FirstSpirit™
Unlock Your Content

FirstSpirit™ DynamicDatabaseAccess

FirstSpirit™ Version 5.0

Version	**2.8**
Status	**RELEASED**
Date	**2012-12-03**
Department	FS-Core
Copyright	2012 e-Spirit AG
File name	DDBA50EN_FirstSpirit_DynamicDatabaseAccess

e-Spirit AG
Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 477 77-0
F +49 231 . 477 77-499

info@e-spirit.com
www.e-spirit.com

e-Spirit

Table of contents

1 Introduction

The FirstSpirit DynamicDatabaseAccess documentation describes the licence-dependent FirstSpirit module for connecting different database technologies. The FirstSpirit DynamicDatabaseAccess platform can be used to display and edit contents from a database in a web application.

Different possible uses are feasible:

- FirstSpirit DynamicDatabaseAccess is used to display the database contents only. In this case, for example, generation of the data records from the FirstSpirit Content-Store is no longer necessary. (In this case the JSP page is delivered directly from the servlet engine).
- FirstSpirit DynamicDatabaseAccess is used for display and maintenance of the database contents. The structures and database tables are defined in FirstSpirit™; the data records are created and changed using the FirstSpirit DynamicDatabaseAccess web application.
- FirstSpirit DynamicDatabaseAccess can of course also directly change the database contents of the FirstSpirit Content-Store. The changes are immediately available in the (dynamic) JSP pages. Within a static page the changes are not displayed until after generation.

FirstSpirit DynamicDatabaseAccess uses database schemata from the FirstSpirit Template-Store. These schemata can be defined within a project. A graphic editor is available for editing a database schema; this editor can be used to create the required database schema. Each schema can fall back on existing database structures or can create new table structures in an existing database (for further information on using database schemata, see "FirstSpirit Manual for Developers (Basics)" or the "FirstSpirit Online Documentation").

Remote access to database schemata from other FirstSpirit projects is also possible (remote projects).

If a valid licence exists for the module, a web component can be installed via the FirstSpirit Server and Project Configuration. This component is used to build the web application, which access the relevant database via JSP tags and servlets (see Chapter 2 ff.).

Attention: The module FirstSpirit DynamicDatabaseAccess demands the use of a JDK version higher or equal to 1.5. It has to be ensured that this JDK version is supported by the used servlet engine.

1.1 Topic of this documentation

Chapter 2: Describes the installation and configuration of the web component via the FirstSpirit Server and Project Configuration (from page 5).

Chapter 3: Introduction to the tag library for controlling the web application with detailed examples of creating, changing and deleting data records and of formulation of database queries via FirstSpirit DynamicDatabaseAccess (from page 11).

2 Configuration

2.1 Installing the module on the server

The FirstSpirit DynamicDatabaseAccess module must first be installed within the Server and Project Configuration application. To this end, the "Modules" menu entry is selected in the Server Properties area. Click the "Install" button to open a file selection dialog. The fsm file (fs-integration.fsm) to be installed can be selected here. The successfully installed file is then displayed in the "Server Properties" dialog:

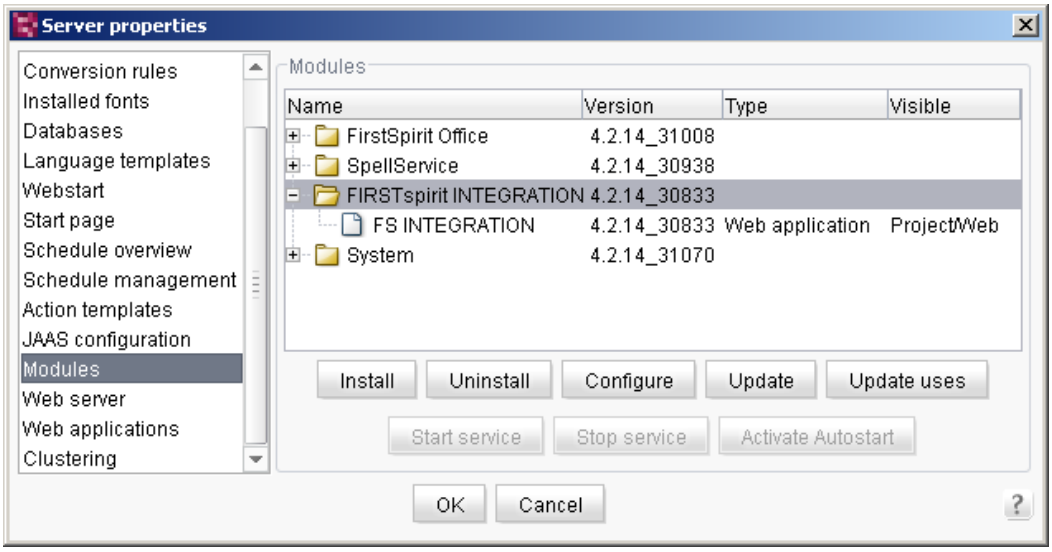

Figure 2-1: Installing the module on the FirstSpirit server

The "FS INTEGRATION" web application is part of the FirstSpirit module DynamicDatabaseAccess. The web application makes available JSP tags and servlets which can be used and opened within the project (see Chapter 3.2 page 14).

The component is "viewable" for the "Project/Web" areas. It is therefore a "local web" component. After installation, this can be added to the different web areas (preview, staging, live) within the required projects (see Chapter 2.2 page 6).

For further information on this dialog see "FirstSpirit Manual for Administrators".

2.2 Installing the web application in the project

The web application must now be installed in the required project. Open the "Web Components" menu entry within the project properties. The web components for a project can be activated in this area.

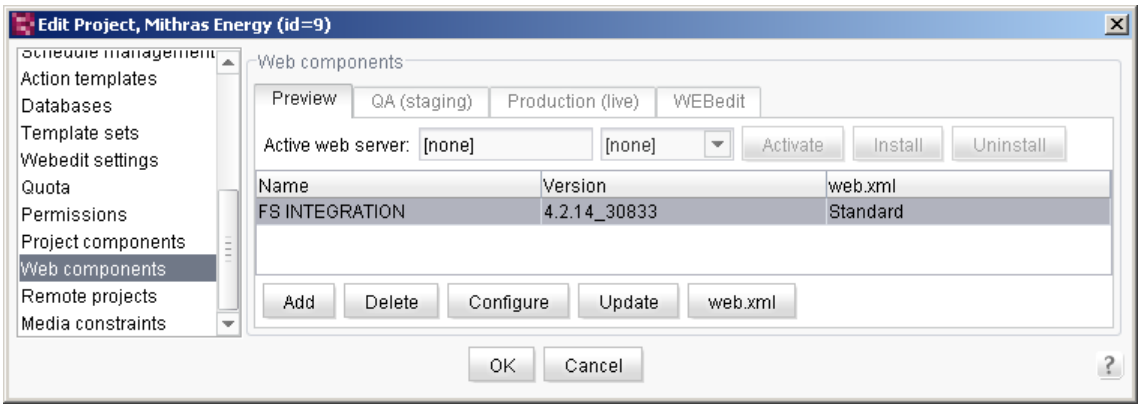

Figure 2-2: Installing the web application within the web areas

Three different web areas exist for each project. The respective tab can be used to individually activate and configure the web components for each area:

Figure 2-3: Web areas within a project

- Preview: Project contents location for which a preview has been requested.
- QA (staging): Location for the generated project contents
- Production (live): Location for the published project contents

Click the button to open the "Add" dialog. The list displayed contains all web components installed on the server (see Chapter 2.1 page 5).

After adding it to a web area it is possible to configure the components, either with a GUI generated by the component or a generic GUI (see 2.3 page 7). After configuring the components must be activated. A component within a project can be activated or deactivated for specific areas only

For further information on this dialog see "FirstSpirit Manual for Administrators".

