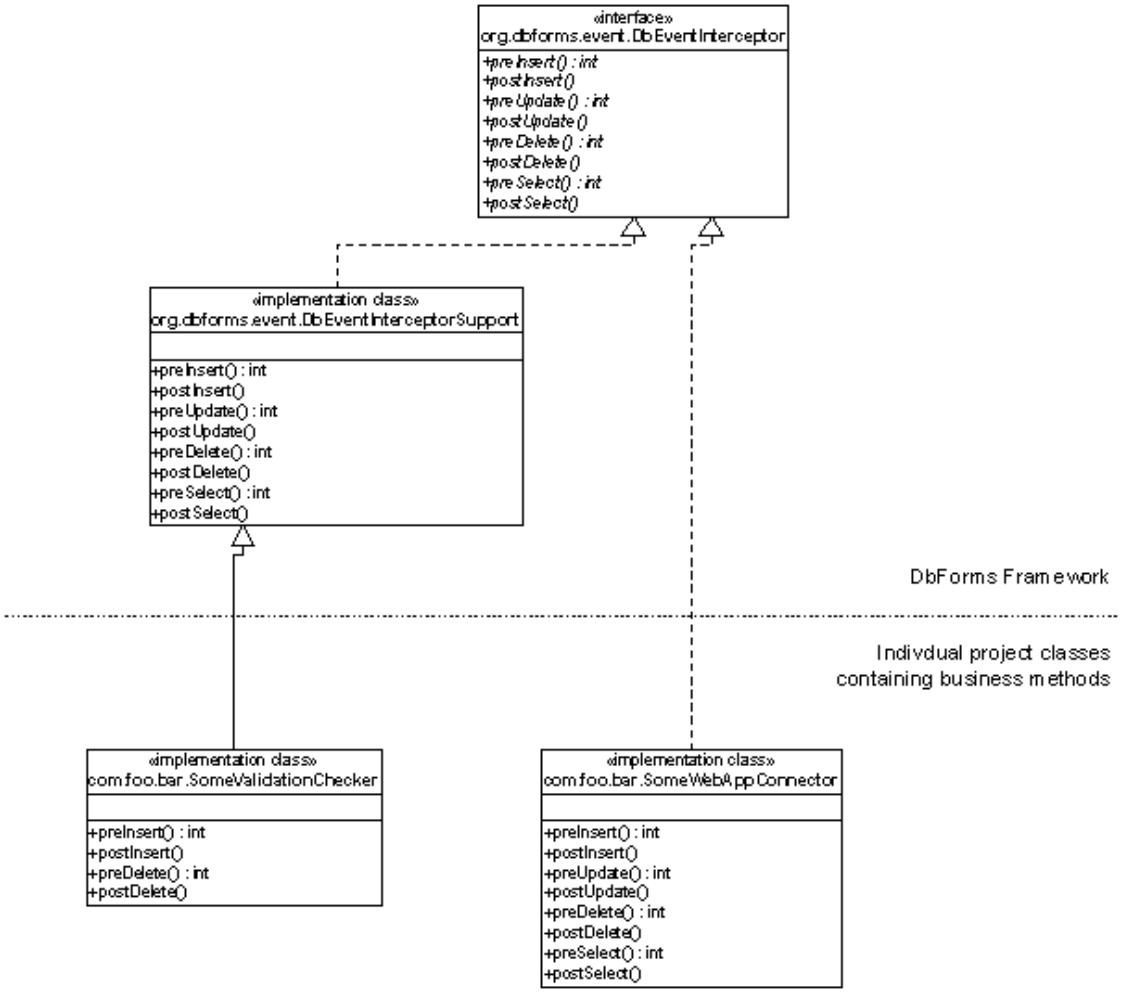
  There are three possible values which could be returned, as well as an exception that could be thrown:

`DENY_OPERATION`	If `DENY_OPERATION` is given back by the interceptor, a SQL exception is raised and the transaction is rolled back if set to `autoCommit = false`
`GRANT_OPERATION`	If `GRANT_OPERATION` is returned by the interceptor, the event will be performed and in cases where `autoCommit = false`, the transaction will be committed.
`IGNORE_OPERATION`	If `IGNORE_OPERATION` is returned by the interceptor, the DBForms event (UPDATE, INSERT, SELECT, DELETE) will not be performed, However in cases where `autoCommit = false`, the transaction will be committed. For **preAddRow** and **postAddRow** methods, returning `IGNORE_OPERATION` means that the row is not written to the ResultSetVector and allows you to make filters which are not possible via SQL alone.
`ValidationException`	If the Interceptor raises a `ValidationException` the transaction is rolled back too. So if you have an exception inside your interceptor, you have two possibilities: Return `DENY_OPERATION` which generates `SQLException` with an NOGRANT message or throw a `ValidationExeception` with your own error text.

- The **postXxx()** methods get called AFTER the operation has finished, and **do not return a value**, as the operation has already completed.

15.3. Class Diagram

Figure 15.1. UML diagram of the DbEventInterceptor interface and some of its implementations



As you can see, there exists an interface `DbEventInterceptor` as described above. Additionally there exists a built-in implementation class `DbEventInterceptorSupport` of this interface. This is a convenience class which allows the developer to override just the methods he/she is interested in. There is no other difference between these two entities.

15.4. Method parameters

Table 15.1. Parameters available in methods of DbEventInterceptor

Parameter	Description
Connection con	a database connection => this object makes it easy to "trigger" other functionality like database-logging, keep up special data constraints, execute a

	sub-query, etc.
DbFormsConfig config	a config object provides the method DbFormsConfig.getServletContext() which gives access to other J2EE-resources which may be stored in application scope. This could be an entry point to any other application, including EJB-Apps, Struts-Apps, Messaging systems, etc.
HttpServletRequest request	an http-request object gives access to various types of information, including the current user#s name and user#s groups/roles.

FieldValues fieldValues:	in the preInsert(), preUpdate() and preDelete()-methods: FieldValues which gives access to the fields contained in the current (selected) row.
	
	`(String) fieldValues.get("lastname").getFieldValueAsObject();`
	
	-or-
	
	`fieldValues.get("lastname").getFieldValue();`
	
	will return the field value of the field #lastName# of the row DbForm's is about to insert, to update or to delete. Note: `...getFieldValue();` always returns a String.
	
	To make changes in preInsert() and preUpdate() so that changes in the FieldValues are reflected in the database, use `setValue(Table table, FieldValues fieldValues, String fieldName, String value)` as shown below:
	
	`this.setValue(table, fieldValues, "lastname", "smith");`
	
	will ensure that the value #smith# be stored in the lastname field in the database. (column lastname must exist!)

DbEventInterceptorData data	the `DbEventInterceptorData` contains the same data as the above parameters and can be accessed by using: • data.getConnection() • data.getConfig() • data.getRequest() • data.getTable()

	• (FieldValues) data.getAttribute(DbEventInterceptorData.FIELDVALUES)

15.5. Installing Interceptors

How do we tell DbForms *when* to invoke *which* Interface implementation?

This information is provided in the dbForms XML configuration file. Similar to the granted-privileges security-constraint, (described in the security chapter) the XML tag which defines an Interceptor has to be placed inside a <table> element.

15.5.1. Table by Table Configuration

Example 15.2. interceptor in dbforms-config.xml

```
      <table name="customer">
      <field name="id" fieldType="int" isKey="true"/>
<field name="firstname" fieldType="char" />
<field name="lastname" fieldType="char" />
<field name="address" fieldType="char" />
<field name="pcode" fieldType="char" />
<field name="city" fieldType="char" />


      <interceptor className = "com.foo.bar.CustomerValidatonChecker"/>


      <interceptor className = "com.foo.bar.TransactionLogger"/>

  </table>
```

The semantics of these declarations could be described as follows:

Invoke `com.foo.bar.CustomerValidatonChecker` *and* `com.foo.bar.TransactionLogger` *, if the user is about to read, insert, update or delete data from table* `customer and call the appropriate methods on those objects`

15.5.2. Global Configuration

Interceptors can also be configured in `dbforms-config.xml` globally.

```
    <dbforms-config>
        <interceptors>
           <interceptor className = "com.foo.bar.CustomerValidatonChecker"/>
               </interceptors>
</dbforms-config>
```

15.5.3. Both Table by Table and Global Configured

If both ways are used, then the*Table by Table interceptor* will be processed first, and the *Global interceptor* will be processed after that. This means of course that changes to your `FieldValues` in the *Table by Table interceptor* get passed to the *Global interceptor*.

15.6. Example

In this example, we override 2 methods (preInsert and preUpdate) to do some basic validation checking.

Example 15.3. Validation checking in interceptor

```
import org.dbforms.config.*;
import org.dbforms.event.*;
import java.sql.*;
import java.util.*;
import javax.servlet.http.*;

    public class CheckCustomerData
        extends DbEventInterceptorSupport {

        private int
        checkCustomer(FieldValues fieldValues) // method invented by developer
            throws ValidationException {

                String lastName = (String) fieldValues.get("lastname").getFieldVal

                String pCode = (String) fieldValues.get("pcode").getFieldValue();

                String city = (String) fieldValues.get("city").getFieldValue();

                // perform form-validation
                if (lastName == null || lastName.trim().length()==0 ||
                    pCode == null || pCode.trim().length()==0 ||
                    city == null || city.trim().length()==0)  {

                        throw new ValidationException("Please fill out the form co

                else return GRANT_OPERATION;

            }


        public int
        preInsert(HttpServletRequest request,  Table table, FieldValues fieldValue
         DbFormsConfig config, Connection con)
            throws ValidationException {


        return checkCustomer(fieldValues);

            }


        public int
```

```
    preUpdate(HttpServletRequest request, Table table, FieldValues fieldValues
       DbFormsConfig config, Connection con)
        throws ValidationException {


    return checkCustomer(fieldValues);
        }


}
```

Remarks:

There is an Exception -- `org.dbforms.config.ValidationException` which may be thrown to signal a message to the end-user. In fact, returning the integer constant `DENY_OPERATION` defined in `DbEventInterceptor` leads to similar results (but with an unspecific error message and no generic message)

15.7. Accumulating error messages

It is now possible to accumulate error messages while validating a form in a dbForms interceptor class. This is useful in situations where many fields are validated and you wish to return several errors instead of just the first occurrence. In the validation process, when an error occurs, simply instantiate a new exception object and store it into a vector. Do not throw this exception! Once the validation process has completed, throw an instance of the MultipleValidationException class and pass the vector as an argument to its constructor. Note that this functionality is dependant on using the xmlError tag. (refer to the section: DbForms Custom Tag Library for more information on using the xmlError tag)

Example 15.4. accumlating error messages

```
public int
    preUpdate(HttpServletRequest req, FieldValues fieldValues, DbformsConfig c
  Connection con)
    throws ValidationException{

        Vector
    errors = new Vector();

         // Validate, if error

    errors.add(new SQLException("English-001:param");


         // More validations, errors

    errors.add(new SQLException("Franais-003:param,param");


        if(!errors.isEmpty)
    throw new MultipleValidationException(errors);


    }
```

15.8. Changing Key Values in an Interceptor

Although you could try and do it via a pluggableEvent, as it currently stands this can not be done.

Apparently if a field is marked with isKey="true" in the config file, the preDelete and preUpdate methods disregard any changes to it.

Carlos Anjos pointed out:

I had a situation where I did some processing in the preDelete method and then I wanted to allow the operation, but on a different row. So I changed the values in fieldValues, but still the orginal row got deleted.

After banging my head for where in my code the error was, I finally understood that dbforms was ignoring the change to that attribute. A friend of mine went through dbforms deleteEvent code and confirmed that.

Chapter 16. Internationalization and i18n Support

16.1. Specifying a charset

To specify a specific charset, do the following [note this example will use Shift_JIS (Japanese) and MYSQL]:

- Set the charset in your .jsp page:

```
<%@page contentType="text/html;charset=Shift_JIS"%>
```

- Adjust your driver's URL

```
<dbconnection
  id="edict"
          name   = "jdbc:mysql://localhost/edict?useUnicode=true&amp;chara
          isJndi = "false"
          conClass  = "org.gjt.mm.mysql.Driver"
          username = "whoever"
          password  = "whatever"
  />
```

Note: to use UTF-8 with MYSQL, you need to use:

```
jdbc:mysql://localhost/edict?useUnicode=true&amp;characterEncoding=UTF-8&a
```

- Add a filter to `web.xml`

```
<filter>
   <filter-name>Set Character Encoding</filter-name>
   <filter-class>org.dbforms.util.external.SetCharacterEncodingFilter</fil
   <init-param>
      <param-name>encoding</param-name>
      <param-value>Shift_JIS</param-value>
   </init-param>
</filter>
```

```
<filter-mapping>
   <filter-name>Set Character Encoding</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

16.2. Defining Resource Bundles

Resource bundles are made up of 1 or more .properties files. Each file contains a key-value reference which is used to retrieve and display text in a given language. Properties files, defined to be of the same resource bundle, have the following naming structure:

- A common prefix is used to associate files into a bundle.
- The locale, for which this file is valid for, is used as a suffix.
- A default properties file exists in order to support non-defined locales. This file does not have a (locale) suffix associated to its name. All files must be located in a directory which is in the classpath.

ApplicationResources.properties

```
msg.bienvenue.1=<B>Bienvenue dans l'application XYZ</B>
msg.bienvenue.2=Heureux de vous servir en franais.
boutton.login=Authentifier
boutton.logout=Quitter
```

ApplicationResources_fr_CA.properties

```
msg.bienvenue.1=<B>Bienvenue dans l'application XYZ</B>
msg.bienvenue.2=Heureux de servir nos amis Canadiens en franais.
boutton.login=Authentifier
boutton.logout=Quitter
```

ApplicationResources_en.properties

```
msg.bienvenue.1=<B>Welcome in XYZ application</B>
msg.bienvenue.2=Happy to serve you in english.
boutton.login=Login
boutton.logout=Logout
```

ApplicationResources_es.properties

```
msg.bienvenue.1=<B>Recepcin en la aplicacin XYZ</B>
msg.bienvenue.2=Feliz servirle en espaol
boutton.login=Conexin
boutton.logout=Fin de comunicacin
```

In this example, the default language is French. In addition to the default language, this application supports English and Spanish. For French speaking clients from Canada, a special resource has also been developed.

16.3. Setting up DbForms

To define that a resource bundle is to be used within a dbForms application, the resource bundle root name must be specified in the web.xml deployment descriptor (associated to the dbForms config servlet):

```
  <web-app>
    <!--=========== DbForms Configuration Servlet ===========-->

  <servlet>
        <servlet-name>org.dbforms.ConfigServlet</servlet-name>
        <servlet-class>org.dbforms.ConfigServlet</servlet-class>
<init-param>
              <param-name>log4j.configuration</param-name>
              <param-value>log4j.properties</param-value>
</init-param>

  <init-param>
              <param-name>
  resourceBundle</param-name>
              <param-value>
  ApplicationResources</param-value>

  </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
```

16.4. Using i18n within DbForms

Instead of 'hard coding' messages, labels, etc., dbForms has made available the 'message' tag. This tag can be used to display data retrieved via a resource bundle. The actual properties file used to retrieve the text is dependant on the end-user's locale-setting.

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>

    <! - - Retrieving a message  - - >
```

```
<db:message key="msg.bienvenue.1"/>

  <BR><BR>

  <center>
<db:message key="msg.bienvenue.2"/></center>

  <BR><BR>
```

Another feature supported, is the ability to have dbForm button tags display language-specific captions. This is achieved by setting the 'captionResource' attribute (in the dbform tag) to true. When this feature is turned on, text in the caption attribute (in a button tag) is recognized as a key and is used to lookup text in the appropriate resource bundle. If the key cannot be found, the caption value is displayed as is.