2.3 Configuring the web application

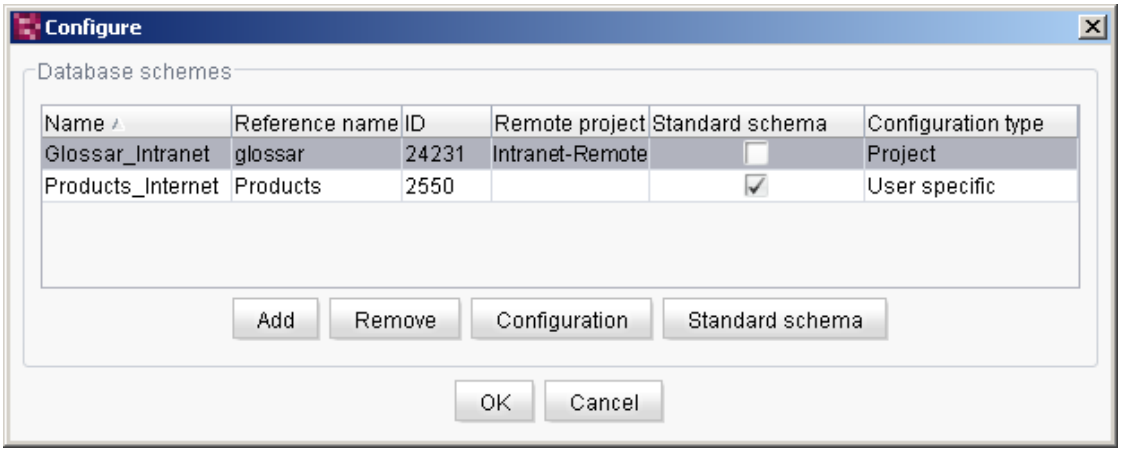

Figure 2-4: Configuring the Integration web application

Click the "Configure" button (cf. Figure 2-2) to configure the component within the web area. The project's database schemata to be used for the respective web area are displayed in the configuration dialog:

Name: Contains the symbolic name for the configuration of a database schema. This name is defined by the administrator on adding a configuration.

Reference name: Contains the reference name of the schema (from the project).

ID: Unique ID of the schema in the project.

Remote project: If the schema stems from a remote project this column contains the symbolic remote project name.

Standard schema: The name “Standard schema” initially identifies the schema first added to the configuration. There is one standard schema for each configuration (within a web area). The standard schema is required if e.g. the JSP tag is used to manually access a specific database table. The standard schema is used if an explicit schema is not defined there.

Configuration type: This column identifies whether a configuration has been defined as a user-specific or a project-specific configuration. As a default, a configuration is created to be project-specific (type: "Project"). The user-specific configuration is described in Chapter 2.3.1.

Add Click this button to add a new database schema to the configuration from the project's Template-Store. A dialog opens with the schemata available for the local project. If the local project has remote access to other FirstSpirit projects these are displayed in separate tabs:

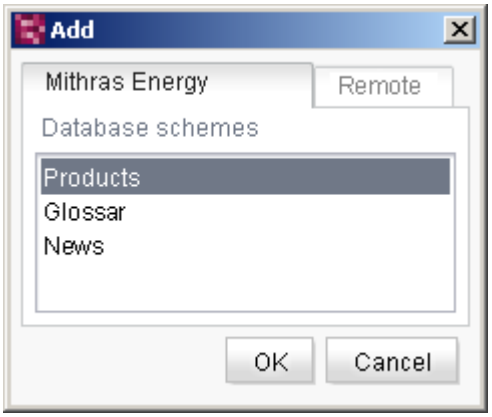

Figure 2-5: Selecting a FirstSpirit schema for the configuration

Remote access to the schemata of other FirstSpirit projects is only possible if the remote type "Remote Schemata" has been activated within the remote project configuration.

Remove Click this button to remove an existing database schema from the configuration.

Configuration Click this button to configure a database schema for FS Integration (see 2.3.1 page 9).

Standard schema Click this button to define an existing database schema from the configuration the standard schema. The standard schema to date remains as a schema within the configuration but is no longer labelled as the standard schema.

2.3.1 Configuring the database schema

Click the "Configure" button within the configuration dialog (cf. Figure 2-4) to open a further configuration dialog for the selected database schema.

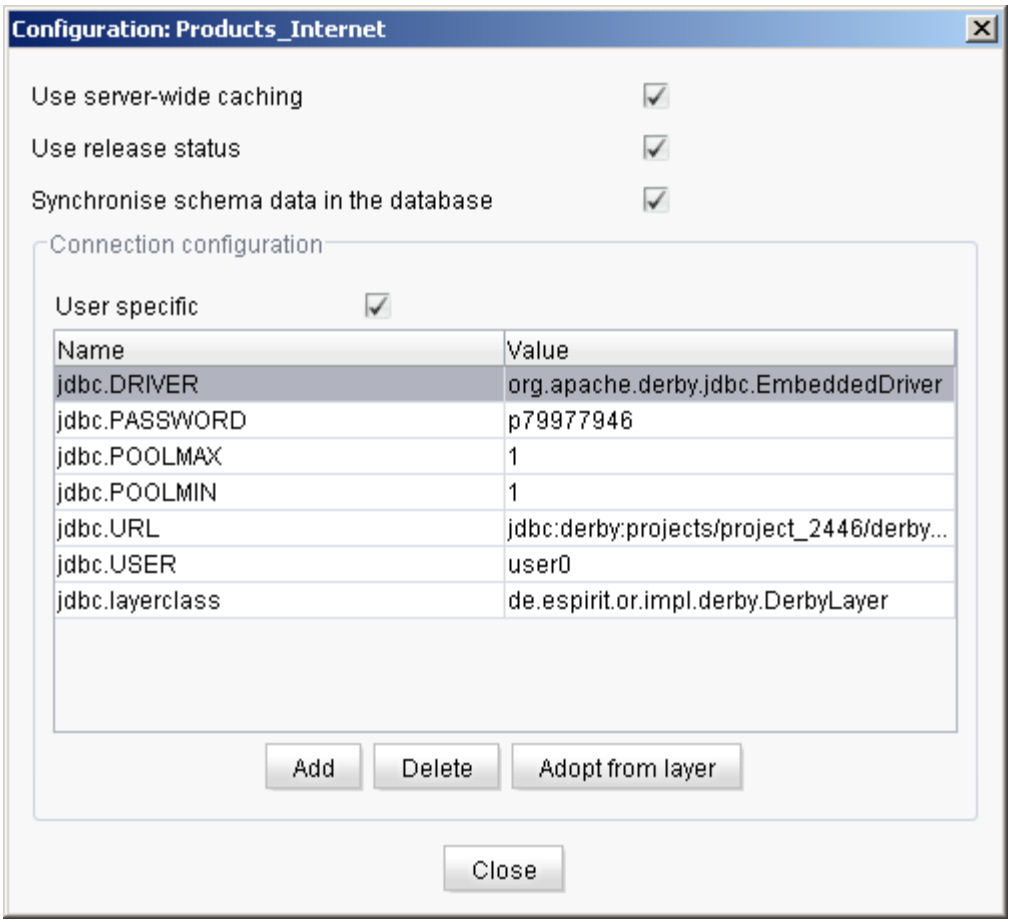

Figure 2-6: Database schema configuration

Different options and parameters are available for configuring the connection with the database for configuration of a database schema.

Using server-wide caching: If this option is activated, server-wide caching of the objects is switched on (“application" scope). Local user caching (HTTP session scope) is used (default setting) if the option is deactivated.

Use release status: If this option is activated the release status (of the database contents) is used for all queries. If this option is deactivated the current status is returned for all queries.

Synchronise schema data in the database: If this option is activated, when the module is

initialised the data of the schema (in the module) is compared with the contents of the database. This function therefore corresponds to that of saving a schema in the FirstSpirit Template-Store. However, the schema contents from the FirstSpirit module (xml file) are adopted instead of the schema contents from the project.

Further parameters for the connection with the database can be configured below these options:

User-specific: If this option is activated a manual or user-specific configuration of the database layer can be defined. If this option is activated the values currently saved in the module (or within the web configuration) are adopted. The values can potentially differ between activated and not activated following an export or changes by the administrator.
If the option is deactivated the database layer used by the schema within the project is used. The parameters are then also adopted from the schema's layer configuration.

For a description of the parameters, see "FirstSpirit Manual for Administrators" (Chap. 4.8.1. "Configuration of data sources" ff.).

Click this button to add further parameters to the configuration.

Click this button to remove parameters from the configuration.

Click this button to adopt the layer configuration of the project instead of the currently saved connection configuration.

3 Syntax

3.1 General information

The following chapter explains the tag library functions for controlling the web application. Not only are general, formal definitions are described (e.g. specification of a prefix for the JSP tag) (see Chapter 3.1), but also use of the available JSP tags is explained (from Chapter 3.2). Further chapters contain current examples on creating, changing and deleting data records (see Chapter 3.3 and Chapter 3.4) and a brief introduction to queries via FirstSpirit DynamicDatabaseAccess (see Chapter 3.5).

3.1.1 Prefix

In order to be able to use FirstSpirit DynamicDatabaseAccess the relevant taglibs must be given in the JSP pages. This documentation uses the prefix "fsi" for FirstSpirit DynamicDatabaseAccess tags and the prefix "c" for the tags of the core JSTL[1] functions.

Example for integration in JSP pages:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="fsweb" prefix="fsi" %>
```

Here it must be noted that the URI for FirstSpirit is always called “fsweb".

Important: If the “fsi" prefix is changed to another value this prefix must be used for the individual tags, i.e. “<**myPrefix**:ref>” instead of “<**fsi**:ref>".

3.1.2 Object/Attribute references

Object and/or attribute references are used in various places within the web module. Irrespective of the use, these can involve lists, individual objects or attributes of individual objects. An object is separated from the attribute or from the attribute chain by a full stop.

[1] JavaServer Pages Standard Tag Library – For information on JSTL see http://wikipedia.org/wiki/JSTL

Example:

```
OBJECT.ATTRIBUTE_1[.ATTRIBUTE_2][.ATTRIBUTE_3]
```

There are two types of object references: relative and absolute.

In general, an absolute object reference is used first, e.g. for an iteration. All other operations are then carried out with relative references.

Absolute references are characterised by parenthesis behind the symbolic identifier, e.g. "movie()". The symbolic identifier is the name of a table from a schema.

- **All data records:** If there are no characters between the opening and closing parenthesis all data records (i.e. a list of entity objects) are returned to the given table. This is comparable with the return of the header function "contentSelect" (for further information, see FirstSpirit Online Documentation[2]).
- **A data record:** If a special data record (an entity object) is to be returned, the unique ID of the data record can be given within the brackets, e.g. "movie(12)".
- **New data record:** The special character "*" (asterisk) is used to create a new data record, e.g. "movie(*)".

The available attributes, which can be applied to an object reference, can be taken from each table in the schema. Their use in this position is similar to use of the system object "#row" in table templates (for further information, see FirstSpirit online documentation[3]).

Example:

```
$CMS_VALUE(#row.title)$
  $--
    statically outputs the "title" column of the current data record
  --$

${m.title}
  $--
    dynamically outputs the column "title" of a data record
  --$
```

[2] Area: .../templatedevelopment/templatesyntax/functions/intheheader/contentselect/contentselect.html
[3] Area .../ templatedevelopment/templatesyntax/systemobjects/row/row_1.html

Examples for the notation of absolute object references:

Notation	Meaning
movie()	all "movie" objects or all data records in the "movie" table
movie(12)	"movie" object with the unique key "12" or the data record with ID "12" from the "movie" table
movie(37).title	"Title" attribute of the "movie" object with the unique key "37" or the "title" column of the data record with ID "37" from the "movie" table
movie(*).title	"Title" attribute of a new "movie" object or the "title" column of a new data record of the "movie" table

Examples for relative references: In the syntax examples "m" is a "movie" object which was determined by using the <fsi:find> tag.

Notation	Meaning
m	A "movie" object which is represented by "m" or a data record (entity) of the "movie" table.
m.actors	"actors" object (of N-relation) of "m" or the saved values of the external key relation (name "actors") of the "movie" to another table.
m.title	"Title" attribute of "m" or the "title" column of a data record in the "movie" table.

3.1.3 Variables defined by tags

Variables are defined in the <fsi:getQueryDetails>, <fsi:iterateResults>, <fsi:navigation>, <fsi:navBottom>, <fsi:navFrame> and <fsi:navTop> tags.

Overlapping can occur with manually defined variables in the JSP page. The following variable identifiers should not be used (or used circumspectly) to avoid overlapping:

- firstFramePage
- hasMore
- lastFramePage
- navPageNo
- navUrl
- pageNo
- pageSize
- parameter
- query
- totalPages
- totalResults

Which tag defines which variable is given in the individual tag descriptions (see Chapter 3.2).

3.2 JSP tags

3.2.1 Search for and display data

FirstSpirit DynamicDatabaseAccess can be used to search for and display data, e.g.:

- Search for an individual data record using (<fsi:find>)
- Search for several data records using (<fsi:search>)
- page-wise output of the data records using (<fsi:iterateResults> in conjunction with <fsi:search pageSize="....">)
- complete output of the data records using (<fsi:iterateResults> in conjunction with <c:forEach>)

A large number of tags are available for this:

- <fsi:setSchema> (specification of a schema) (see Chapter 3.2.1.1 page 15).
- <c:out> / ${...} (output of attributes) (see Chapter 3.2.1.2 page 16).
- <fsi:ref> (define references) (see Chapter 3.2.1.3 page 17).
- <fsi:find> (search for objects) (see Chapter 3.2.1.4 page 21).
- <c:forEach> (output of lists) (see Chapter 3.2.1.5 page 23).
- <fsi:search> (search for data records) (see Chapter 3.2.1.6 page 24).
- <fsi:query> (define queries) (see Chapter 3.2.1.7 page 26).
- <fsi:iterateResults> (complete output) (see Chapter 3.2.1.8 page 27).
- <fsi:getQueryDetails> (info on query) (see Chapter 3.2.1.9 page 28).

Note: As the database queries are temporarily stored the cache should be reset for an up-to-date display. This is done by importing the "WebContext" class:

```
<%@ page import="de.espirit.firstspirit.opt.integration.web.WebContext" %>
```

The cache can then be reset with the following call:

```
<%
  WebContext.getWebContext(pageContext).getSession().rollback();
%>
```

3.2.1.1 Specifying the schema (<fsi:setSchema>)

The JSP tag <fsi:setSchema> can be used to specify a schema.

All tags located between the opening and closing tag use this schema.

The "schema" attribute is to be given for the tag, which expects the unique identifier of the schema from the module configuration as a value.

Attributes:

Attribute	Expected value	Mandatory parameter
schema	unique identifier of the schema from the module configuration	No

Example:

```
<fsi:setSchema schema="movies">
  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        <c:out value="${m.title}" />
      </p>
    </c:forEach>
  </fsi:search>
</fsi:setSchema>
```

In the example the schema to be used by the <fsi:search>, <fsi:query> and <c:forEach> tags is set to "movies".

3.2.1.2 Output of attributes (<c:out> / ${...})

Either the <c:out> tag of the JSTL[4] or the JSP EL notation (${...}) can be used for the output of attributes.

Attributes of the <c:out> tag:

Attribute	Expected value	Mandatory parameter
value	attribute to be output in JSP EL notation (${...}) or a fixed value	Yes
escapeXml	specifies whether the characters "<", ">", "&", "'" and "'" are to be converted into HTML entities or not. Possible values are: "true" or "false"	No
default	Default value which is output if "value" zero is returned.	No

[4] JavaServer Pages Standard Tag Library – For information on JSTL see http://wikipedia.org/wiki/JSTL

Example of attribute output with the <c:out> tag:

```
<fsi:find item="movie(98)" resultName="m">
      Title: <c:out value="${m.title}" default="" />
</fsi:find>
```

Example of the output of a fixed value with the <c:out> tag:

```
<fsi:find item="movie(98)" resultName="m">
      Fixed value: <c:out value="VALUE" />
</fsi:find>
```

Example of attribute output with the JSP EL notation:

```
<fsi:find item="movie(98)" resultName="m">
      Title: ${m.title}
</fsi:find>
```

Important: In earlier versions the <fsi:get> tag was used for attribute output but it is no longer supported. However, it is possible to convert the <fsi:get> syntax into a <c:out> or JSP EL syntax.

Example:

```
Old syntax:
<fsi:find item="movie(98)" resultName="m">
      Title: <fsi:get item="m.title" />
</fsi:find>


New syntax:
<fsi:find item="movie(98)" resultName="m">
      Title: <c:out value="${m.title}" />
</fsi:find>

or

<fsi:find item="movie(98)" resultName="m">
      Title: ${m.title}
</fsi:find>
```

3.2.1.3 Define unique references (<fsi:ref>)

The saving of values within the JSP pages with HTML input components or with JavaScript-based editors (e.g. TinyMCE, HTMLArea, FCK Editor, etc.) is realised in an HTML form. Each HTML input component has a parameter "name" and a "value". When it is saved the store servlet tries to save the transferred value on the basis of the "name" parameter (see Chapter 3.3).

In order for the store servlet to be able to assign the correct columns or data records in the

database it requires a unique reference to the column and data record. The <fsi:ref> tag can be used to obtain a unique reference to an attribute or object for the save.

Attributes:

Attribute	Expected value	Mandatory parameter
Value	Object or attribute reference, e.g. "m.title"	Yes
resultName	Variable name for further use in the JSP page. If the parameter is given the unique reference is not output.	No

Important: It is only necessary to use the <fsi:ref> tag to save or change values (see also "Store Servlet" description in Chapter 3.3).

The tag supports two attributes: "value" and "resultName". The "value" attribute is a mandatory parameter. If the "value" parameter is used without "resultName" the unique reference is directly output:

```
<form action="<%= application.getContextPath() %>/do.store"
      method="post">

  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        Title:
        <input name="<fsi:ref value='m.title' />"
               value="<c:out value='${m.title}'
                             escapeXml='true' />" />
      </p>
    </c:forEach>
  </fsi:search>

  <input type="hidden"
         name="schema"
         value="movies" />
  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />
  <input type="submit"
         value="Submit" />
</form>
```

If the "resultName" parameter is given the unique reference is no longer directly output. The reference must be output manually using the <c:out> tag or the JSP EL syntax (${...}):

```
<form action="<%= application.getContextPath() %>/do.store"
      method="post">

  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <fsi:ref value="m.title" resultName="t" />
      <p>
        Title:
        <input name="${t}"
               value="<c:out value='${m.title}'
                             escapeXml='true' />" />
      </p>
    </c:forEach>
  </fsi:search>

  <input type="hidden"
         name="schema"
         value="movies" />
  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />
  <input type="submit"
         value="Submit" />
</form>
```

Another important aspect of use of <fsi:ref> is the selection and saving of data from an external key relation. In this case the <fsi:ref> tag is not only used for the "name" attribute but is also simultaneously used for the "value" attribute:

```
<form action="<%= application.getContextPath() %>/do.store"
      method="post">

  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        <select name="<fsi:ref value='m.rating' />">

          <fsi:search resultName="rL">
            <fsi:query>
              <QUERY entityType="rating" />
            </fsi:query>

            <c:forEach items="${rL}" var="r">
              <option value="<fsi:ref value='r' />"
                      <c:if test="${m.rating == r}">
                        selected="selected"
                      </c:if>>
                ${r.abbreviation} (${r.name})
              </option>
            </c:forEach>
          </fsi:search>

        </select>
      </p>
    </c:forEach>
  </fsi:search>

  <input type="hidden"
         name="schema"
         value="movies" />
  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />
  <input type="submit"
         value="Submit" />
</form>
```

3.2.1.4 Object search (<fsi:find>)

The <fsi:find> tag can be used to search for precisely one object and save it in a variable. The variable can only be accessed between the opening and closing tag of <fsi:find>. The tag has both mandatory parameters "item" and "resultName", as well as the optional parameter "schema".

Attributes:

Attribute	Expected value	Mandatory parameter
item	Object or attribute reference, e.g. "movie(103)"	Yes
resultName	Variable name for further use within the tag.	Yes
schema	unique identifier of the schema from the module configuration	No

The parameter "item" is used to give an object reference (e.g. movie(103)) for the search. Here it must be noted that the object reference returns one object only and not a list of objects.

The variable name for further use is defined with the "resultName" parameter.

The "schema" parameter can be used to define a schema. The object search then takes place within the defined schema only:

```
<fsi:find item="movie(104)" resultName="m" schema="movies">
  <p>
    Title: ${m.title}
  </p>
</fsi:find>
```

The <fsi:setSchema> tag can also be used instead of the "schema" parameter.

Important: Use of the <fsi:setSchema> tag is recommended as use of the "schema" attribute is susceptible to errors (especially with nesting) :

```
<fsi:setSchema schema="movies">
  <fsi:find item="movie(104)" resultName="m">
    <p>
      Title: ${m.title}
    </p>
  </fsi:find>
</fsi:setSchema>
```

If neither the "schema" attribute nor the <fsi:setSchema> tag is given the search automatically takes place in the standard schema (cf. Chapter 2.3 page 7).

As an alternative to using <fsi:find>, the search can also be realised with <fsi:search>:

```
<fsi:setSchema schema="movies">
  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" limit="1">
        <EQ attribute="fs_id"
            datatype="java.lang.Integer"
            value="104" />
      </QUERY>
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        Title: ${m.title}
      </p>
    </c:forEach>
  </fsi:search>
</fsi:setSchema>
```

3.2.1.5 Output of lists (<c:forEach>)

The <c:forEach> tag of the JSTL[5] can be used to output lists.

The list of objects is handed over to the tag with the mandatory parameter "tags" (e.g. ${mL})). The content of the tag is listed once for each element in the list.

If an element in the list is to be accessed with a fixed identifier, this can be defined with the optional parameter "var". The element can be accessed within the tag with <c:out> or JSP EL (${...}) (cf. Chapter 3.2.1.2 page 16).

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

Attributes:

Attribute	Expected value	Mandatory parameter
items	List of objects	Yes
var	Variable name for further use within the tag.	No

Important: In earlier versions the <fsi:iterate> tag was used for the output of lists but it is no longer supported. However, it is possible to convert the <fsi:iterate> syntax into a <c:forEach> syntax.

[5] JavaServer Pages Standard Tag Library – For information on JSTL see http://wikipedia.org/wiki/JSTL

Example:

```
Old syntax:
<fsi:iterate list="movie()" resultName="m">
  <p>
    <fsi:get item="m.title"/>
  </p>
</fsi:iterate>


New syntax:
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

3.2.1.6 Search for data records (<fsi:search>)

The <fsi:search> tag can be used to search for data records. Alternatively, the query servlet can be used to execute queries or search for data records.

The search is based on the xml query defined with the <fsi:query> tag. The <fsi:query> tag must therefore be defined as the first tag within <fsi:search>. The result of the query is then available to other tags within the <fsi:search> tag.