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
<! - - Lookup caption in associated resource bundle - - >

    <db:form  tableName="myTable"
  captionResource="true" >

        <db:gotoButton caption="
  button.login" destination="/login.jsp" />

        <db:gotoButton caption="
  button.logout" destination="/logout.jsp" />

    </db:form>
```

16.5. Setting the Pattern Attribute

Many tags make use of the `pattern` attribute. For example,

```
<dateField pattern="date"
```

To set values globally for each type and language, you can define `dbforms.pattern.timestamp` `dbforms.pattern.date` etc. For Example:

ApplicationResources.properties

```
dbforms.pattern.date=yyyy-MM-dd
dbforms.pattern.timestamp=yyyy-MM-dd HH:mm
```

If you use the abbriviations SHORT, MEDIUM, FULL you can define them too:

```
dbforms.pattern.timestamp.short
```

`dbforms.pattern.date.full.`

As defaults, Dbforms is using the following if you do not specify the patterns for each field individually.

```
public static final int DATE_STYLE_DEFAULT = java.text.DateFormat.MEDIUM;
public static final int TIME_STYLE_DEFAULT = java.text.DateFormat.SHORT;
```

16.6. Additional Information

An article on using Unicode with DbForms 2.4rcX is available at either:

- *javaworld* [http://www.javaworld.com/javaworld/jw-09-2004/jw-0906-unicode.html]
- `dbforms/articles/usefull/Unicode_development_with_DbForms.html`

Chapter 17. Validation Framework

17.1. Commons-Validator framework

After having evaluated several validation frameworks, we finally decided to integrate Apache's Commons-Validator into DbForms. This framework is capable of performing basic server-side validations such as verifying if a field is required, or that it matches a regular expression. Different validation rules can be defined for different locales. Although the framework already provides several basic validation routines, custom validation routines can be created and added to the framework.

This framework also supports client-side validation through the generation of JavaScript methods. Use of these methods is enabled via an attribute setting within dbForms.

Commons-validator uses two xml files for setting up validation:

i. Validator-rules.xml

Contains general validation rules.

ii. Validation.xml

This file is used to associate validation rules to table fields.

17.2. Setting up DbForms

To define that the validation framework is to be used within a dbForms application, the associated XML files must be specified in the web.xml deployment descriptor (associated to the dbForms config servlet):

```
  <web-app>
    <!--=========== DbForms Configuration Servlet ===========-->

  <servlet>
        <servlet-name>org.dbforms.ConfigServlet</servlet-name>
        <servlet-class>org.dbforms.ConfigServlet</servlet-class>
<init-param>
                <param-name>log4j.configuration</param-name>
                <param-value>log4j.properties</param-value>
</init-param><init-param>
                <param-name>resourceBundle</param-name>
                <param-value>ApplicationResources</param-value>
</init-param>

  <init-param>
                <param-name>validation</param-name>
                <param-value>/WEB-INF/validation.xml</param-value>
  </init-param>

  <!--More than one validation file is possible-->
  <!--init-param>
                <param-name>validation</param-name>
                <param-value>/WEB-INF/validation.xml,/WEB-INF/custom/validatio
  </init-param-->
```

```
<init-param>
                <param-name>validator-rules</param-name>
                <param-value>/WEB-INF/validator-rules.xml</param-value>

</init-param>
      <load-on-startup>1</load-on-startup>
  </servlet>
```

Version 1.1 of DbForms has been upgraded to use the Apache commons framework. This framework required Java XML Parser (JAXP) v1.1. Older Web containers (such as Tomcat v3.2.x) are still using JAXP v1.0 technology and are therefore incompatible.

17.3. Using validation within DbForms

Using validation within dbForms is as simple as setting an attribute in the 'dbform' tag. In dbForms v1.1, the attribute 'formValidatorName' has been added to allow a dbForm developer to associate a (dbForms) form to an [entry] in the Validation.xml file. Take for example the following JSP:

```
<%@ taglib uri="/WEB-INF/dbforms.tld" prefix="db" %>
    <db:dbform  tableName="bugs" captionResource="true"
    formValidatorName="bugs"
      javascriptValidation="true">

        <table>
            <tr>
    <td>
                <db:textField fieldName="title" size="40"/>
    </td>
            </tr>
        </table>
    <br>
        <db:updateButton  caption="Save" /><br>

        <db:deleteButton  caption="Delete"/><br>

    </db:form>
```

The 'formValidationName' attribute is set to 'bugs'. As you will see in the next section, the Validation.xml file includes an [entry] for 'bugs'. This [entry] will specify which fields will be validated, which validation(s) will be performed on each field and what error messages to display if the validation(s) fail.

Another feature which has been added to dbForms is support for client-side javaScript validation. When the attribute 'JavaScript Validation' is set to 'true', DbForms will generate methods to automatically validate client-side using the same rules found in Validation.xml.

17.4. Generating Validation.xml

generate_validation.xsl can now (included from dbforms 1.3) be used to generate the `validation.xml` file that dbforms uses for validation. (The source XML file is `dbforms-config.xml`.)

Please note that we added the attribute generate-jsp (true/false) to table tags because there were a few tables we wanted in the `dbforms-config.xml` file for foreign-key reasons but we didn't want tables to be generated. The line test="not(@generate-jsp='false')" returns true when (a) generate-jsp is set to

some value other than false or (b) when it doesn't exist. (generate-jsp!='false' by comparison returns true only when generate-jsp is set to something other than false.) This way, its functionality is retained while making it an optional attribute as far as `DevGui is concerned.`

Limitations:

i. Assumes that all foreign-keys are required and that the server will do its own testing for valid entry.

ii. Asserts that isKey'd columns that aren't also autoInc'd are

- required
- in need of maxlength validating

iii. the resulting validation.xml asserts that the database server will insure that there are no key column conflicts

17.5. Validation.xml Tags

Various tags are used within the Validation.xml file to describe validation rules. The following describes each tag in detail:

Table 17.1. Validation Tag Details

Tag Name	Description
formset	This tag is used to support i18n. Suppose that a form is to be validated and that a field within this form is a postal code. Depending upon the locale, this postal code could have very different formats (US/Canadian/German). Multiple formset tags can be defined and each, can be associated to a different locale. Association may be defined using either of the following attributes: • country • language If an un-specified locale is requested, the default (no attributes) formset entry is executed.
form	This tag corresponds to the value of the dbform tag 'formValidatorName' attribute.
field	This tag identifies the table field. (refer to db-forms-config.xml)
depends	Defines which type(s) of validation(s) are required for the field.
msg	This tag specifies the error message to display if the validation fails. The following attributes are required: • name

	Specifies validation type. • resource Determines if the key should be resolved using i18n. • key If 'resource = false': Corresponds to error message. If 'resource = true': key is used to lookup error message in appropriate resource bundle.
arg	Is used to pass arguments to messages. The following attributes are required: • name Specifies validation type. • resource Determines if the key should be resolved using i18n. • key If 'resource = false': Argument sent as is. If 'resource = true': key is used to lookup error message in appropriate resource bundle.
var	Defines variables for use in validation.

17.6. Validation.xml file

In a previous section, we defined that our (dbform) form would be associated to an [entry] called 'bugs'. The 'bugs' form contained a single textField entry (referring to 'title'). Let's take a look at the resulting **Validation.xml** file:

```
<formset>
<form name="bugs">
```

```
      <field property="title" depends="required,mask,maxlength">
      <msg name="required" key="errors.required" resource="true"/>
      <arg0 name="required" key="bugsForm.title.displayname" resource="true"/>

      <msg name="mask" key="bugsForm.error.title.mask" resource="true"/>

      <msg name="maxlength" key="errors.maxlength" resource="true"/>
      <arg0 name="maxlength" key="bugsForm.title.displayname" resource="true"/>
      <arg1 name="maxlength" key="${var:maxlength}" resource="false"/>

      <var>
                  <var-name>mask</var-name>
                  <var-value>^[A-Z][A-Za-z0-9\s]*$</var-value>
      </var>
      <var>
                  <var-name>maxlength</var-name>
                  <var-value>10</var-value>
      </var>
            </field>
  </form>
    </formset>
```

Its content may be summarized as follows:

- The formset is not associated to any country or language therefore it will be used for all locales.
- The 'value' input in the (dbForms) TextField which represents the 'title' column will be validated as follows:
  - Value is required
  - Value must start with an uppercase character and contain alphanumeric characters only.
  - Value+ must not contain more then 10 characters.
- Most messages (and arguments) are resolved using i18n except for 'arg1' of 'maxlength' which uses a locally-defined variable.

17.7. Types of validation available

The following shows a listing of the various types of validations possible. Of course, the developer is free to add custom validation-rules as needed.

17.7.1. required

Value is required, and must be input by the end-user.

ApplicationResource.properties:

```
errors.required={0} is required.
myForm.title.displayname=Title
```

Validation.xml:

```
<field property="title" depends="required">
                <msg name="required" key="errors.required" resource="true"/>
                <arg0 name="required" key="myForm.title.displayname" resource=
</field>
```

Result when error:

```
Title is required.
```

17.7.2. mask

Validation made via a "regular expression".

ApplicationResource.properties:

```
myForm.error.title.mask=The Title must begin with an uppercase cha
 and contain alphanumeric characters only.
```

Validation.xml:

```
<field property="title" depends="mask">
                <msg name="mask" key="myForm.error.title.mask" resource="true"
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[A-Z][A-Za-z0-9\s]*$</var-value>
                </var>
</field>
```

Result when error:

```
The Title must begin with an uppercase character
 and contain alphanumeric characters only.
```

17.7.3. range

Validation made via a "regular expression".

ApplicationResource.properties:

```
errors.range={0} must be between {1} and {2}.
myForm.age.displayname=age
```

Validation.xml:

```
<field    property="age" depends="range">

                <msg name="range" key="errors.range"/>
                <arg0 name="range" key="myForm.age.displayname" resource="true
                <arg1 name="range" key="${var:min}" resource="false"/>
                <arg2 name="range" key="${var:max}" resource="false"/>

                <var>
                    <var-name>min</var-name>
                    <var-value>1</var-value>
                </var>

                <var>
                    <var-name>max</var-name>
                    <var-value>110</var-value>
                </var>

</field>
```

Result when error:

```
age must be between 1 and 110.
```

17.7.4. byte, short, long, integer, double, float

Validation to insure a specific data type. Each of these validations may be used separately.

ApplicationResource.properties:

```
errors.integer={0} should be of type integer.
myForm.age.displayname=age
```

Validation.xml:

```
<field property="age" depends="integer">
    <msg name="integer" key="errors.integer"/>
    <arg0 name="integer" key="myForm.age.displayname" resource="tr
</field>
```

Result when error:

```
age should be of type integer.
```

17.7.5. minlength

Validates if the number of characters input by the user, respects the minimum number required.

ApplicationResource.properties:

```
errors.minlength={0} cannot contain less than {1} characters.
```

```
myForm.description.displayname=Description
```

Validation.xml:

```
        <field property="description" depends="minlength">
            <msg name="minlength" key="errors.minlength"/>
            <arg0 name="minlength" key="myForm.description.displayname" re

            <var>
                <var-name>minlength</var-name>
                <var-value>25</var-value>
            </var>
</field>
```

Result when error:

```
        Description cannot contain less than 25 characters.
```

17.7.6. maxlength

Validates if the number of characters input by the user, respects the maximum number required.

ApplicationResource.properties:

```
        errors.maxlength={0} cannot contain more than {1} characters.
myForm.description.displayname=Description
```

Validation.xml:

```
        <field property="description" depends="maxlength">
            <msg name="maxlength" key="errors.maxlength"/>
            <arg0 name="maxlength" key="myForm.description.displayname" re

            <var>
                <var-name>maxlength</var-name>
                <var-value>25</var-value>
            </var>
</field>
```

Result when error:

```
        Description cannot contain more than 25 characters.
```

17.7.7. date

Validates if the date input is a valid date. This validation requires two arguments: 'SimpleDateFormat' and a (date) pattern. Valid patterns include:

- datePattern
- datePatternStrict which validates length.

Hence, if the SimpleDateFormat is set to 'yyyy-MM-dd' and the user input '2002-4-4', using 'datePattern-Strict' would return an error. (Expecting 10 characters got 8)

ApplicationResource.properties:

```
errors.date={0} is not a valid date.
myForm.opendate.displayname=Start Date
```

Validation.xml:

```
<field property="opendate" depends="date">
    <msg name="date" key="errors.date"/>
    <arg0 name="date" key="myForm.opendate.displayname" resour

    <var>
        <var-name>datePatternStrict</var-name>
        <var-value>yyyy-MM-dd</var-value>
    </var>
</field>
```

Result when error:

```
Start Date is not a valid date.
```

17.7.8. email

Validates an e-mail address.

ApplicationResource.properties:

```
errors.email={0} does not contain a valid e-mail address.
myForm.email.displayname=User E-mail address
```

Validation.xml:

```
<field property="email_user" depends="email">
    <msg name="email" key="errors.email"/>
    <arg0 name="email" key="myForm.email.displayname" resource="tr
</field>
```

Result when error:

```
User E-mail address does not contain a valid e-mail address.
```

17.7.9. creditcard

Validates a credit card number (using checksum).

ApplicationResource.properties:

```
errors.creditcard={0} does not contain a valid credit card number.
myForm.creditcard.displayname=Client credit card
```

Validation.xml:

```
<field property="email_user" depends="creditcard">
                <msg name="creditcard" key="errors.creditcard"/>
                <arg0 name="creditcard" key="myForm.creditcard.displayname" re
</field>
```

Result when error:

```
Client credit card does not contain a valid credit card number.
```

Please refer to the sample application 'bugtracker' for a detailed exemple of using validation within db-Forms. This application can be found in the distribution package.

17.8. Validator-rules.xml

A sample is included in the bugtracker example in the `/examples` directory.

In addition, a sample `validation.js` is available there as well. This is a contecnation of `validator-rules.xml` used for performance reasons via the `javascriptValidationSrcFile` attribute of the `dbform` tag.

Finally, since DBF's validation was based on Struts, you can also check out *the validation chapter from "Struts In Action"* [articles/validating_user_input.pdf]

Chapter 18. JavaScript Calendar Application

18.1. What is it?

DbForms contains a *JavaScript Calendar application* [http://developer.iplanet.com/viewsource/husted_calendar/husted_calendar.html] written by Robert W. Husted (**rhusted@requisite.com**) with friendly permission of author. The calendar is now developed as an open source project *'jscal'* [http://jscal.sourceforge.net/]

The calendar can be used to edit values of date fields using a small calendar popup window.

18.2. How to use the calendar within dbforms

To use it, application developers have to do following steps:

i. Copy the directory *dbformslib* with all subdirectories into the root of the web application. This directory can be found within directory *dist* inside DbForms distribution.

ii. See Javadoc for class *java.text.SimpleDateFormat* to find more information about supported formats and date patterns. Be aware that the pattern is case sensitive and that lowercase 'm' does not mean 'month' but 'minute'. The format handling inside the JavaScript calendar is separated from the handling inside dbforms Java code. And unfortunately the meaning of subpatterns is somtimes different in both worlds. E.g. 'mm' means 'minute' in Java patterns and '1 or 2 digit month' for the calendar. For the beginning it is recommended to build a pattern out of the following subpatterns:

- 'dd': two digit day of month
- 'MM': two digit month of year (uppercase!)
- 'yyyy': 4 digit year

As separator you may use '.','-','_' or '/'. Examples of valid patterns are 'yyyy-MM-dd', 'dd.MM.yyyy', 'MM/dd/yyyy'. Such patterns should be understood and handled by the calendar and the dbforms Java code in the same way. See next subsection about how to add support for more date formats.

iii. Include the JavaScript file *calendar.js* containing the calendar application in your JSPs:

```
<script language="javascript" src="dbformslib/jscal/calendar.js\"> </script>
```

If you have JSPs within subdirectories, you can use an absolute path with your context path included in pathname:

```
<script language="javascript" src="/contextpath/dbformslib/jscal/calendar.js
```

If you want to avoid to write context path into your pages and want pages still to work also within subdirectories, you may write instead the following version, which looks a bit more complicated, but will work whereever you copy your pages:

```
<script language="javascript" src=<%= "\"" +request.getContextPath() + "/dbf
```

iv. To activate the calendar for a date field, all you have to do is to set the attribute **useJsCalendar** to `"true"`:

```
<db:dateField fieldName="birthdate" useJsCalendar="true" />
```

When running the application, a small calendar icon will be shown after the date field:

If user clicks on the icon (and has JavaScript activated), a popup window appears showing a calendar. User can navigate to the desired month and then simply click on the date. That selected date will be written into the form field.

18.3. Support for more date formats

If you need to use a date format that can not be expressed by the subpatterns mentioned above ('dd','MM','yyyy'), you have to some more work:

i. Construct a date pattern for your desired format using the possibilities documented in class java.text.SimpleDateFormat. Set this format as global date format inside dbforms config file.

ii. Construct a date pattern for the same format using the calendar's date patterns. Have a look at the comments inside file *calendar.js* for details. This step can be neccessary, because some subpatterns have different meanings in both worlds. The new pattern for the calendar can be set by using the attribute *jsCalendarDatePattern* inside tag dateField:

Here is an example:

Inside dbforms-config.xml you have:

```
<date-format>yyyy-M-d</date-format>
```

Inside the JSP you set the corresponding calendar format:

```
<
db:dateField
fieldName="birthdate" jsCalendarDateFormat="yyyy-mm-dd"
```

Another example:

Inside dbforms-config.xml you have:

```
<date-format>MMM dd',' yyyy</date-format>
```

Then inside the JSP you set:

```
<
db:dateField
fieldName="birthdate" jsCalendarDateFormat="Mon DD, yyy
```

iii. The calendar has a build in parsing routine that is able to parse the simple patterns mentioned above. If the date pattern is different, it just uses the build in constructor for a JavaScript date object. If your date pattern is not understood by the calendar, you may experience that the calendar always starts up with the current date, not with with date that was set in the form field. It might then be neccessary to add some custom parsing code into the file calendar.js.

Normally the JavaScript calendar trys to use the locale of the used browser for names of weekdays and months. But if that might lead to problems when you use patterns including textual representations of dates like in 'May 08, 2002'. A calendar running e.g. on a browser in Germany would set this date as 'Mai 08, 2002' and you would get an error when trying to save this value. That's why the calendar automatically will use English names if the date pattern is not a simple one, so that the result date given back to dbforms hopefully will be correctly understood and parsed. If the default locale of your server is not an English one, it would currently be the best to avoid date patterns containing textual representations of months.

Chapter 19. Foreign Key support within DbForms

19.1. Introduction

DbForms contains limited support for foreign keys within database schemas. DbForms does currently not check foreign key constraints or start cascading actions like updating foreign keys within a referencing table whenever the key of a referenced row is updated. But:

- There is support to express foreign keys within dbforms config file.
- Foreign key information is available within running web application to be used by servlets or JSPs.
- Devgui tries to include foreign key information automatically within dbforms config file by using JDBC metadata methods
- Some of the stylesheets used by devgui to automatically generate JSPs use foreign key information from config file. So they are able to generate select tags etc. to allow the user to choose from rows within another table.

19.2. Foreign-Key tag within dbforms-config.xml

Foreign key infomation within config file has to be expressed using the foreign-key tag that has to be nested within the table tag belonging to the referencing tables.

19.2.1. Here is an excerpt from the DTD:

```
<!ELEMENT foreign-key (reference+)>

<!ATTLIST foreign-key
    foreignTable    CDATA #REQUIRED
    name            CDATA #REQUIRED
    visibleFields   CDATA #IMPLIED
    format          CDATA #IMPLIED
    displayType (select|radio|none) "none"
>


<!ELEMENT reference EMPTY>

<!ATTLIST reference
    local           CDATA #REQUIRED
    foreign         CDATA #REQUIRED
>
```

19.2.2. Let's say you have created a database containing tables:

```
create table
customer
    (cno integer not null primary key,
    first_name char(20) not null,
    last_name  char(20) not null
    phone      char(20) not null,
    ...
    )

create table
order
    (orderno integer not null primary key,
    cust_no integer not null,
    order_date date not null,
    ...
    constraint customer_known_in_order
        foreign key (cust_no)
        references customer(cno)
    )
```

19.2.3. Then within dbforms-config.xml you could write:

```
<table name="customer">
    <field name="cno" fieldType="integer" isKey="true"/>
    <field name="first_name" fieldType="char" size="20"/>
    <field name="last_name" fieldType="char" size="20"/>
    <field name="phone" fieldType="char" size="20"/>
    ...
<table>

<table name="order">
    <field name="cno" fieldType="integer" isKey="true"/>
    <field name="cust_no" fieldType="integer"/>
    <field name="order_date" fieldType="date"/>
    ...


    <foreign-key foreignTable="customer"

              name="customer_in_order"

              displayType="select"

              visibleFields="first_name,last_name,phone"

              format="%s %s (Phone %s)" >


    <reference local="cust_no" foreign="cno"/>

    </foreign-key>
</table>
```

- the foreign-key has to be nested within the referencing table
- attributes:
    - foreignTable: name of referenced table (required)
    - name: name of this constraint (required, used for a faster look-up)
    - displayType: This is a suggestion how to present this reference within an automatically generated page
        - select: create a select tag
        - radio : create a set of radio fields
        - none : ignore reference, do nothing (default if not set)
    - visibleFields: which columns of the referenced table shall be presented to the user? This gets directly passed to corresponding attribute within tableData tag, see docs there
    - format: which format pattern shall be used to format the visibleFields? See docs for tag tableData for details.
- nested tags reference: models pairs of columns that take part in the reference:
    - local: name of the referencing column
    - foreign: name of the referenced column

      If a key consists of more than one column, there will be more than one nested tag, e.g. if the customer is just unique within an area, we'd have:

```
<foreign-key foreignTable="customer" ...>
  <reference local="area_id" foreign="area_id"/>
  <reference local="cust_no" foreign="cno"/>
 </foreign-key>
```

19.3. Support within XSL Stylesheets

This has been assigned, but it is not yet implemented (as of 1.1.2pr2).

Some of the stylesheets used by devgui use the information about foreign keys to create automatically tags allowing to select from a referenced table. In the above example, we might get a:

```
<db:select fieldName="cust_no">
<db:tableData
    name="customer_in_order"
    foreignTable="customer"
```

```
    visibleFields="first_name,last_name,phone"
    format="%s %s (Phone %s)"
    storeField="cno"

/>
</db:select>
```

19.4. Simplified Reference with New Attributes for table tag:

If a table is referenced from lots of other tables, it might be neccessary to repeat the attribute values for visibleFields and format within each foreign-key tag again and again. To avoid this, two new attributes for the table tag have been introduced:

- **defaultVisibleFields** sets the default for attribute visibleFields whenever referencing to this table
- **defaultVisibleFieldsFormat** does the same for format

19.4.1. Example>

So the previous example of the **dbforms-config.xml** could have been rewritten as:

```
<table name="customer"

       defaultVisibleFields="first_name,last_name,phone"

       defaultVisibleFieldsFormat="%s %s (Phone %s)">

     <field name="cno" fieldType="integer" isKey="true"/>
     <field name="first_name" fieldType="char" size="20"/>
     <field name="last_name" fieldType="char" size="20"/>
     <field name="phone" fieldType="char" size="20"/>
   ...

<table>

<table name="order">
   <field name="cno" fieldType="integer" isKey="true"/>
   <field name="cust_no" fieldType="integer"/>
   <field name="order_date" fieldType="date"/>
   ...

   <foreign-key foreignTable="customer"
      name="customer_in_order"
      displayType="select"  >

   <reference local="cust_no" foreign="cno"/>

   </foreign-key>
</table>
```

19.5. Detection of references within DevGui

DevGui uses JDBC metadata methods to detect foreign keys and will automatically insert corresponding foreign-key tags into generated dbforms-config.xml. This will surely only work if the JDBC driver has support for this feature, otherwise the file has to be edited manually. DevGui sets several default values, which might be edited before applying xsl stylesheets:

- if a key contains just one column, the `displayType` is set to "select", if it contains more, `displayType` is set to "none". This is a problem of the underlying tags that will later be used within web application.
- devgui does not set the `visibleFields` attribute within foreign-key tags, instead it sets the `defaultVisibleFields` attribute within each table. The attribute will initially be set to the primary key.

The `displayType` or `defaultVisibleFields` attribute might be changed before applying the xsl stylesheets. It might also be a good idea to set a corresponding `defaultVisibleFieldsFormat`.

Chapter 20. Connection Support

20.1. ConnectionFactory

The ConnectionFactory class provides JDBC connection objects to the DbForms client (specifically, to the org.dbforms.util.DbConnection class). It gives a common interface to retrieve connection objects, but defer the database interaction to specialized *connectionProvider* classes.

The `connectionFactory` use is optional and must be enabled to work, so it will not be used if the new properties are not declared as shown below.

`dbconnection` configuration is still compatible with previous means such as:

```
<dbconnection
   name = "java:comp/env/jdbc/resource"
   isJndi = "true" />
```

where `jdbc/resource` should match a resource defined in the servlet container and

```
<dbconnection
   name     = "jdbc:mysql://localhost/test"
   isJndi   = "false"
   conClass = "org.gjt.mm.mysql.Driver"
   username = "somename"
   password = "somepass" />
```

20.1.1. Included ConnectionProvider classes

The connectionFactory package provides three *connectionProvider* classes; the jars for these classes need to be downloaded and put in your applications WEB-INF/lib directory.

Protomatter rquires `protomatter.jar`

JakartaConnectionPool requires `commons-dbcp.jar`

- **ProtomatterConnectionProvider**

  Provides connection objects using the Protomatter Connection Pool library

  http://protomatter.sourceforge.net/

  `class name: org.dbforms.conprovider.ProtomatterConnectionProvider`

- **JakartaConnectionProvider**

  Provides connection objects using the Jakarta commons-dbcp ConnectionPool library

http://jakarta.apache.org/commons/index.html

class name: `org.dbforms.conprovider.JakartaConnectionProvider`

- **SimpleConnectionProvider**

  Provides connection objects, but does not use connection pool components.

  class name: `org.dbforms.conprovider.SimpleConnectionProvider`

20.1.2. DbForms and ConnectionFactory configuration

The examples below show how to configure the "dbconnection" xml element contained in the dbforms-config.xml file.

20.1.2.1. ProtomatterConnectionProvider

```
<!--
  -  uses the Protomatter ConnectionPool
  -
  -  see http://protomatter.sourceforge.net/ for further information
  -->
<dbconnection
  id                    = "protomatter"
  isJndi                = "false"
  isPow2                = "true"
  default               = "true"
  connectionProviderClass = "org.dbforms.conprovider.ProtomatterConnection
  connectionPoolURL     = "jdbc:protomatter:pool:postgresPool"
  conClass              = "org.postgresql.Driver"
  name                  = "jdbc:postgresql://localhost/myDatabase?charSe
  username              = "postgres"
  password              = ""/>
```

20.1.2.2. JakartaConnectionProvider

```
<!-
  -  uses the Jakarta commons-dbcp ConnectionPool
  -
  -  see http://jakarta.apache.org/commons/index.html for further informat
  -
  -  Note: this provider does NOT use the connectionPoolURL attribute.
  -->
<dbconnection
  id                    = "jakarta-commons-dbcp"
  isJndi                = "false"
  isPow2                = "true"
  default               = "false"
  connectionProviderClass = "org.dbforms.conprovider.JakartaConnectionProv
  connectionPoolURL     = ""
  conClass              = "org.postgresql.Driver"
  name                  = "jdbc:postgresql://localhost/myDatabase?charSe
```

```
  username                = "postgres"
  password                = ""/>
```

20.1.2.3. SimpleConnectionProvider