The results list of the found elements can either be output completely (<c:forEach> - see Chapter 3.2.1.5) or page-based (<fsi:iterateResults> - see Chapter 3.2.1.8). Further information for the query can be determined using the <fsi:getQueryDetails> tag (see Chapter 3.2.1.9).

<fsi:query> does not have any mandatory parameters; however, the optional "pageSize", "resultName" and "schema" parameters can be used:

- **"pageSize" parameter:** The "pageSize" parameter is used to divide the elements of the results list between different (virtual) pages (cf. Chapter 3.2.4 Navigation, from page 36). The value of the parameter defines how may elements are displayed on each page.
- **"resultName" parameter:** The variable name for further use is defined by the "resultName" parameter
- **"schema" parameter:** It is also possible to use the "schema" parameter to define the

schema in which the search is to take place:

```
<fsi:search resultName="mL" schema="movies">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

The <fsi:setSchema> tag can also be used instead of the "schema" parameter:

```
<fsi:setSchema schema="movies">
  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        ${m.title}
      </p>
    </c:forEach>
  </fsi:search>
</fsi:setSchema>
```

Important: Use of the <fsi:setSchema> tag is recommended as use of the "schema" attribute is susceptible to errors (especially with nesting).

Attributes:

Attribute	Expected value	Mandatory parameter
pageSize	Number of elements to be displayed per page.	No
resultName	Variable name for further use within the tag.	No
schema	unique identifier of the schema from the module configuration	No

Example of complete output of all elements:

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

Example of page-based output of elements:

```
<fsi:search pageSize="5">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:iterateResults resultName="m">
    <p>
      ${m.title}
    </p>
  </fsi:iterateResults>
</fsi:search>
```

3.2.1.7 Create query (<fsi:query>)

The <fsi:query> tag contains a query which is used as the basis for the search within an <fsi:search> tag (cf. Chapter 3.2.1.6 page 24). Queries use the same xml syntax as the queries of a schema in the Template-Store and in the header function "contentSelect" (for further information see FirstSpirit Online Documentation[6]).

The <fsi:query> tag must be defined as the first tag within <fsi:search>. The result of the query is then available to other tags within the <fsi:search> tag.

Apart from the <QUERY> tag, no other tags are allowed within <fsi:query>.

[6] Area: .../vorlagenentwicklung/vorlagensyntax/funktionen/im_header/contentselect/contentselect.html

Example:

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie">
      <ORDER>
        <ORDERCRITERIA attribute="title" descending="1" />
      </ORDER>
    </QUERY>
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

3.2.1.8 Output of all elements of a page (<fsi:iterateResults>)

The <fsi:search> tag can be used to distribute the elements of a list between different (virtual) pages (cf. Chapter 3.2.1.6). While the <c:forEach> tag outputs or runs through all elements in the list (see Chapter 3.2.1.5), the <fsi:iterateResults> tag limits itself to the elements on the current page.

The tag has the mandatory parameter "resultName". The variable name for further use is defined with the "resultName" parameter.

The tag defines the "hasMore" variable which can be used to check within the tag whether more elements are to follow or not.

Note: When using <fsi:iterateResults> it must be noted that the list of the next higher <fsi:search> tag (innermost tag) is used as the basis:

Example:

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">

    <select name="<fsi:ref value='m.rating' />">

      <fsi:search>
        <fsi:query>
          <QUERY entityType="rating" />
        </fsi:query>
```

```
        <fsi:iterateResults resultName="r">
            ${r.abbreviation} (${r.name})
        </fsi:iterateResults>
      </fsi:search>

    </select>

  </c:forEach>
</fsi:search>
```

In the example the set of the "rating" table data records are used for <fsi:iterateResults>.

Attributes:

Attribute	Expected value	Mandatory parameter
resultName	Variable name for further use within the tag.	Yes

Variables:

Variable	Meaning	Return data type
hasMore	Returns whether the list contains further elements or not.	Boolean

3.2.1.9 Information on the query <fsi:getQueryDetails>

The <fsi:getQueryDetails> tag returns information on the current data record search (<fsi:search>).

The "pageNo", "pageSize", "parameter", "totalPages" and "totalResults" variables are available within the tag.

Variables:

Variable	Meaning	Return data type
pageNo	Number of the currently displayed (virtual) page.	Integer
pageSize	Number of elements to be displayed on the page.	Integer

parameter	Returns all search parameters.	Map
totalPages	Number of all (virtual) pages (basis of calculation: Number of data records and "pageSize")	Integer
totalResults	Number of all elements (over all pages).	Integer

Example:

```
<fsi:search>
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:getQueryDetails>
      <%= totalResults %> films found!
  </fsi:getQueryDetails>
</fsi:search>
```

3.2.2 Removing a query (<fsi:clearQuery>)

The <fsi:clearQuery> tag can be used to cancel the dividing up of the results between virtual pages (via the query servlet or <fsi:search>). The query results are then removed from the user's HTTP session and re-releases the memory required for the division between the pages.

The tag has the parameter "if". The parameter expects details of the request parameter name as the value. If this optional parameter is given the tag checks whether such a request parameter exists before removing the division.

```
<fsi:clearQuery if="resetNavigation" />
```

If the page is opened and if the request parameter "resetNavigation" exists when the call is made, the division is removed.

Attributes:

Attribute	Expected value	Mandatory parameter
if	Details of a request parameter name	No

3.2.3 Comparison operations

The following comparison operations are available:

- <c:if> If-Then expression (see Chapter 3.2.3.1 page 30).
- <c:choose> If-Then-Else expression (see Chapter 3.2.3.2 page 32).
- <fsi:contains> Contains expression (see Chapter 3.2.3.3 page 33).
- <fsi:matches> Matches expression (see Chapter 3.2.3.4 page 34).

3.2.3.1 If-Then expression (<c:if>)

The <c:if> tag of the JSTL[7] can be used to check a condition. If the condition applies ("if") the instruction is executed within the tag ("then"):

```
<c:if test="BEDINGUNG">
  BODY
</c:if>
```

The following comparison operators can be used in the condition:

```
== or eq
!= or ne
<  or lt
>  or gt
<= or le
>= or ge
```

If the check should be whether the content of a string objects "v" contains the character string "VALUE" the syntax is:

```
<c:if test="${v == 'VALUE'}">
  ...
</c:if>
```

Important: In older versions an if-Then expression was realised with the <fsi:if>, <fsi:equals> and <fsi:then> tags. These tags are no longer supported. However, the old syntax can be converted into the new syntax:

[7] JavaServer Pages Standard Tag Library – For information on JSTL see http://wikipedia.org/wiki/JSTL

Old syntax:

```
<fsi:if>
  <fsi:equals object1="user.login" value2="Admin" />
  <fsi:then> [X] </fsi:then>
</fsi:if>
```

New syntax:

```
<c:if test="${user.login == 'Admin'}"> [X] </c:if>
```

Further, the following logical operators can be used in the condition:

```
&& or and
|| or or
!  or not
```

Example of use of logical operators:

```
<c:if test="${user.login == 'Admin' && user.isActive}"> [X] </c:if>
```

Important: In the older version the <fsi:and> and <fsi:or> tags were available as logical operators. As the tags are no longer supported the expressions should be converted into the current syntax.

Old syntax:

```
<fsi:if>
  <fsi:and>
    <fsi:equals object1="user.login" value2="Admin" />
    <fsi:equals object1="user" object2="User(12)" />
  </fsi:and>
  <fsi:then> [X] </fsi:then>
</fsi:if>
```

New syntax:

```
<fsi:ref value="User(12)" resultName="u" />
<c:if test="${user.login == 'Admin' && user == u}"> [X] </c:if>
```

Important: The check of two objects or values for equivalence (in the older version) with the <fsi:equals> tag is no longer possible. The check for equivalence can be realised using the logical operator "==". To convert the old syntax into the new syntax, see the following examples:

Old syntax:

```
<fsi:if>
  <fsi:or>
      <fsi:equals object1="user.login" value2="Admin" />
      <fsi:equals object1="user" object2="User(12)" />
  </fsi:or>
  <fsi:then> [X] </fsi:then>
</fsi:if>
```

New syntax:

```
<fsi:ref value="User(12)" resultName="u" />
<c:if test="${user.login == 'Admin' || user == u}"> [X] </c:if>
```

3.2.3.2 If-Then-Else expression (<c:choose>)

The <c:choose> tag of the JSTL[8] can be used to realise an If-Then-Else expression. The <c:choose> tag does not have any parameters whatsoever.