```
<!-
  -  DOES NOT use connection pooling !!!
  -
  -  Note: this provider does NOT use the connectionPoolURL attribute.
  -->
<dbconnection
  id                      = "simple"
  isJndi                  = "false"
  isPow2                  = "true"
  connectionProviderClass = "org.dbforms.conprovider.SimpleConnectionProvi
  connectionPoolURL       = ""
  conClass                = "org.postgresql.Driver"
  name                    = "jdbc:postgresql://localhost/myDatabase?charSe
  username                = "postgres"
  password                = ""/>
```

20.1.3. configuration parameters

id	string identifier for the dbconnection element. Gives the dbconnection an alpha-numeric name. Otherwise it is referred to by the order it was configured with the first 'dbconnection' starting at zero.
default	boolean value, specify that the current 'dbconnection'element specifies the default connection. The default connection is used by all the data access tags that don't specify a 'dbConnectionName' attribute value.
isJndi	boolean value, specify that the connection objects are retrieved using a JNDI service.
isPow2	boolean value, specify that the connectionFactory class will be used to objain jdbc connection objects [better to use #useConnectionFactory#, don't you think so ?].
connectionProviderClass	the full qualified class name of the connectionProvider class.
connectionPoolURL	the connection pool url string. Used by ProtomatterConnectionProvider class.
conClass	the JDBC driver connection class string
name	the JDBC URL string
username	database user name value
password	database password value

20.1.4. Setting JDBC Properties

To set jdbc properties in jdbc drivers that can't specify properties using the JDBC url, you can nest property tags inside the dbconnection tag:

```
<dbconnection>
  <property name="charSet" value="8859_1" />
</dbconnection>
```

This properties will result in a DriverManager.getConnection(url, properties) call. It is not necessary to include the username and the password in the property list, this will be handled by the connection class.

Currently properties are supported for native connections (isJNDI=false, isPow2=false) and for connection with the Connection factory (isPow2=true). **Example:**

```
<dbconnection
  id                    = "simple"
  isPow2                = "true"
  connectionProviderClass = "org.dbforms.conprovider.SimpleConnectionProvi
  connectionPoolURL     = ""
  name                  = "jdbc:mysql://localhost/test"
  isJndi                = "false"
  conClass              = "org.gjt.mm.mysql.Driver"
  username              = "secret"
  password              = "howdidyoufindout">

  <property name="useUnicode"        value="true" />
  <property name="characterEncoding" value="Shift_JIS" />
</dbconnection>
```

20.2. Connection pool properties

Every connectionProvider class that uses a connection pool facility can manage its own connection pool properties.

The dbconnection element can configure those properties using "pool-property" child elements.

20.2.1. Protomatter ConnectionProvider supported properties

Protomatter ConnectionProvider class uses the following properties to configure its connection pool:

pool.initialSize:

The initial pool size (default is 0).

pool.maxSize:

The max pool size (default is -1). If the max pool size is -1, the pool grows infinitely.

pool.growBlock:

The grow size (default is 1). When a new object is needed, this many are created.

pool.createWaitTime:

The time (in ms) to sleep between pool object creates (default is 0).

This is useful for database connection pools where it's possible to overload the database by trying to make too many connections too quickly.

jdbc.validityCheckStatement:

A SQL statement that is guaranteed to return at least 1 row. For Oracle, this is "select 1 from dual" and for Sybase it is "select 1".

This statement is used as a means of checking that a connection is indeed working.

pool.maxConnectionIdleTime:

If this property is present, and the pool.maidThreadCheckInterval property is also present, then a thread will be created that looks for connections that have been idle for more than pool.maxConnectionIdleTime seconds. When this thread finds them, it closed the connection and logs a warning with a stack trace of when the connection was checked out of the pool.

This is primarily here as a debugging aid for finding places where connections are not getting close, and should not be used in a production environment.

pool.maidThreadCheckInterval:

This is the number of seconds between attempts by the maid thread (if present) to find idle connections.

For further details: see http://protomatter.sourceforge.net/1.1.8/javadoc/com/protomatter/jdbc/pool/JdbcConnectionPool.html

20.2.2. Protomatter ConnectionProvider configuration example

```
<dbconnection
  id                      = "protomatter"
  isJndi                  = "false"
  isPow2                   = "true"
  defaultConnection       = "true"
  connectionProviderClass = "org.dbforms.conprovider.ProtomatterConnection
  connectionPoolURL       = "jdbc:protomatter:pool:postgresPool"
  conClass                = "org.postgresql.Driver"
  name                    = "jdbc:postgresql://myHost/myDb"
```

```
    username               = "postgres"
    password               = "">

    <!-- jdbc properties -->
    <property name="charSet" value="ISO-8859-1" />

    <!-- Connection pool dataSource properties -->
    <pool-property name="pool.initialSize"             value="4"   />
    <pool-property name="pool.maxSize"                 value="10"  />
    <pool-property name="pool.growBlock"               value="2"   />
    <pool-property name="pool.createWaitTime"          value="100" />
    <pool-property name="pool.maxConnectionIdleTime"   value=""    />
    <pool-property name="pool.maidThreadCheckInterval" value=""    />
    <pool-property name="jdbc.validityCheckStatement"  value=""    />

  </dbconnection>
```

20.2.3. Jakarta ConnectionProvider supported properties

Jakarta ConnectionProvider class uses the following properties to configure its connection pool:

validationQuery:

> The SQL query that will be used to validate connections from this pool before returning them to the caller. If specified, this query MUST be an SQL SELECT statement that returns at least one row.

maxActive:

> The maximum number of active connections that can be allocated from this pool at the same time, or zero for no limit.

maxIdle:

> The maximum number of active connections that can remain idle in the pool, without extra ones being released, or zero for no limit.

maxWait:

> The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely.

For further details: http://jakarta.apache.org/commons/dbcp/api/org/apache/commons/dbcp/BasicDataSource.html

useLog: "true" | "false"

> if "true", force the dataSource to log its statements using log4j.
>
> *Note: Developer added this one to activate the dataSource log.*

20.2.4. Jakarta ConnectionProvider configuration example

```
<dbconnection
  id                      = "jakarta-commons-dbcp"
  isJndi                  = "false"
  isPow2                  = "true"
  default                 = "false"
  connectionProviderClass = "org.dbforms.conprovider.JakartaConnectionProv
  connectionPoolURL       = ""
  conClass                = "org.postgresql.Driver"
  name                    = "jdbc:postgresql://localhost/myDatabase"
  username                = "postgres"
  password                = "">

  <!-- jdbc properties -->
  <property name="charSet" value="ISO-8859-1" />

  <!-- connection pool dataSource properties -->
  <pool-property name="validationQuery" value=""     />
  <pool-property name="maxActive"       value="10"   />
  <pool-property name="maxIdle"         value="5"    />
  <pool-property name="maxWait"         value="-1"   />
  <pool-property name="useLog"          value="true" />

</dbconnection>
```

20.3. The ConnectionProvider class

A specialized *connectionProvider* class is a common class that extends the *ConnectionProvider* abstract class and provides the implementation of the following abstract methods:

- `protected abstract void init() throws Exception;`
- `protected abstract Connection getConnection() throws SQLException;`

A connectionProvider can implement the code to interact with a connection pool system.

20.3.1. The init method

The *init* method should be used to initialize the chosen data source object or the connection pool component.

Examples:

- loading a specified DriverManager class
- initialize a java DataSource object
- initialize a connection pool object

Here's what the JakartaConnectionProvider class does:

```
/**
 *  Initialize the Jakarta Commons connection pool.
 *
 * @throws Exception if any error occurs
 */
protected void init() throws Exception
{
  dataSource = new BasicDataSource();
  dataSource.setDriverClassName(prefs.getJdbcDriver());
  dataSource.setUrl(prefs.getJdbcURL());
  dataSource.setUsername(prefs.getUser());
  dataSource.setPassword(prefs.getPassword());
  dataSource.setValidationQuery(null);
  dataSource.setMaxActive(20);
  dataSource.setMaxIdle(5);
  dataSource.setMaxWait(-1);
}
```

20.3.2. The getConnection method

Simply, it must return an initialized Connection object.

Example of the JakartaConnectionProvider class:

```
/**
 *  Get a JDBC Connection
 *
 * @return  a JDBC Connection
 *
 * @exception  SQLException Description of the Exception
 */
protected Connection getConnection() throws SQLException
{
  return dataSource.getConnection();
}
```

20.4. How To code your own connection provider class

To create your own connection provider class, follow these steps:

i. create a new class that extends " `org.dbforms.conprovider`.ConnectionProvider" class:

```
public class MyOwnConnectionProvider extends ConnectionProvider
{
  ...
}
```

ii. create a default constructor like this one:

```
/**
 *  Default constructor.
 *
 * @throws Exception because of the <code>throws Exception</code> clause
 *                   of the  <code>init</code> method.
 */
public MyConnectionProvider() throws Exception
{
   super();    // up to now, do nothing...
}
```

iii. implement the following ConnectionProvider abstract methods:

- `protected abstract void init() throws Exception;`
- `protected abstract Connection getConnection() throws SQLException;`

20.4.1. Using pool-property elements to configure the JDBC connection pool attributes

A connectionProvider class can configure its connection pool attributes sing the values specificed by the pool-property xml elements, children of the dbconnection element of the dbforms-config.xml file.

Here's an example of the configuration of the dbconnection element that uses the ProtomatterConnectionProvider class to obtain JDBC connections from the database:

```
<dbconnection
  id                     = "protomatter"
  isJndi                 = "false"
  isPow2                 = "true"
  defaultConnection      = "true"
  connectionProviderClass = "org.dbforms.conprovider.ProtomatterConnection
  connectionPoolURL      = "jdbc:protomatter:pool:postgresPool"
  conClass               = "org.postgresql.Driver"
  name                   = "jdbc:postgresql://myHost/myDb"
  username               = "secret"
  password               = "moreSecret">

  <!-- jdbc properties -->
  <property name="charSet" value="ISO-8859-1" />

  <!-- Connection pool dataSource properties -->
  <pool-property name="pool.initialSize"             value="4"   />
  <pool-property name="pool.maxSize"                 value="10"  />
  <pool-property name="pool.growBlock"               value="2"   />
  <pool-property name="pool.createWaitTime"          value="100" />
  <pool-property name="pool.maxConnectionIdleTime"   value=""    />
```

```
  <pool-property name="pool.maidThreadCheckInterval" value=""    />
  <pool-property name="jdbc.validityCheckStatement"  value=""    />
</dbconnection>
```

When a ConnectionProvider class is instantiated, it can access to the connection pool properties using the following statement:

```
Properties props = prefs.getPoolProperties();
```

The "name" attribute of the pool-property xml element becomes the property key that must be used to retrieve the property value from the above Properties class.

Every connection pool package uses its own configuration system; here's the example code used to configure the *ProtomatterConnectionProvider* class:

```
Properties props = null;
...

// now set the connection pool custom properties;
// if the connectionPool properties object is null,
// instance a new properties object anyway, to use default values;
if ((props = prefs.getPoolProperties()) == null)
  props = new Properties();

// use defaults values as specified into the documentation;
setIntegerArg(args, props, CP_PROPS_INITIALSIZE,             "0");
setIntegerArg(args, props, CP_PROPS_MAXSIZE,                 "-1");
setIntegerArg(args, props, CP_PROPS_GROWBLOCK,               "1");
setIntegerArg(args, props, CP_PROPS_CREATEWAITTIME,          "0");
setArg       (args, props, CP_PROPS_VALIDITYCHECKSTATEMENT,  null);
setIntegerArg(args, props, CP_PROPS_MAXCONNECTIONIDLETIME,   null);
setIntegerArg(args, props, CP_PROPS_MAIDTHREADCHECKINTERVAL, null);

// finally create the pool and we're ready to go!
Class.forName(CP_DRIVER).newInstance();
connectionPool = new JdbcConnectionPool(getLastToken(prefs.getConnectionPo
```

and here's the (simpler) code for J *akartaConnectionProvider* class:

```
Properties props = null;
...

// now set the connection pool custom properties;
// if the connectionPool properties object is null,
// instance a new properties object anyway, to use default values;
if ((props = prefs.getPoolProperties()) == null)
  props = new Properties();

String validationQuery = props.getProperty(CP_PROPS_VALIDATION_QUERY, null

if (!Util.isNull(validationQuery))
```

```
  dataSource.setValidationQuery(validationQuery.trim());

dataSource.setMaxActive (Integer.parseInt(props.getProperty(CP_PROPS_MAX_A
dataSource.setMaxIdle   (Integer.parseInt(props.getProperty(CP_PROPS_MAX_I
dataSource.setMaxWait   (Long.parseLong  (props.getProperty(CP_PROPS_MAX_W

// if PROPS_LOG == true, use log4j category to log the datasource info;
String useLog = props.getProperty(CP_PROPS_USE_LOG, "false");

if (!Util.isNull(useLog)  "true".equals(useLog.trim()))
{
  cat.info("::init - dataSource log activated");
  dataSource.setLogWriter(new Log4jPrintWriter(cat, cat.getPriority()));
}
```

Chapter 21. Multiple Database Connections

This is still being tested and debugged but to use this feature:

I. specify multiple <dbconnection> elements in the dbforms-config.xml

It is allowed to have at most 1 connection without an id within a config file. That will be the default connection.

II. Specify the dbConnectionName to be the id of the dbconnection you wish to use in relevant tags including: **dbform, tableData, queryData, getConnection, blobContent**

```
    <db:select fieldName="tc">
    <db:queryData name ="combinations" query="SELECT DISTINCT tc, tier, cou
disableCache="true"
    dbConnectionName="simple"/>

    </db:select>
```

would be used with a connection such as the following:

```
    <dbconnection
    id = "simple"
<<< this will be used for the connectionName (your arbitrary choice)
isJndi = "false"
isPow2 = "true"
connectionProviderClass = "org.dbforms.conprovider.SimpleConnectionProvider
connectionPoolURL = ""
conClass = "org.postgresql.Driver"
name ="jdbc:postgresql://localhost/myDatabase?charSet=ISO-8859-1"
username = "postgres"
password = ""
    />
```

Chapter 22. Pluggable events

22.1. Introduction

DbForms uses a declarative event configuration system that provides the following features:

- event classes registration
- event classes override on a table-by-table basis
- event classes configuration on a table-by-table basis

22.2. Default events

Dbforms supports and uses by default the following event types and the related event classes:

22.2.1. Database Events

Event Type	Event Class
insert	org.dbforms.event.InsertEvent
update	org.dbforms.event.UpdateEvent
delete	org.dbforms.event.DeleteEvent

22.2.2. Navigation events

Event Type	Event Class
navFirst	org.dbforms.event.NavFirstEvent
navPrev	org.dbforms.event.BoundedNavPrevEventImpl
navNext	org.dbforms.event.BoundedNavNextEventImpl
navLast	org.dbforms.event.NavLastEvent
navNew	org.dbforms.event.NavNewEvent
navGoto	org.dbforms.event.GotoEvent

22.3. Registration of the event classes

Database and Navigation event factories automatically register the default event classes into the system. Developers can register additional event classes by declaring and configuring elements into the dbforms-config file, as shown in the example below:

```
<dbforms-config>
```

```
    <!-- dbforms config stuff here... -->
    <!-- define custom events -->

    <events>

      <!-- define database custom events here -->
      <database-events>❶
        <database-event id="insert2" type="insert" className="org.foo.bar.MyInsert
        <database-event id="update2" type="update" className="org.foo.bar.MyUpdate
      </database-events>

      <!-- define navigation custom events here -->
      <navigation-events>❷
        <navigation-event id="new2" type="navNew"   className="org.foo.bar.MyNavNe
      </navigation-events>

    </events>
  </dbforms-config>
```

❶ To add database custom events, a developer must:

- add a new `/dbforms-config/events/database-events` element, if it does not exist yet
- add a new `/dbforms-config/events/database-events/database-event` element for every database custom event to register into the system

❷ To add navigation custom events, a developer must:

- add a new `/dbforms-config/events/navigation-events` element, if it does not exist yet
- add a new `/dbforms-config/events/navigation-events/navigation-event` element for every navigation custom event to register into the system

22.3.1. Event attributes

Every database-event element and navigation-event element uses the following mandatory attributes:

attribute name	description
id	the event identifier. Must be unique.
type	the event type. Must be a supported type value.
className	the full qualified class name of the event class.

22.4. Override the default event classes

The declarative event configuration system let developers change the default event classes with custom ones. In this manner, every table element uses those event classes instead of the default ones.

22.4.1. global overriding

To override a default event class in a global way, it is sufficient to:

- declare a new `database-event` or `navigation-event` element
- set the value of its type attribute with a suppored event type value
- set the value of its className attribute with the full qualified class name of your custom event class

Example:

```
<events>
  <database-events>
    <database-event type="update" className="org.foo.bar.MyUpdateEvent"/>
  </database-events>
</events>
```

This configuration overrides the default update event class with the custom org.foo.bar.MyUpdateEvent one.

All the table elements will use the custom org.foo.bar.MyUpdateEvent class instead of the default one (org.dbforms.event.UpdateEvent).

22.4.2. Log information about event override

DbForms logs any information regarding the event class override procedure.

Here's an example log statement that shows that the custom "org.foo.bar.MyUpdateEvent" update event class overrides the default "org.dbforms.event.UpdateEvent" one.

```
::addEventInfo - the event information having id, class [update, org.foo.bar.MyU
                 overrides the event class [org.dbforms.event.UpdateEvent]
```

22.5. Override the event classes on a table-by-table basis

Every table object specified by a `/dbforms-config/table` element uses by default the standard event configuration. A developer can choose to override the event class associated to a certain event type, by adding and configuring the events element in the dbforms-config file as shown below:

```
<dbforms-config>
 ...
 <table name="MYTABLE">
```

```
    <field name="TABLEID"  fieldType="varchar" size="10" isKey="true"/>
    <field name="NAME"     fieldType="varchar" size="16"/>
    ...

    <interceptor className="foo.bar.MyInterceptor"/>

    <!--
      -  Here you can specify custom event classes that override the default one
      -  Event properties can be configured using "property" elements.
      -->
    <events>❶
      <event type="insert" id="insert2">

        <!-- event properties configuration -->
        <property name="sourceView"  value="VIEW_USER" />
        <property name="targetTable" value="USER" />
      </event>
    </events>

  </table>

  ...

</dbforms-config>
```

❶ To override the event classes associated to the event types, a developer must:

- add a new `/dbforms-config/table/events` element, if it does not exist yet
- add a new `/dbforms-config/table/events/event` element for every event class to override

22.5.1. Event attributes

Every `/dbforms-config/table/events/event` element uses the following mandatory attributes:

attribute name	description
type	the event type. Must be a supported type value.
id	the event identifier. Must be equal to the identifier attribute of an event specified by a `/dbforms-config/events/database-events/database-event` or `/dbforms-config/events/navigation-events/navigation-event` element.

22.5.2. Event properties configuration

Every `/dbforms-config/table/events/event` can be configured using a set of properties (obviously, those properties are useful only if the event class support them). To declare the properties, the developer must add to the element a set of children elements and configure their attributes.

22.5.3. Properties attributes

Every `/dbforms-config/table/events/event` element uses the following mandatory attributes:

attribute name	description
name	the name of the event property.
value	the value of the event property.

22.5.4. Example

The following example shows how to override an update event in an alias of a table.

```
<dbforms-config>

        <table name="explanations" ...>
              ...
        </table>

        <query name="explanations_with_custom_update" from="explanations">
                    <interceptor className="your.package.dbi.interceptor_class_nam
                <events>
                        <event type="update" id="update2"/>
                </events>
        </query>



        <events>
                   <!-- define database custom events here -->
                <database-events>
                        <database-event id="update2" type="update" className="your
                </database-events>

        </events>

</dbforms-config>
```

Chapter 23. EJBs and JBoss

23.1. The Easy Way

John Gagon who consulted developers on how to alter the code to use EJBs with DbForms finally reported that:

I have Session EJBs working with DBForms....I didn't change any source code to DBForms either.

What I did was within an interceptor. Lets say we have:

```
preInsert(HttpServletRequest request, DBFormsConfig config, Hashtable fieldValues,
{

        //request and config stuff can go here.

        //jndi lookup...etc.
        //(this could go in an abstract class that sandwiches between the dbforms
        //and your own)
        try
        {

        fieldValues = beanInterface.doFunction(fieldValues, con);
           //pass fieldValues for business logic.
           //shouldn't know about dbforms or request really. we want the beans to
           //the bean can have a flag that tells it whether to persist
           // or return persisance to the caller.
           //some utility beans can also convert the raw fieldValues to Beans
           //and you can get dbforms apps to integrate with EJB apps...(kewl)

        }catch
        {
           //based on what is caught, you can ... rethrow ValidationExceptions, re
           //throw blank ValidationExceptions(), log stuff, or simply decide to re
           //catch transactional exceptions, you could "recheck".
        }


}
```

In this way, the interceptors are thin adapters that can do little extra dbforms and web based work without doing a lot of business logic (like I used to do). The session bean does business work on the fieldValues and can work with other servers, track certain rules, calculate.

I think you can even do some transactional locking ie: between an inserts/selects and updates by throwing these exceptions or waiting "while(not_finished)"...of course, that can reduce performance. You could also session flag so that other interceptor methods cannot do that. (or you can put transactions in your database functions if your database supports it. (you could use the stateful session bean and declare a level of transactional thread safety on those session beans I think too)

23.2. Example

This example was tested on JBoss 3.0.0-Tomcat 4.0.6

- In application.xml, make sure you have

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

- Make sure you have the jar you compiled
- do not play with jndi.properties
- don't need jboss-web.xml

23.2.1. Example interceptor

```
public class BasicTestInterceptor extends DbEventInterceptorSupport
{

 static Category log = Category.getInstance(BasicTestInterceptor.class.getName())

 /**
  * Checks to see if a number is in range inclusively given three Strings.
  * @param check String representing a number.
  * @param low String representing the low number of the range.
  * @param high String representing the high number of the range
  * @param inclusive set to true if you want the low <= check and check <= high .
  * @param digits the number in digits to the left of the decimal to round.
       *  positive, non-zero int
  * @return value converted to a string.
  * @throws ValidationException when encountering other exceptions like
       *  NumberFormatException.
  *
  */
 public static boolean inRange(String check, String low, String high, boolean inc
   throws ValidationException
 {
   try
   {
    NumericValidationLocal numericValidation = EjbUtil.getNumericValdiationBean()
    if(numericValidation==null)
    {
      System.out.println("Null numericValidation");
      return false;
    }
    return numericValidation.inRange(check,low,high,inclusive);
     }
     catch(MathException me)
     {
     throw new ValidationException(me.getMessage());
   }
   catch(Exception e)
   {
     throw new ValidationException("");
   }

    }//inRange(...)

 ...
```

```
}//BasicTestInterceptor
```

23.2.1.1. NumericValidationLocal

```
package com.atser.sejb.math;


public interface NumericValidationLocal
 extends javax.ejb.EJBLocalObject
{

 /**
  * Checks to see if a number is in range.
  * Range is inclusive given three Strings.
  * @param check String representing a number.
  * @param low String representing the low number of the range.
  * @param high String representing the high number of the range
  * @param inclusive set to true
  *  if you want the low <= check and check <= high .
  * @param digits the number in digits to the
  *  left of the decimal to round. positive, non-zero integer only.
        *  For example: 1438 with 1 passed in will give us 1440.
  * @return value converted to a string.
  * @throws ValidationException for any number format problems.
  */
 public boolean inRange(
     String check,
     String low,
     String high,
     boolean inclusive
     )
   throws MathException;


}//NumericValidationLocal
```

23.2.1.2. NumericValidationLocalHome

```
package com.atser.sejb.math;


public interface NumericValidationLocalHome
 extends javax.ejb.EJBLocalHome
{

 NumericValidationLocal create() throws javax.ejb.CreateException;

}//NumericValidationLocal
```

23.2.1.3. NumericValidationStatelessBean

```
import javax.ejb.*;//SessionContext
```

```
import org.apache.log4j.*;//Category


public class NumericValidationStatelessBean
 implements javax.ejb.SessionBean
{

 private static Category log =
     Category.getInstance("NumericValidationStatefulBean");


 private javax.ejb.SessionContext ctx;

 /**
  * Provides generic range checking.
  * Checks to see if a number is in range.
  * Range is inclusive given three Strings.
  * and returns true if in range or false if out of
  * specified range.
  */
 public static boolean inRange(String check,
     String low, String high, boolean inclusive)
   throws MathException
 {
   try
   {
     double c = Double.parseDouble(check);
     double l = Double.parseDouble(low);
     double h = Double.parseDouble(high);
     if(inclusive)
       return (l<=c && c<=h);
     else
       return (l<c && c<h);
   }
   catch(NumberFormatException nfe)
   {
     throw new MathException("Invalid arguements to method:
                             inRange(String check, String low, String high,
                             boolean inclusive):"+check+","+low+"
                             ,"+high+","+inclusive);
   }


   }//inRange(...)


 public void ejbCreate()
 {
   log.debug("ejbCreate()");
 }//ejbCreate()


 public void ejbRemove()
 {
   log.debug("ejbRemove()");
 }//ejbRemove()


 public void ejbActivate()
 {
   log.debug("ejbActivate()");
 }//ejbActivate()
```

```
  public void ejbPassivate()
  {
    log.debug("ejbPassivate()");
  }//ejbPassivate()


  public void setSessionContext(javax.ejb.SessionContext ctx)
  {
    this.ctx = ctx;
    log.debug("setSessionContext(..)");
    return;
  }//setSessionContext(..)


 }//NumericValidationStatefulBean
```

23.2.2. jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
   <enterprise-beans>
     <session>
       <ejb-name>com/atser/sejb/math/NumericValidation</ejb-name>
       <jndi-name>com/atser/sejb/math/NumericValidation</jndi-name>
     </session>
   </enterprise-beans>
 </jboss>
```

23.2.3. ejb-jar.xml

```
 <!DOCTYPE ejb-jar
   PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
   'http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd'>
 <ejb-jar>
   <enterprise-beans>
     <session>
       <!-- The JNDI name of the bean.  -->
       <ejb-name>com/atser/sejb/math/NumericValidation</ejb-name>

       <!-- Class configuration for the bean -->
       <local-home>com.atser.sejb.math.NumericValidationLocalHome</local-home>
       <local>com.atser.sejb.math.NumericValidationLocal</local>
       <ejb-class>com.atser.sejb.math.NumericValidationStatelessBean</ejb-class>
       <session-type>Stateless</session-type>
       <transaction-type>Container</transaction-type>
     </session>
   </enterprise-beans>
 </ejb-jar>
```

23.2.4. web.xml

```
 <ejb-local-ref>
```

```
        <ejb-ref-name>ejb/NumericValidation</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <local-home>com.atser.sejb.math.NumericValidationLocalHome</local-home>
        <local>com.atser.sejb.math.NumericValidationLocal</local>
        <ejb-link>com/atser/sejb/math/NumericValidation</ejb-link>
    </ejb-local-ref>
```

23.3. JBoss dbconnection configuration -JNDI name

For JBoss-3.2.0 the configuration goes as follows:

i. Copy the proper database description file from "<JBoss-dir>/docs/examples/jca" to the deployment directory of your configured JBoss profile (for example "<JBoss-dir>/server/default/deploy") and customize it.

ii. Restart JBoss

iii. After that, the running JNDI 'nodes' can be checked using the JMX-Console (http://localhost:<port>/jmx-console), using the "listXML" operation. In my case, I had an entry saying

```
--- listXML results ---
java:

DB2DS
org.jboss.resource.adapter.jdbc.WrapperDataSource
--- /listXML results ---
```

iv. From this I could easily derive the correct JNDI-name, which I configured as:

```
--- dbforms-config.xml ---
<dbconnection
    id = "1"
    isJndi = "true"
    name = "java:DB2DS"
    default = "true"
    username = "user"
    password = "password"
    connectionProviderClass =
        "org.jboss.resource.connectionmanager.LocalTxConnectionManager"
    conClass  = "COM.ibm.db2.jdbc.app.DB2Driver"/>
-- /dbforms-config.xml ---
```

Chapter 24. JasperReports

Dbforms has included a servlet (StartReportServlet) to enable the use of *JasperReports* [http://jasperreports.sourceforge.net] which is a "report-generating tool that has the ability to deliver rich content onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files".

The StartReportServlet will look for the report xml file in `WEB-INF/custom/reports`, or `WEB-INF/reports`.

If the report xml is newer then the jasper file report will be recompiled.

Example usage:

i. with a simple goto button:

```
<db:gotoButton
    destTable="web_parts"
    destination="/reports/Artikel"
/>
```

ii. for one record:

```
<db:gotoButton
    destTable="web_parts"
    keyToDestPos="currentRow"
    destination="/reports/Artikel"
/>
```

In `WEB-INF/web.xml`, Servlet mapping must be set to handle all /reports by this servlet!!!

```
<servlet>
   <servlet-name>startreport</servlet-name>
   <display-name>startreport</display-name>
    <servlet-class>org.dbforms.servlets.StartReportServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>startreport</servlet-name>
   <url-pattern>/reports/</url-pattern>
</servlet-mapping>
```

Also, JasperReports.jar needs to be in your `WEB-INF/lib` directory.