It can contain:

- one or several <c:when> tags and
- no or one <c:otherwise> tag.

<c:when> has the mandatory parameter "test" with which a condition is a defined. If the condition applies (is true) the content of <c:when> is executed.

The <c:otherwise> tag must be defined as the last tag within <c:otherwise>. The content of the tag is executed if none of the given conditions of the <c:when> tag apply.

```
<c:choose>
  <c:when test="CONDITION_1">
    RUMPF_1
  </c:when>

  <c:when test="CONDITION_2">
    RUMPF_2
  </c:when>

  ...

  <c:otherwise>
```

[8] JavaServer Pages Standard Tag Library – For information on JSTL see http://wikipedia.org/wiki/JSTL

```
    RUMPF_N
  </c:otherwise>
</c:choose>
```

3.2.3.3 Contains expression (<fsi:contains>)

The JSP tag <fsi:contains> can be used to check whether a value or object is contained in a list.

The mandatory parameter "list" is used to handover a list to the tag for the comparison. The variable name for further use is to be given as the value for the mandatory parameter "resultName".

The alternative parameters "value" and "object" contain the value or object to be compared.

Important: The two parameters "value" and "object" cannot be given simultaneously.

In addition, the optional parameter "schema" can be used to define the schema for the comparison.

Important: Use of the <fsi:setSchema> tag is recommended (cf. Chapter 3.2.1.1 page 15) as use of the "schema" attribute is susceptible to errors (especially with nesting).

Example of the check whether a value is contained in a list:

```
<fsi:contains list='l'
              value='WERT'
              resultName="hasW" />
<c:if test="${hasW}">
  [X]
</c:if>
```

Example of the check whether an object is contained in a list:

```
<fsi:contains list='m.actors'
              object='person(93)'
              resultName="hasP93" />
<c:if test="${hasP93}">
  [X]
</c:if>
```

Attributes:

Attribute	Expected value	Mandatory parameter
resultName	Variable name for further use within the tag.	Yes
list	Object or attribute reference of the list, for example "m.actors".	Yes
object	Comparative object reference or comparative attribute reference, e.g. "user(12)".	Alternative parameter to "value"
value	Comparative value	Alternative parameter to "object"
schema	unique identifier of the schema from the module configuration	No

3.2.3.4 Matches expression (<fsi:matches>)

The JSP tag <fsi:matches> can be used to check whether the intersection of two lists is not empty.

The variable name for further use is defined with the mandatory "resultName" parameter.

Two alternative parameters exist for handover of the lists. The first parameter is denoted with the postfix "1" and the second with "2". The prefix "list" must be used to define an object reference to a list and the "values" prefix for a reference to a values list.

Example: Check whether two value lists intersect. The first list contains the values "a", "b" and "c" and the second list contains the values "b", "c" and "d". As the two lists are value lists the prefix "values" must be used for both. The postfix "1" is used for the first list and "2" for the second. The parameter for the first value list is therefore "values1" and for the second value list it is "values2".

The values of the list for the parameters "values1" and "values2" are given separated by commas. As several Java classes insert a "[" symbol or attach a "]" symbol when the .toString()

method is opened the list can also optionally be entered between these two symbols.

Example:

```
<fsi:matches values1="a,b,c"
             values2="[b,c,d]"
             resultName="hasIntersection" />
<c:if test="${hasIntersection}">
  [X]
</c:if>
```

It is also possible to use the "schema" parameter to define the schema in which the search is to take place:

```
<fsi:matches values1="a,b,c"
             values2="[b,c,d]"
             resultName="hasIntersection"
             schema="movies" />
<c:if test="${hasIntersection}">
  [X]
</c:if>
```

Attributes:

Attribute	Expected value	Mandatory parameter
resultName	Variable name for further use within the tag.	Yes
list1	Object or attribute reference of the list, e.g. "m.actors".	Alternative parameter to "value1"
value1	Comparative value	Alternative parameter to "list1"
list2	Object or attribute reference of the list, e.g. "m.actors".	Alternative parameter to "value2"
value2	Comparative value	Alternative parameter to "list2"
schema	Unique identifier of the schema from the module configuration.	No

3.2.4 Navigation

The <fsi:search> tag can be used to divide individual data records between virtual pages. FirstSpirit DynamicDatabaseAccess offers tags for creating a navigation bar to navigate between the individual virtual pages.

A navigation bar consists of three areas:

- First virtual page or top navigation limit (fsi:navTop)
- Sub-area or navigation section of virtual pages, starting from the current page (fsi:navFrame)
- Last virtual page or bottom navigation limit (fsi:navBottom)

Examples of navigation bars (the current page is highlighted in bold in each):

- Display the first and last virtual page and the sub-area:
  [1] … [6] [7] **[8]** [9] [10] … [21]
- Display the last virtual page and the sub-area
  (the first page is displayed in the sub-area):
  [1] **[2]** [3] [4] [5] … [21]
- Display the first virtual page and the sub-area
  (the last page is displayed in the sub-area):
  [1] … [17] [18] [19] **[20]** [21]
- Display the sub-area
  (the first and the last page are displayed in the sub-area):
  [1] [2] [3] [4] [5]

3.2.4.1 Navigation bar (<fsi:navigation>)

The <fsi:navigation> tag generates a navigation bar. The <fsi:navTop>, <fsi:navFrame> and <fsi:navBottom> tags can be used within the <fsi:navigation> tag.

The "frameSize" parameter can be used to define how many virtual pages are to be displayed in the sub-area (<fsi:navFrame). If the parameter is not given the default value "5" is used.

Attributes:

Attribute	Expected value	Mandatory parameter
frameSize	Number of virtual pages to be displayed in <fsi:navFrame> (integer); default value: „5“	No

Example:

```
<fsi:search pageSize="2">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:navigation frameSize="3">

    <fsi:navTop resultName="isVisible">
      <c:if test="${isVisible}">
        <a href="${navUrl}">[${navPageNo}]</a> ...
      </c:if>
    </fsi:navTop>

    <fsi:navFrame resultName="isVisible">
      <c:choose>
        <c:when test="${!isVisible}">
          <b>[${navPageNo}]</b>
        </c:when>
        <c:otherwise>
          <a href="${navUrl}">[${navPageNo}]</a>
        </c:otherwise>
      </c:choose>
    </fsi:navFrame>

    <fsi:navBottom resultName="isVisible">
      <c:if test="${isVisible}">
        ... <a href="${navUrl}">[${navPageNo}]</a>
      </c:if>
    </fsi:navBottom>

  </fsi:navigation>
  ...
</fsi:search>
```

3.2.4.2 First / last virtual page (<fsi:navTop>, <fsi:navBottom>)

The two JSP tags <fsi:navTop> and <fsi:navBottom> are available within the <fsi:navigation> tag.

The variable name for further use can be defined using the "resultName" parameter. This variable can be used to check whether the first (<fsi:navTop>) or last (<fsi:navBottom>) virtual page is not displayed within the sub-area (<fsi:navFrame>). The variable returns "true" if the first and/or last virtual page is not displayed in the sub-area. This can be used, for example, to prevent duplicate output if the first and/or last page is already displayed in the sub-area.

```
...
<fsi:navTop resultName="isVisible">
  <c:if test="${isVisible}">
    ...
  </c:if>
</fsi:navTop>
...
```

The "navPageNo" and "navUrl" variables are available within the two tags. "navPageNo" can be used to output the number of the virtual target page and "navUrl" to output its URL. In addition, the number of virtual pages can be output with "navPageSize"

Variables:

Variable	Meaning	Return data type
navPageNo	Number of the virtual target page	Integer
navUrl	URL of the virtual target page (incl. parameter)	string
navPageSize	Number of virtual pages	Integer

Example:

```
<fsi:search pageSize="2">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:navigation frameSize="3">

    <fsi:navTop resultName="isVisible">
      <c:if test="${isVisible}">
        <a href="${navUrl}">[${navPageNo}]</a> ...
      </c:if>
    </fsi:navTop>

    <fsi:navFrame resultName="isVisible">
      <c:choose>
        <c:when test="${!isVisible}">
          <b>[${navPageNo}]</b>
        </c:when>
        <c:otherwise>
          <a href="${navUrl}">[${navPageNo}]</a>
        </c:otherwise>
      </c:choose>
    </fsi:navFrame>

    <fsi:navBottom resultName="isVisible">
      <c:if test="${isVisible}">
        ... <a href="${navUrl}">[${navPageNo}]</a>
      </c:if>
    </fsi:navBottom>

  </fsi:navigation>

  ...

</fsi:search>
```

3.2.4.3 Sub-area of all virtual pages (<fsi:navFrame>)

The <fsi:navFrame> tag, which is available within the <fsi:navigation> tag, is used to display all virtual pages in the sub-area.

The number of pages to be displayed is defined by the "frameSize" attribute.

The variable name for further use can be defined using the "resultName" parameter. This variable can be used to check whether the page to be displayed in the sub-area does not correspond to the current page. The variable returns "true" if the page to be displayed by the sub-area is not the current page.

```
...
<fsi:navFrame resultName="isVisible">
  <c:if test="${isVisible}">
    ...
  </c:if>
</fsi:navFrame>
...
```

The "navPageNo" and "navUrl" variables are available within the tag. "navPageNo" can be used to output the number of the virtual page to be displayed by the sub-area and "navUrl" to output its URL. In addition, the number of virtual pages can be output with "navPageSize".

Variables:

Variable	Meaning	Return data type
navPageNo	Number of the virtual target page	Integer
navUrl	URL of the virtual target page (incl. parameter)	string
navPageSize	Number of virtual pages	Integer

Example:

```
<fsi:search pageSize="2">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:navigation frameSize="3">

    <fsi:navTop resultName="isVisible">
      <c:if test="${isVisible}">
        <a href="${navUrl}">[${navPageNo}]</a> ...
      </c:if>
    </fsi:navTop>

    <fsi:navFrame resultName="isVisible">
      <c:choose>
        <c:when test="${!isVisible}">
          <b>[${navPageNo}]</b>
        </c:when>
        <c:otherwise>
          <a href="${navUrl}">[${navPageNo}]</a>
        </c:otherwise>
      </c:choose>
    </fsi:navFrame>

    <fsi:navBottom resultName="isVisible">
      <c:if test="${isVisible}">
        ... <a href="${navUrl}">[${navPageNo}]</a>
      </c:if>
    </fsi:navBottom>

  </fsi:navigation>

  ...

</fsi:search>
```

3.3 Change / create data records (Store Servlet)

After configuring the FirstSpirit DynamicDatabaseAccess module a servlet is automatically available for saving data – the "Store Servlet".

The servlet can be used to create new data records or to change existing data records.

Important: No data can be saved if the configuration option "Use Release Status" is activated for the schema (see Chapter 2.3.1 page 9)!

The saving of values within the JSP pages via HTML input components or via JavaScript-based editors (e.g. TinyMCE, HTMLArea, FCK Editor, etc.) is usually realised in an HTML form. Each HTML input component has a parameter "name" and a "value". When it is saved the store servlet tries to save the transferred value on the basis of the "name" parameter.

In order for the store servlet to be able to assign the correct columns or data records in the database it requires a unique reference to the column and data record. The <fsi:ref> tag can be used to obtain a unique reference to an attribute or object (see Chapter 3.2.1.3).

Example:

```
<input type="text"
       name="<fsi:ref value='m.title' />"
       value="<c:out value='${m.title}'
                     escapeXml='true' />" />
```

To create a new data record, the key term "*" and the name of the schema's table are used as the basis for formation of the absolute object reference:

```
TABLE NAME(*)
```

The absolute reference, the required column name, is appended for the individual input fields, e.g.:

```
<input type="text"
       name="<fsi:ref value='movie(*).name' />"
       value="" />
```

The store servlets is opened with:
`"<%= application.getContextPath() %>/do.store".`

The <form> tag is used in an HTML form:

```
<form method="post"
      action="<%= application.getContextPath() %>/do.store">
  ...
</form>
```

Important: "Post" must be used for the HTTP transfer method as the length of the parameter names and values can be very long!

In addition the following request parameters are handled separately:

- "schema" (see Chapter 3.3.1)
- "url_ok" (see Chapter 3.3.2)
- "url_error" (see Chapter 3.3.2)
- "pw_params" (see Chapter 3.3.4)
- "url_pw_error“ (see Chapter 3.3.4)

3.3.1 Define a schema

The optional "schema" parameter can be used to define the name of a schema configuration to be used to save the data. The standard schema is used if the parameter is not given.

```
<input type="hidden" name="schema" value="movies" />
```

3.3.2 Handling a successful or faulty save

The "url_ok" parameter enables to define a URL to be displayed if the save is successful. The "url_error" parameter is used to define a URL to be displayed in the event of an error.

```
<input type="hidden"
       name="url_ok"
       value="$CMS_REF(pageref:"saved")$" />
<input type="hidden"
       name="url_error"
       value="$CMS_REF(pageref:"error")$" />
```

The behaviour in the event of an error can be adjusted using the key term "forward:" in the url_error parameter. If "forward:" is set in front of the URL a Forward is executed, otherwise a Redirect (default value):

```
<input type="hidden"
      name="url_error"
      value="forward:..." />
```

If an Exception is triggered when the data is saved, this is available after the error page is opened. The Exception can be accessed using the JSP expression `request.getAttribute("exception");`.

3.3.3 Formatting date and number fields

A format for parsing the input can be defined for date and number fields. To this end ".format" must be given behind the <fsi:ref> tag:

```
<input type="name" name="<fsi:ref value='movie(*).release_date' />"
value="01/01/70" size="50" maxlength="25" />
<input type="hidden" name="<fsi:ref value='movie(*).release_date' />.format"
value="MM/dd/yy" />
```

3.3.4 Saving passwords

The login password is frequently stored in the database for the management of user data. If the user changes it is sensible for the user to enter the new password twice (to intercept typing errors).

The store servlet offers the possibility of checking the double input of passwords. This is done using the "pw_params" parameter. The HTML input is inserted in the page twice, whereby the "name" attribute must have the same value for both input fields. The (identical) name of the two input fields ("name" attribute of the <input> tag) is given as the value for "pw_params":

```
<fsi:ref value="user(*).password" resultName="pwd" />

New password: <input type="text" name="${pwd}" value="" />
Repeat:   <input type="text" name="${pwd}" value="" />

<input type="hidden" name="pw_params" value="${pwd}" />
```

The "url_pw_error" parameter can be used to define a URL which is displayed if the two values do not match:

```
<input type="hidden"
       name="url_pw_error"
       value="$CMS_REF(pageref:"passwordmismatch")$" />
```

Request parameter:

Parameter	Expected value
schema	unique identifier of the schema from the module configuration
pw_params	identical name of the two input fields ("name" attribute of an <input> tag)
url_ok	URL (if save is successful)
url_error	URL (if save is faulty)
url_pw_error	URL (if values of the two password fields differ)

Simple example:

```
$CMS_IF(#global.preview)$
  <form method="post"
        action="<%=
          application.getContextPath()
        %>/do.store">
    <table>
      <tr>
        <td valign="top">*</td>
        <td valign="top">
          <input type="text"
                 name="<fsi:ref value='User(*).Name' />"
                 value="" />
        </td>
        <td>
          <input type="hidden"
                 name="url_ok"
                 value="$CMS_REF(#global.node)$" />
          <input type="submit"
                 value="Add" />
        </td>
      </tr>
    </table>
  </form>
$CMS_END_IF$
```

3.4 Delete data records (Delete Servlet)

The Delete Servlet can be used to delete data records. The servlet is available immediately following installation of FirstSpirit DynamicDatabaseAccess.