Chapter 25. Using XML for Data input

25.1. Introduction

With the new event system it is possible to add different (other then JDBC!) datasources to dbforms. To do so you must tell dbforms which dataAccessClass it should use in the config file (see below).

Data retrieving is done by xpath expressions for the data itselfs and for the fields. The new table attribute alias is used to retrieve the data from the system via an url. The query part is used to build an xpath expression to get the data. The url is given by:

```
file:///books.xml?/books/book[@ISBN="42"]
```

This will retrieve the book from the booklist in the data file books.xml with the ISBN 42. For more see the documentation of xpath. Fields will be retrieved by the resulting dataset using the relative xpath expression in the attribute expression of the field.

Data access is read only up to now. Idea's how to build an generic read/write data access are welcomed. Maybe we should integrate support for an xml data base like jindice from apache?

Be sure to add the following to the lib directory in your webapp.

i. xml-apis.jar

ii. xmlParserAPIs.jar

iii. xalan.jar

25.2. xml data file example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<books>
    <book>
      <title>This is a test</title>
      <ISBN>42</ISBN>
      <author>
        <name>Douglas</name>
        <surname>Adman</surname>
      </author>
    </book>
</books>
```

25.3. definig a xml table in dbforms-config.xml

```
  <table
    name="XMLBOOKS"
```

```
 dataAccessClass="org.dbforms.event.datalist.dao.DataSourceXML"
 alias="file:///$(SERVLETCONTEXT_REALPATH)/WEB-INF/db_xml/books.xml?/books/b
>
    <field
      name="TITLE"
      fieldType="varchar"
expression="title"
    />
    <field
      name="ISBN"
      fieldType="varchar"
expression="ISBN"
    />
    <field
      name="AUTHOR"
      fieldType="varchar"
expression="author/name"
    />
</table>
```

Chapter 26. Navigation

26.1. Introduction

Since version 1.1.4 DbForms contains two different navigation systems:

- A new one - in the datalist package - which implements a simplified data list pattern described by Claudio Fratarcangeli,Data List Handler: A Pattern for Large Search Result Sets, published on TheServerSide.com. The new system WORKS WITH A `whereClause`, AND DOES NOT RE-EXECUTE THE SEARCH QUERY EVERY TIME.
- The original one - in the classic package. The original code re-executes the search query every time the client requests a subset of the hits. THIS DOES NOT WORK WHEN A `whereClause` ATTRIBUTE IS USED IN THE `dbForm` TAG! THE OLD SYSTEM NEEDS TO BE CONFIGURED IN DBFORMS-CONFIG.XML AS IT IS NO LONGER THE DEFAULT.

  See the `dbform`tag in *the TagLib* [../../taglib/DbFormsTags_Frame.html]for more information.

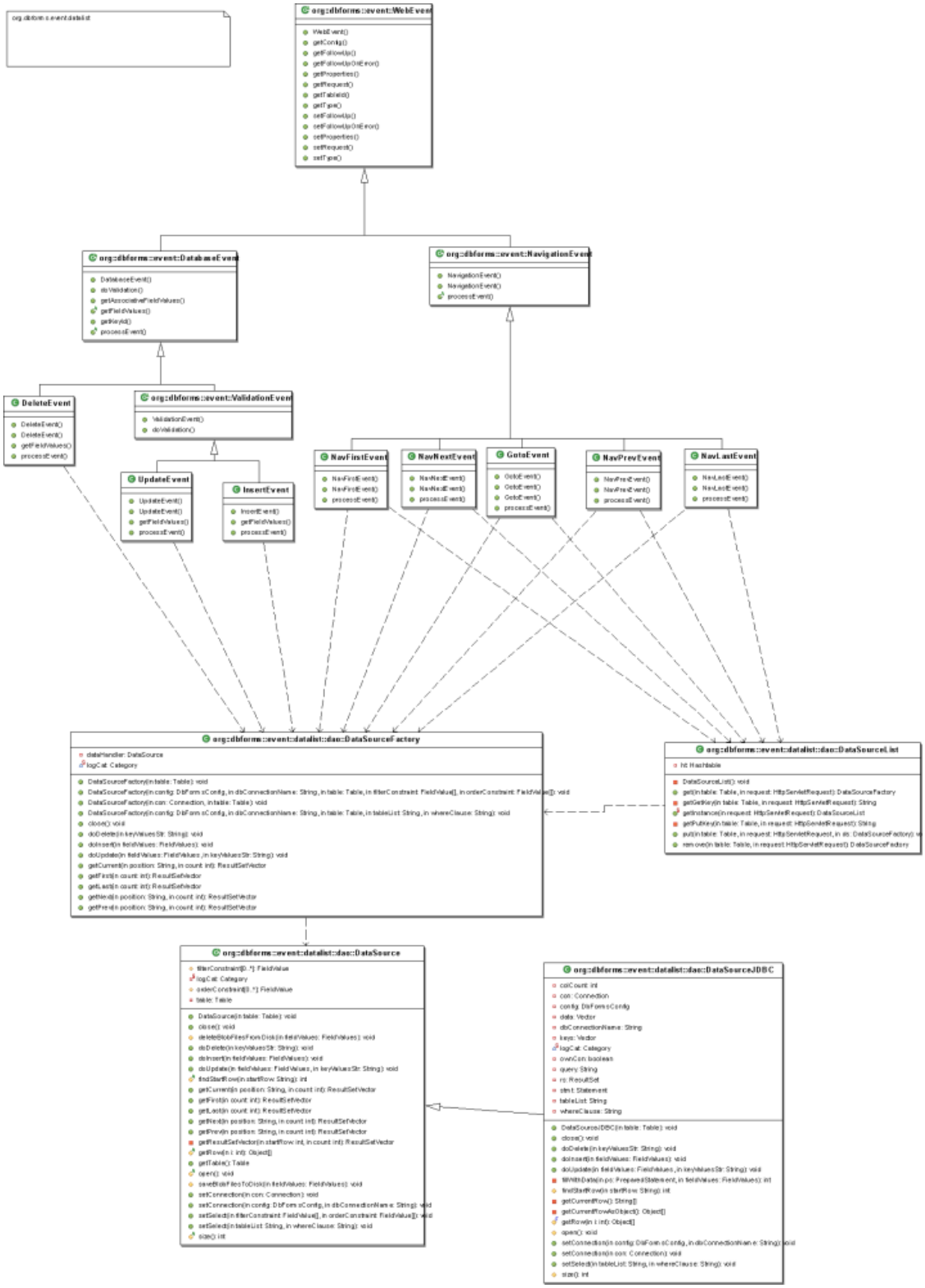
The new dataList system is now used by default!

26.2. Datalist navigation

The new datalist based navigation has following advantages:

- Data source is not only jdbc. It should be possible to write data source handlers for all other data, e.g. xml.
- Navigation should be independent from filtering. Navigation should be possible with free form select, too --i.e. use of a `whereClause` should be ok.

Figure 26.1. Class diagram of datalist navigation



To perform this goals the following points are done:

26.2.1. navigation events

On the contrary to the classic navigation, where each request will result in an new query to the database with special selections to get the next/previous/first and last data, the new navigation system makes an query to the database only if an GotoEvent happens. This query context represented by an DataSource object is stored in the session context an will be reused if one of the other navigation events happens.

26.2.2. DataSourceFactory

All navigation and database events use the DataSourceFactory class to retrieve an concrete implementation of the abstract DataSource class. The DataSourceFactory class gets takes the DataAccessClass attribute of the table class uses reflection to instanciate a new concrete implementation. If no DataAccessClass is given in table the DataSourceJDBC class is used. DataSourceJDBC is an implementation wich uses jdbc database access to get the data.

You can override this behaviour if you set the DataAccessClass in dbforms to another class.

Example:

```
<table name="line_flightbars" dataAccessClass="de.aucos.linedata.LineData" >
    <field name="flightbar_id" fieldType="integer" isKey="true" />
    <field name="flightbartype" fieldType="char" />
    <field name="flightbarstatus" fieldType="char" />
    <field name="stationid" fieldType="integer" />
    <field name="shuttleid" fieldType="integer" />
</table>
```

26.2.3. abstract class DataSource

The abstract class DataSource is the base class for all data access. Retrieving data is done by four abstract methods which must be overloaded by implementing class:

- protected abstract void open(): This method is called to open the resultset.
- protected abstract int size(): This method must return the size of the whole resultset with all data fetch
- protected abstract Object[] getRow(int i) This method must return the row at special index as an Object[]
- protected abstract int findStartRow(String startRow) This method gets the index of an row given by the special DbForms startrow string, This string has the format FieldID ":" Length ":" Value example: if key id = 121 and field id=2 then keyValueStr contains "2:3:121" if the key consists of more than one fields, the key values are seperated through "-" example: value of field 1=12, value of field 3=1992, then we'll get "1:2:12-3:4:1992"

This methods are called by the DataSource methods getCurrent, getFirst, getPrev, getNext and getLast. This datasource methods are called indirect by the DataSourceFactory during service of the navigation events.

To insert/update/delete data exists empty methods doInsert/doUpdate/doDelete which could be overloaded by the implementing classes. This method are called indirect by the DataSourceFactory which is called through the insert/update/delete events of the datalist events.

26.2.4. JDBC Access

The class DataSourceJDBC is an implementation of the DataSource class which retrieves data from JDBC data sources. doInsert/doUpdate/doDelete are implemented and do there work via SQL statements for insert/update/delete. The interesting part is retrieving of the data. This is done by a simplified data list handler pattern:

- protected abstract void open():

  This method just creates an java.SQL.Resultset and stores it in the class.

- protected abstract int size():

  This method first retrieves all data sets of the current java.SQL.Resultset, storing them in the internal data vector and returns then the size of the internal data vector.

- protected abstract Object[] getRow(int i)

  This method first tries to lookup the requested row in the data vector. This could be done, if the requested row is less then the current size of the data vector. If the requested row is outside the data vector the method will retrieve the next records from the java.SQL.Resultset up to the time, the size of the data vector is equal to the retrieved row or no more records are found.

- protected abstract int findStartRow(String startRow)

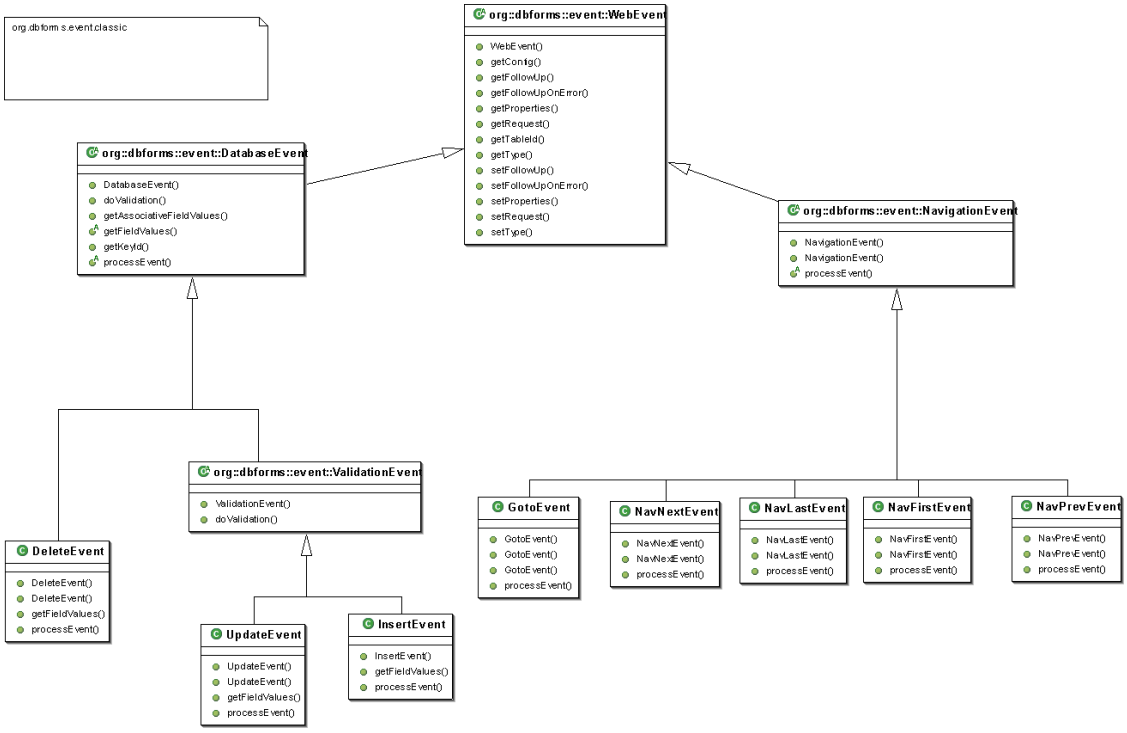
  To do the mapping of the DbForms key string a second vector is used. This vector stores all key value strings of each row from the java.SQL.Resultset. So searching for the mapping is done in two tasks: First the startRow is searched by an linear search in the keys vector. If found, the index of the key vector is return as result. If the startRow is not found in the keys vector, data from the java.SQL.Resultset is retrieved and stored up to the moment the key string of the just retrieved row is equal to the search row or nomore data is found.

26.3. Classic navigation

Classic navigation is moved from package org.dbforms.event to package org.dbforms.event.classic in version 1.1.4. The whole navigation is simplified, unnessary classes are removed and the usage of "instanceof EventType" is removed from DbForms. The often used Hashtable of FieldValues is rebuilt with an deligator pattern to FieldValues to make the usage more fault proved (see API documentation).

Details of the class structure could be seen in the class diagram.

Figure 26.2. Class diagram of classic navigation



26.3.1. HowTo use classic navigation

To use classic navigation you must define custom events in your dbforms-config.xml:

```
<!-- ========== custom events ================================== -->
<events>
  <database-events>
    <database-event type="insert" className="org.dbforms.event.classic.InsertE
    <database-event type="update" className="org.dbforms.event.classic.UpdateE
    <database-event type="delete" className="org.dbforms.event.classic.DeleteE
  </database-events>
  <navigation-events>
    <navigation-event type="navGoto" className="org.dbforms.event.classic.Goto
    <navigation-event type="navPrev" className="org.dbforms.event.classic.NavP
    <navigation-event type="navNext" className="org.dbforms.event.classic.NavN
    <navigation-event type="navFirst" className="org.dbforms.event.classic.Nav
    <navigation-event type="navLast" className="org.dbforms.event.classic.NavL
  </navigation-events>
</events>
```

This is is necessary starting after the 1.1.4dev release. Classic navigation was the default up through 1.1.4dev

26.4. ToDo

There is a missing point in the new system:

- Greate a thread that will throw away unused DataSources when the user navigate to another page.

- What do you think?

26.5. Classic Configuration in Dbforms-config.xml

To use the classic navigation system, please add the following within dbforms-config.xml:

```
<events>
  <database-events>
    <database-event type="insert" className="org.dbforms.event.classic.InsertEve
     <database-event type="update" className="org.dbforms.event.classic.UpdateEv
     <database-event type="delete" className="org.dbforms.event.classic.DeleteEv
   </database-events>

   <navigation-events>
     <navigation-event type="navGoto" className="org.dbforms.event.classic.GotoE
     <navigation-event type="navPrev" className="org.dbforms.event.classic.NavPr
     <navigation-event type="navNext" className="org.dbforms.event.classic.NavNe
     <navigation-event type="navFirst" className="org.dbforms.event.classic.NavF
     <navigation-event type="navLast" className="org.dbforms.event.classic.NavLa
   </navigation-events>
</events>
```

The new navigation system can also be enabled on a table by table basis by:

- Registering new navigation classes
- Overriding on a table by table basis

Part V. Additional Information

Table of Contents

Chapter 27. Additional Tools and Useful How To Examples

Additional tools for and information on how to use database forms is available in various locations including our wiki, how-tos for the bookstore, and in the `dbforms/contrib` directory

Any techniques you have for our bookstore How-To examples or Tools you want to contribute can be done through our mailing list.

27.1. Wiki

Please see and contribute your helpful solutions/suggestions to the *DbForm's Wiki* [http://jdbforms.sourceforge.net/wiki.php]

27.2. Changing Select Box Value From Another Select Box

An example to show how to change the values shown in a second select box based on the value chosen by a user in the first select box.

See the *HowTos* [http://localhost:8080/bookstore/howto/index.html] if you have the bookstore example installed, or view it under `dbforms/examples/bookstore/howto/index.html`

27.3. Tabbed Selector

An example to show how to "break up" a page using a tabSelector to allow for the input of one form over several pages

See the *HowTos* [http://localhost:8080/bookstore/howto/index.html] if you have the bookstore example installed, or view it under `dbforms/examples/bookstore/howto/index.html`

27.4. Printing a JasperReport

Shows how to to print a JasperReport from dbforms for either the whole list or a single row of data using <db:goto> tag

See the *HowTos* [http://localhost:8080/bookstore/howto/index.html] if you have the bookstore example installed, or view it under `dbforms/examples/bookstore/howto/index.html`

27.5. Create an Excel Worksheet

Shows how to to create an Excel worksheet report with Dbforms using <db:goto> tag

See the *HowTos* [http://localhost:8080/bookstore/howto/index.html] if you have the bookstore example installed, or view it under `dbforms/examples/bookstore/howto/index.html`

27.6. Use the Custom Formatter

How to use the Custom Formatter

See the *HowTos* [http://localhost:8080/bookstore/howto/index.html] if you have the bookstore example installed, or view it under `dbforms/examples/bookstore/howto/index.html`

27.7. Tool to Merge dbforms-config.xml Files.

There is an `update-config` ant task and other resources to assist in overlaying a newly generated `dbforms-config.xml` file to add needed attributes / elements or keep keep hand coded changes you want in your config.

It can be found in the `/dbforms/contrib/mark_dimon` directory

27.8. How To Use DbForms With JSTL

See the *HowTos* [http://localhost:8080/bookstore/contrib/ivan_codarin/HowtoUseWithJSTL/] if you have the bookstore example installed, or view it under `dbforms/examples/http://localhost:8080/bookstore/contrib/ivan_codarin/HowtoUseWithJSTL/`

27.9. How To Add Row Count Support

See the *HowTos* [http://localhost:8080/bookstore/howto/index.html] if you have the bookstore example installed, or view it under `dbforms/examples/bookstore/howto/index.html`

27.10. How To Monitor SQL Statements

It is possible to monitor the execution of sql statements of a JDBC application using two open source tools: P6Spy (see http://www.p6spy.com/) and IronEye SQL (see http://www.irongrid.com/). P6Spy is, according to its web site, an open source framework to support applications that intercept and optionally modify database statements. It does so by acting as a JDBC proxy driver. IronEye SQL is a lightweight Java tool that provides statistics and various performance graphs of all SQL that flows between an application and a database. It in this instance, it acts as a Java Swing GUI frontend to P6Spy.

Jahia software, http://www.jahia.org/, provides a SQL Profiler, licensed under the Jahia Open Software License. The profiler is an alternative to the IronEye SQL tool.

Using IronEye SQL and P6Spy to monitor a DbForms application is quite easy. The P6Spy JDBC driver is inserted between DbForms and your actual JDBC driver class. Then, DbForms operations can be monitored by connection to P6Spy from IronEye SQL. The following checklist shows the steps needed.

1. Download and install P6Spy and IronEye SQL.

2. Copy the three JAR files `ironeyesql.jar`, `p6spy.jar`, and `log4j-1.2.8.jar.jar` into the `WEB-INF/lib` directory of your web application.

3. Create a `spy.properties` file into `WEB-INF/classes` directory of your web application or include it into the JAR archive of your application. An extract of a sample file is shown below.

```
# ----
module.log=com.p6spy.engine.logging.P6LogFactory
#module.outage=com.p6spy.engine.outage.P6OutageFactory
module.monitor=com.irongrid.monitor.server.MonitorFactory
```

```
monitorport=2000

# oracle...
realdriver=oracle.jdbc.driver.OracleDriver

# are you sure you want to set me to true ? Have you read the docs ?
deregisterdrivers=true
# ----
```

4. Configure database access in `dbforms-config.xml` using the `dbconnection` element below as a basic model. The key portion is the `conClass` attribute.

```
<dbconnection
 isJndi                  = "false"
 isPow2                  = "true"
 defaultConnection       = "true"
 connectionProviderClass =
   "org.dbforms.conprovider.SingleConnectionProvider"
 conClass                = "com.p6spy.engine.spy.P6SpyDriver"
 name                    = "jdbc:oracle:thin:@devel:1521:myDbService"
 username                = "user"
 password                = "passwd"/>
```

5. Start your JDBC web application.

6. Execute the IronEye SQL application and press the "connect" button to connect to P6Spy. As the web application produces queries, they will be visible through IronEye SQL.

Chapter 28. DbForms Custom Tag Library

This tag library contains tags for placing data-forms, data-fields, templates and other DbForms-object on JSP-pages. This tag library may be imported into third party - tools like Macromedia UltraDev, Web-Gain's Studio etc. to enable application-building by drag drop. (All named products are properties of their respective owners.)

Hints for using this API:

- Please keep in mind that all tag-names and attribute-names are case sensitive.
- The annotation [Required] means that the application developer **must** provide a value for that attribute. Other attributes may be left out, default values will be provided instead.
- The annotation [RT Expr] means that an attribute get evaluated by the JSP container at runtime and therefore JSP expressions are allowed in that attribute values.

View the Tag Library **[../../taglib/DbFormsTags_Frame.html]**

Chapter 29. Description of fields send to/retrieved from html page

Table 29.1. description of internally used data fields

name	value	description
lang	output from Locale.getLanguage(), see java doc	the language setted inside dbforms
country	otuput from Locale.getCountry(), see java doc	the country setted inside dbforms
invtable	tableId	table
invname_[tableId]	conectioname	involved dbconnection for given table
autoupdate_[tableId]	true/false	autoupdate setted for given table
overridefieldcheck_[tableId]	true/false	If set to true the field values will not be checked if they are changed. The database update will be done always.
fu_[tableId]		follow up page for for given table
fue_[tableId]		follow up error page for given table
formValidatorName_[tableId]		name of validator
customEvent		empty field to transport custom events
firstpos_[tableId]		first position on current form for given table
lastpos_[tableId]		last position on current form for given table
f_[tableId]_[rowNo]@root_[fieldId]		value of field in table at row given row
of_[tableId]_[rowNo]@root_[fieldId]		old value of field in table at given row
pf_[tableId]_[rowNo]@root_[fieldId]		formatting pattern for given field at given row
k_[tableId]_[rowNo]@root		key value of table and row
f_[tableId]_insroot_[fieldId]		value of field to insert in table
of_[tableId]_insroot_[fieldId]		old value of field to insert in table
search_[tableId]_[fieldId]		search text for field in table
searchalgo_[tableId]_[fieldId]		search algorithm for field in table
searchmode_[tableId]_[fieldId]		search mode for field in table
sort_[tableId]_[fieldId]	none, desc, asc	sort mode for field in table

name	value	description
ac_update_[tableId]_[rowNo]@root_[buttonId]		update event for table at given row
ac_update_[tableId]_[rowNo]@root_[buttonId]_arname		name of associated radio button
ac_update_[tableId]_[rowNo]@root_[buttonId]_fu		follow up page
ac_update_[tableId]_[rowNo]@root_[buttonId]_fue		follow up error page
ac_updatear_[tableId]_[buttonId]		
ac_updatear_[tableId]_[buttonId]_arname		name of associated radio button
ac_updatear_[tableId]_[buttonId]_fu		follow up page
ac_updatear_[tableId]_[buttonId]_fue		follow up error page
ac_delete_[tableId]_[rowNo]@root_[buttonId]		delete event for table at given row
ac_delete_[tableId]_[rowNo]@root_[buttonId]_arname		
ac_delete_[tableId]_[rowNo]@root_[buttonId]_fu		
ac_delete_[tableId]_[rowNo]@root_[buttonId]_fue		
ac_deletear_[tableId]_[buttonId]		
ac_deletear_[tableId]_[buttonId]_arname		
ac_deletear_[tableId]_[buttonId]_fu		
ac_deletear_[tableId]_[buttonId]_fue		
ac_insert_[tableId]_root_[buttonId]		insert event for table
ac_insert_[tableId]_root_[buttonId]_fu		
ac_insert_[tableId]_root_[buttonId]_fue		
ac_first_[tableId]_[buttonId]		navigation event for table x
ac_first_[tableId]_[buttonId]_fu		
ac_first_[tableId]_[buttonId]_fue		
ac_prev_[tableId]_[buttonId]		navigation event for table
ac_prev_[tableId]_[buttonId]_fu		
ac_prev_[tableId]_[buttonId]_fue		

name	value	description
ac_next_[tableId]_[buttonId]		navigation event for table
ac_next_[tableId]_[buttonId]_fu		
ac_next_[tableId]_[buttonId]_fue		
ac_last_[tableId]_[buttonId]		navigation event for table
ac_last_[tableId]_[buttonId]_fu		
ac_last_[tableId]_[buttonId]_fue		
ac_reload_[tableId]_[buttonId]_fue		reload event. Reloads all data from current dataset. Query is not reexecuted! Changed fields stay changed
ac_reload_[tableId]_[buttonId]_fu		
ac_reload_[tableId]_[buttonId]_fue		
ac_reload_[tableId]_ins_[buttonId]		reload event in insert mode. Reloads all data from database. Changed fields stay changed
ac_reload_[tableId]_ins_[buttonId]_fu		
ac_reload_[tableId]_ins_[buttonId]_fue		
ac_reload_[tableId]_force_[buttonId]		force the reload event. Reloads all data from database. Changed fields will be reloaded
ac_reload_[tableId]_force_[buttonId]_fu		
ac_reload_[tableId]_force_[buttonId]_fue		
filter_[tableId]		prefix that all the parameters created by this tag have
filter_[tableId]_sel		index of the currently selected condition
filter_[tableId]_cond_[condId]		text of the currently selected condition
filter_[tableId]_cond_[condId]_value_[valueId]		current value of the input tag corresponding to the filterValue tag identified by [valueId]
filter_[tableId]_cond_[condId]_valuetype_[valueId]		type of the value identified by [valueId]

Chapter 30. Client Side Unit Testing

Client side unit testing has been introduced into dbforms! Now it is possible to test client side http responses during unit tests.

How it works:

- before starting unit tests, tomcat as application server is started with the bookstore application deployed.
- The unit tests uses *maxq* [http://maxq.tigris.org] and the HTTPClient package to test the responses of the server.

See `tests/test/dbforms/org/bookstore` for three example test cases.

To generate own unit tests you can use the new maven task `maxq`. This task first starts tomcat as application server and second starts maxq as recorder of your web pages. Only thing you must do to use the recorder is to set your `proxyport` to `localhost:8090` so that all request can be proxied by maxq. For detail information see the link above.

Part VI. UML diagrams

Table of Contents

Chapter 31. Class diagrams

31.1. org.dbforms.event package

The Event package contains the EventEngine and the Event Factory classes. Provides the DbEventInterceptor interface and the DbEventInterceptorSupport class.

Click here for the diagram [./images/figures/uml/class/event.ucd.png] picture.

31.1.1. EventEngine

This class is invoked by the Controller servlet. It parses a request to find out which event(s) need to be instanciated. The fine-grained parsing (parsing of additional data, etc) is done by the WebEvent-Object itself (in order to hide complexity from this class and to keep the framework open for implementations of new Event-classes).

31.1.2. EventFactory

The EventFactory abstract class provides the interface and the implementation of protected methods used by eventFactory subclasses (see NavEventFactory and DatabaseEventFactory)

31.1.3. NavEventFactory

The NavEventFactory abstract class provides the interface for a NavigationEvent concrete class (see NavEventFactoryImpl).

31.1.4. NavEventFactoryImpl

The NavEventFactoryImpl class is the concrete implementation of the NavEventFactory abstract class; it initializes also the default navigation events.

31.1.5. DatabaseEventFactory

The DatabaseEventFactory abstract class provides the interface for a DatabaseEvent concrete class (see DatabaseEventFactoryImpl).

31.1.6. DatabaseEventFactoryImpl

The DatabaseEventFactoryImpl class is the concrete implementation of the DatabaseEventFactory abstract class; it initializes also the default database events.

31.1.7. NavigationEvent

The NavigationEvent abstract class provides the interface for NavigationEvent subclasses. See the Datalist diagram page [REF HERE] for further informations about NavigationEvent subclasses.

31.1.8. ReloadEvent

ReloadEvent is used to reload the current page with data from Request object. When you want to do action on different field when one of them change. Use for field manipulation server side.

Example: Select child change when Select parent change.

31.1.9. NoopEvent

31.2. org.dbforms.event.datalist package

The Datalist package provides the implementation of a simplified data list pattern used by the new navigation system. See the navigation chapter for further details.

Click here for the diagram [./images/figures/uml/class/datalist.ucd.png] picture.

31.2.1. InsertEvent

This event prepares and performs a SQL-Insert operation.

31.2.2. UpdateEvent

This event prepares and performs a SQL-Update operation.

31.2.3. DeleteEvent

This event prepares and performs a SQL-Delete operation.

31.2.4. NavFirstEvent

This event scrolls the current ResultSet to its first row of data.

31.2.5. NavLastEvent

This event scrolls the current ResultSet to its last row of data.

31.2.6. NavLastEvent

This event scrolls the current ResultSet to its last row of data.

31.2.7. NavNextEvent

This event scrolls the current ResultSet to the next row of data; provides bounded navigation.

31.2.8. NavPrevEvent

This event scrolls the current ResultSet to the previous row of data; provides bounded navigation.

31.2.9. GotoEvent

This event forces the controller to forward the current request to a Request-Dispatcher specified by the Application-Developer in a `org.dbforms.taglib.DbGotoButton`.

31.3. org.dbforms.event.datalist.dao package

The DAO (data access object) package provides the DataSource* classes that manage the database operations (insert, update, delete).

Click here for the diagram [./images/figures/uml/class/dao.ucd.png] picture.

31.3.1. DataSourceList

DataSourceList holds a list of DataSourceFactory objects in the session context. It's needed by the navigation events to store the datasource by a per session mode. So it is possible to reuse the data between different calls and it's not neccessary to refetch again.

31.3.2. DataSourceFactory

The DataSourceFactory class creates and holds internally a DataSource object as dataHandler. The dataSource object type is specified by the dataAccess class name for the given table.

31.3.3. DataSource

The DataSource abstract class provides the interface for the objects that do data access operations. Specialized subclasses must implement abstract methods for a given data source type (i.e.: JDBC data source, XML datasource and so on).

31.3.4. DataSourceJDBC

Concrete DataSource class; uses JDBC to manage data form relational databases.

31.3.5. DataSourceXML

Concrete DataSource class; manages XML data.

31.3.6. org.dbforms.util.ResultSetVector

ResultSetVector class is an object wrapper for the data retrieved from a given data source.

31.4. org.dbforms.conprovider package

The conprovider package provides the ConnectionFactory used by DbForms to get a new JDBC connection object. The ConnectionFactory uses specialized ConnectionProvider objects to manage the interaction with connection pool libraries or other connection providers.

Click here for the diagram [./images/figures/uml/class/conprovider.ucd.png] picture, or read further documentation about connection support.

31.4.1. ConnectionFactory

Provides SQL Connection objects using the underlying ConnectionProvider instance.

31.4.2. ConnectionProvider

the ConnectionProvider abstract class defines the interface for specialized connectionProvider subclasses. Provides SQL Connection objects using the underlying ConnectionProvider instance.

Subclasses must extend this class and implement `initialize` and `getConnection` methods.

31.4.3. ConnectionProviderPrefs

The ConnectionProviderPrefs class holds the preferences for a given connectionProvider object.

31.4.4. JakartaConnectionProvider

Connection provider class for Apache Jakarta commons-dbcp component.

31.4.5. ProtomatterConnectionProvider

Connection provider for Protomatter Connection pool component.

31.4.6. SimpleConnectionProvider

Provides non-pooled connections.

Chapter 32. Sequence diagrams

32.1. The controller

This diagram [./images/figures/uml/sequence/event-servletController.usd.png] shows the interaction between the Controller class that receives the incoming http request and all the other classes that manage the event creation and its procession.

32.1.1. Steps

- The controller servlet gets the incoming http request that contains the data to process; the request object contains also the data that identify the event to instantiate and execute.

  Then, the controller instantiates the `EventEngine` object.

- The controller servlet calls the `eventEngine.generatePrimaryEvent()` method. `generatePrimaryEvent()` gets the `action` parameter from the `request`, try to identify the related event type (database or navigation) and then uses the appropriate factory to create the related event object. `generatePrimaryEvent()` returns the event object to the controller.

- The controller servlet checks if the returned event object is an instance of a `Database event` (i.e.: insert, delete, update events). If yes, validate the incoming data (the validation step occurs only if the `formValidatorName` request parameter exist and the current event is an `Update` or `Insert` event. The controller executes the `doValidation()` method that uses the `CommonsValidator` Jakarta component) and process the event object.

  Note: the Navigation events are managed by the `DbFormsTag` class. The controller manages only the database events.

- The controller servlet stores the event reference into the session context. MUST UPDATE the diagram picture...

- The controller servlet checks if the evetEngine has got involved tables: it find out which tables where on the jsp page (the view); one jsp file may contain multiple (db)forms, and each forms could contain many subforms nested inside.

  If yes, the controller gets the list of all the involved tables, generates and executes update events for all of them.

- The controller servlet forwards the client to the target resource referred from the `followUp` attribute of the jsp form if there are no errors. Else redirects to the resource specified by the `followUpOnError` form attribute.

32.2. The dbform JSP tag

This diagram [./images/figures/uml/sequence/taglib-DbFormTag.doStartTag.usd.png] shows the interaction between the `DbFormTag` class and all the other classes that are used to render the data into the target jsp page.

32.2.1. Steps

- The jsp page containing the `dbform` tag instantiates and initializes the `DbFormTag` class.

  The `DbFormTag` object gets the main table object, and checks if it has granted-privileges that make queries possible.

- The `DbFormTag` object checks if the main table contains interceptor objects. If yes, process those objects calling their `PRE_SELECT` hook up method.

- The `DbFormTag` object generates the HTML form.

- The `DbFormTag` executes the `initOverrulingOrder()` method to overrule other default declarations eventually done into the XML configuration file. `initOverrulingOrder()` initialises the datastructures containing informations about how table should be ordered. The information is specified in the JSP this tags lives in. This declaration OVERRULES other default declarations eventually done into the XML configuration file.

- The `DbFormTag` object sets the initial navigation position.

- The `DbFormTag` object executes the `checkLinkage()` method to check if the fieldvalue-list related to the position strings for a certain subform is valid in the current state.

- The `DbFormTag` object tries to retrieve from the `request` context the any previously instantiated `WebEvent` object. If the event doesn't exist, the `DbFormTag` object generates a new **navigation** event and then processes it.

  Note: the member `localWebEvent` action string is used to identify the navigation event type to instantiate.

- The `DbFormTag` object checks if the main table contains interceptor objects. If yes, process those objects calling their `POST_SELECT` hook up method.

- The `DbFormTag` object recalculates its cursor position.

32.3. The Goto event

This diagram [./images/figures/uml/sequence/datalist-gotoEvent.processEvent.usd.png] shows the interaction between the `Controller` object, the `GotoEvent` object and all the other actors that are used to process this navigation event.

32.3.1. Steps

- The `Controller` servlet calls the GotoEvent's `processEvent` method.

- The `GotoEvent` object parses the position string and build a FieldValues data structure representing the values of the fields decoded from that position. The position string uses the following format:

  ```
  field(0).id : field(0).length : field(0).value - ... - field(n).id : field(n).
  ```

  and contains data for all the fields of the current table. It calls `table.mapChildFieldValues` if the event was generated by a parent-child table; else calls `table.getFieldValues`.

Note: the Table's `mapChildFieldValues` method is used to build a new `FieldValues` object that inherits the fields' names from the child table and the fields' values from the parent table.

- Having the FieldValues data for the input position, the `GotoEvent` object try to build the key position string. It uses the same format as position string:

```
field(0).id : field(0).length : field(0).value - ... - field(n).id : field(n).
```

but contains data only for the fields that define the key set for the current table.

- The `GotoEvent` object gets the `DataSourceList` instance and remove the `DataSourceFactory` associated to the current table and to the current the current request URI from its internal list. (the current request URI example value: `/bookstore_v1_1_x_20030819/tests/testAUTHOR_list.jsp?AUTHOR`)
- The `GotoEvent` object instantiates a new `DataSourceFactory` object and stores it into the `DataSourceList`
- The `GotoEvent` object calls the `DataSourceFactory.getCurrent` method passing to it the key position string to return a new `ResultSetVector` object to the `Controller`

32.4. The NavNextEvent

This diagram [./images/figures/uml/sequence/datalist-navNextEvent.processEvent.usd.png] shows the interaction between the `Controller` object, the `NavNextEvent` object and all the other actors that are used to process this navigation event.

Note: `NavNextEvent` works similar to `NavFirstEvent`, `NavLastEvent` and `NavPrevEvent`, so the diagram above should explain the sequence of class messages for all those events. For further details, see the

```
public ResultSetVector processEvent(FieldValue[] childFieldValues,
                                    FieldValue[] orderConstraint,
                                    int          count,
                                    String       firstPosition,
                                    String       lastPosition,
                                    Connection   con,
                                    String       dbConnectionName)
                             throws SQLException
```

method of the selected NavigationEvent class.

32.4.1. Event initialization

This navigation event is initialized by the following parameters:

Table 32.1. initialization parameters and their meaning

action string	the input action string generated by the parent jsp page
step width	the number of records to show into the target jsp

	page

Table 32.2. navigation event parameters generated by a form

action string	
step width	1

32.4.2. Steps

- The `Controller` servlet calls the NavNextEvent's `processEvent` method.
- The `NavNextEvent` object retrieve the instance of the `DataSourceList` object.
- The `NavNextEvent` object get the `DataSourceFactory` related to the current table and to the current request URI.
- The `NavNextEvent` object retrieve the `FieldValues` object (a data structure representing the values of the fields decoded from a given position string) related to the the last resultset position string. [ADD A NEW MESSAGE INTO THE DIAGRAM] Then it get the key position string for the current table from that `FieldValues`.
- The `NavNextEvent` object calls the `DataSourceFactory.getNext` method to obtain the `ResultSetVector` object. If that `ResultSetVector` is null, `NavNextEvent` calls `DataSourceFactory.getLast` to obtain the last records from the current table (it simulates a `NavLastEvent` event).

32.5. The InsertEvent

This diagram [./images/figures/uml/sequence/datalist-navInsertEvent.processEvent.usd.png] shows the interaction between the `Controller` object, the `InsertEvent` object and all the other actors that are used to process this database event.

Note: `InsertEvent` works similar to `UpdateEvent` and `DeleteEvent`, so the diagram above should explain the sequence of class messages for all those events. For further details, see the

```
public void processEvent(Connection con)
    throws SQLException, MultipleValidationException
```

method of the selected DatabaseEvent class.

32.5.1. Event initialization

This database event is initialized by the following parameters and example values:

Table 32.3. initialization parameters and their meaning

action string	the input action string generated by the parent jsp page

event type	the string identifying the event type
table id	the dbforms' table identifier (used to build the query string)
key id	the dbforms' key identifier related to the current table
insert button count	button count used to identify individual insert buttons

Table 32.4. insert event parameters generated by a main form

action string	ac_insert_0_root_5
event type	ac_insert
table id	0
key id	root
button count	5

Table 32.5. insert event parameters generated by a subform

action	ac_insert_1_0@root_17
event type	ac_insert
table id	1
key id	0@root
button count	17

32.5.2. Event steps

- The `Controller` servlet calls the InsertEvent's `processEvent` method.
- The `InsertEvent` object checks if the current user has got the INSERT privilege.
- The `InsertEvent` object builds a `FieldValues` object representing the set of `FieldValue` objects to process. Then it copy the `FieldValues` data into a new `fieldValueHashTable` object that will be used as interceptor's input parameter.
- The `InsertEvent` object checks if there is any interceptor object binded to the current main table. If yes, executes its `PRE_INSERT` hook method.

  Then the `InsertEvent` object synchronizes data from the `fieldValueHashTable` structure used as interceptor's input parameter (the hash table could contains values modified by the interceptor code) to the original `FieldValues` object (that will be used by the database insert operation).
- The `InsertEvent` object builds a `FieldValues` object representing the set of `FieldValue` objects to process. Then it copy the `FieldValues` data into a new `fieldValueHashTable` object that will be used as interceptor's input parameter.
- The `InsertEvent` object calls the `checkSufficentValues` method to check a list of condi-

tions on the the input FieldValues object:

- it must contains the same number of fields as the current main Table
- any autoInc field must NOT have a related FieldValue object (generated by a request parameter) because that field value must be calculated by the underlying RDBMS

if all the conditions above are true, the system continues the execution of the event flow, else it generates a `SQLException` that stops the event procession.

- The `InsertEvent` object retrieve the `DataSourceFactory` and executes its `doInsert` method to store the data contained into the `FieldValues` object (the new record to insert) into the underlying database.
- The `InsertEvent` object checks (again) if there is any interceptor object binded to the current main table. If yes, executes its `POST_INSERT` hook method.

Part VII. Appendices

Table of Contents

Appendix A. Changes

Please see the changes file in the release or cvs for updated information.

Appendix B. To Do List

Please see the toDo file in the release or best yet CVS for updated information.

Appendix C. Acknowledgements of Joachim Peer

I want to thank all the people evaluating DbForms and providing feedback to me. Especially I want to thank Philip Grunikiewicz and his team for their valuable contributions and comments.

I also want to express a big thank you to the people at Apache Group, especially Craig Mc Clanahan, for the wonderful software called Struts which is a generic, flexible and extensible framework for application programmers.

It is not just a great inspiration (regarding input form validation, form repopulation, etc.) but also a great repository of cool software.

In detail, I used the following components of Struts.

XML digester (this is really a great thing!)

Lots of code from BaseHandlerTag (co-written by Don Clasen and Luis Arias, Apache Group)

The XSL-stylesheet for converting taglib-docs to taglib-descriptors

These components helped me setting up DbForms much faster than I thought it would be possible.

Appendix D. Authors

appendix (through V. 0.9) -- holds a masters degree in Management Information Systems of Johannes Kepler University, Linz, Austria, with a focus in the fields of software engineering, multimedia application development and knowledge and process management. He is a Sun certified programmer for the Java 2 platform.

Philip Grunikiewicz (through v 1.1)

Shawn Asamoto-Brown (from version 1.1.3pr1)

Appendix E. References

Table E.1. References

[Johnson]	Ralph Johnson and Brian Foote #Designing Reusable Classes # http://st-www.cs.uiuc.edu/users/johnson/frameworks.html#Definition
[Servlet]	http://java.sun.com/products/servlet/
[JSP]	http://java.sun.com/products/jsp/
[Taglib]	http://java.sun.com/products/jsp/taglibraries.html
[Struts]	http://jakarta.apache.org/struts/index.html
[Gamma]	Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides #Design Patterns: Elements of Reusable Object-Oriented Software# Addison Wesley. October 1994.
[Poolman]	http://sourceforge.net/projects/poolman/