Important: No data can be deleted if the configuration option "Use Release" is activated for the schema (see Chapter 2.3.1 page 9)!

To delete a data record in the database the delete servlet requires a unique reference to the data record. The unique reference must be transferred to the delete servlet with the request parameter "delete". The <fsi:ref> tag can be used to obtain a unique reference to an attribute or object (see Chapter 3.2.1.3).

Example:

```
...
<c:forEach items="${list}" var="item">
  ...
  <input type="hidden"
         name="delete"
         value="<fsi:ref value='item' />" />
  ...
</c:forEach>
...
```

The delete servlets is opened with:

```
"<%= application.getContextPath() %>/do.delete"
```

The <form> tag is used in an HTML form:

```
<form method="post"
      action="<%= application.getContextPath() %>/do.delete">
  ...
</form>
```

Important: "Post" must be used for the HTTP transfer method as the length of the parameter names and values can be very long!

In addition the following request parameters are handled separately:

- "schema" (see Chapter 3.3.1)
- "url_ok" (see Chapter 3.3.2)
- "url_error" (see Chapter 3.3.2)

Request parameter:

Parameter	Expected value
delete	Unique reference to the data record to be deleted, e.g. `<fsi:ref value='item' />`
schema	unique identifier of the schema from the module configuration
url_ok	URL (if save is successful)
url_error	URL (if save is faulty)

Simple example:

```
<fsi:search resultName="list">

  <fsi:query>
    <QUERY entityType="User">
      <ORDER>
        <ORDERCRITERIA attribute="Name" descending="0" />
      </ORDER>
    </QUERY>
  </fsi:query>

  <table>
    <c:forEach items="${list}" var="item">
      <tr>
        <td valign="top">${item.fs_id}</td>
        <td valign="top">${item.Name}</td>

        $CMS_IF(#global.preview)$
          <form method="post"
                action="<%=
                  application.getContextPath()
                %>/do.delete">
            <td>
              <input type="hidden"
                     name="delete"
                     value="<fsi:ref value='item' />" />
              <input type="hidden"
                     name="url_ok"
                     value="$CMS_REF(#global.node)$" />
              <input type="submit"
                     value="Del" />
            </td>
          </form>
        $CMS_END_IF$

      </tr>
    </c:forEach>
```

```
  </table>
</fsi:search>
```

3.5 Query (Query Servlet)

In addition to the <fsi:search> and <fsi:query> tags the database queries can also be carried out using the Query Servlet. The result of such a query is saved in the user session and can be output with <fsi:iterateResults>.

The query servlet is opened with:

```
 "<%= application.getContextPath() %>/do.query"
```

The <form> tag is used in an HTML form:

```
<form method="post"
      action="<%= application.getContextPath() %>/do.query">
  ...
</form>
```

Important: "Post" must be used for the HTTP transfer method as the length of the parameter names and values can be very long!

Analog to the <fsi:search> tag ("pageSize" parameter) the Query Servlet also has a parameter to divide data records between different virtual pages. This request parameter is called "page_size". The value expected by the parameter is how many data records to be displayed on a virtual page. If the parameter is not given the default value 10 (data records per virtual page) is used.

```
<input type="hidden" name="page_size" value="1" />
```

The following tags are available for displaying a navigation bar (for navigation through the individual virtual pages):

- <fsi:navigation> (see Chapter 3.2.4.1 page 36).
- <fsi:navTop> (see Chapter 3.2.4.2 page 38).
- <fsi:navFrame> (see Chapter 3.2.4.3 page 40).
- <fsi:navBottom> (see Chapter 3.2.4.2 page 38).

The request parameter "query_xml" is used to hand over the XML of a query to the query servlet. The syntax of the XML is identical to the syntax which can be used within the header function "contentSelect" in templates (for further information see FirstSpirit Online Documentation[9]):

```
<input type="hidden" name="query_xml" value="&lt;QUERY
entityType=&quot;person&quot;&gt;&lt;FILTERPARAM parameter=&quot;person&quot;
datatype=&quot;java.lang.String&quot; value=&quot;%&quot; /&gt;&lt;LIKE
attribute=&quot;surname&quot; parameter=&quot;person&quot;
/&gt;&lt;/QUERY&gt;" />
```

Important: To avoid problems with the HTML <input> tag, the query is to be quoted in the XML or HTML symbols "<", ">" and "“", i.3. "<", ">" and """!

If parameters are used in an XML query (<FILTERPARAM> tag), the individual values of the parameters can be handed over to the query servlet. The syntax is made up of the key term "param." and the value of the "parameter" parameter of the <FILTERPARAM> tag:

```
<input type="hidden" name="query_xml" value="&lt;QUERY
entityType=&quot;person&quot;&gt;&lt;FILTERPARAM parameter=&quot;person&quot;
datatype=&quot;java.lang.String&quot; value=&quot;%&quot; /&gt;&lt;LIKE
attribute=&quot;surname&quot; parameter=&quot;person&quot;
/&gt;&lt;/QUERY&gt;" />

<input type="text" name="param.person" value="" />
```

In addition the following request parameters are handled separately:

- "schema" (see Chapter 3.3.1)
- "url_ok" (see Chapter 3.3.2)
- "url_error" (see Chapter 3.3.2)

Request parameter:

Parameter	Expected value
query_xml	XML query, e.g.
	"<QUERY entityType="person"><FILTERPARAM parameter="person" datatype="java.lang.String" value="%" /><LIKE attribute="surname" parameter="person" /></QUERY>"

[9] Area: .../vorlagenentwicklung/vorlagensyntax/funktionen/im_header/contentselect/contentselect.html

page_size	Number of elements to be displayed per page; default value: „10“
schema	unique identifier of the schema from the module configuration
url_ok	URL (if save is successful)
url_error	URL (if save is faulty)

Simple example:

```
<fsi:iterateResults resultName="p">
  <p>
    ${p.surname}, ${p.name}
  </p>
</fsi:iterateResults>
...
<form action="<%= application.getContextPath() %>/do.query"
      method="post">

  <input type="hidden"
         name="query_xml" value="&lt;QUERY
entityType=&quot;person&quot;&gt;&lt;FILTERPARAM parameter=&quot;person&quot;
datatype=&quot;java.lang.String&quot; value=&quot;%&quot; /&gt;&lt;LIKE
attribute=&quot;surname&quot; parameter=&quot;person&quot;
/&gt;&lt;/QUERY&gt;" />

  <input type="text"
         name="param.person"
         value="" />

  <input type="hidden"
         name="page_size"
         value="1" />

  <input type="hidden"
         name="schema"
         value="movies" />

  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />

  <input type="submit"
         value="Submit" />
</form>
```

4 Legal notices

The module "FirstSpirit DynamicDatabaseAccess" is a product of the e-Spirit AG, Dortmund, Germany.

When using this module only the licence agreed between the e-Spirit AG and the user is valid.

You can find information about third-party software which is potentially used for the module but not produced by the e-Spirit AG, their own licences and - as the case may be - information about updates on the start page of each FirstSpirit server in the area "Legal notices".