

HDPagination Reference Document

Version 1.2

PREFACE.....	4
CHAPTER 1. INTRODUCTION TO HDPAGINATION.....	4
1.1. ABOUT HDPAGINATION.....	4
1.2. JSF (JAVASERVER FACES) OR NON-JSF.....	5
CHAPTER 2. WHY HDPAGINATION	6
2.1. PAGINATION IS COMPLEX WITHOUT FRAMEWORK.....	6
2.2. HDPAGINATION MAKES PAGINATION EASY.....	7
2.2.1. <i>Non-JSF application:</i>	8
2.2.2. <i>JSF application</i>	10
CHAPTER 3. ACTION.....	10
3.1. ABSTRACT ACTION CLASS.....	10
3.2. CONFIGURE ACTION.....	12
3.2.1. <i>Properties</i>	12
3.3. MAP REQUEST TO PAGINATIONACTION.....	15
CHAPTER 4. QUERYCALLBACK.....	16
4.1. JDBCQUERYCALLBACK.....	17
4.1.1. <i>AbstractJdbcQueryCallback</i>	18
4.1.2. <i>AbstractColumnMapJdbcQueryCallback</i>	18
4.1.3. <i>AbstractNoParamBindJdbcQueryCallback</i>	19
4.1.4. <i>DefaultJdbcQueryCallback</i>	20
4.2. JPAQUERYCALLBACK.....	20
4.2.1. <i>AbstractJPAQueryCallback</i>	20
4.2.2. <i>NotProcessResultJPAQueryCallback</i>	21
4.2.3. <i>NoParamBindJPAQueryCallback</i>	22
4.2.4. <i>DefaultJPAQueryCallback</i>	22
4.3. HBN3QUERYCALLBACK.....	23
4.3.1. <i>AbstractHBN3QueryCallback</i>	23
4.3.2. <i>NotProcessResultHBN3QueryCallback</i>	24
4.3.3. <i>NoParamBindHBN3QueryCallback</i>	24
4.3.4. <i>DefaultHBN3QueryCallback</i>	25
4.4. HBN2QUERYCALLBACK.....	25
4.5. SET YOUR OWN QUERY STATEMENT TO COUNT RECORDS.....	25
CHAPTER 5. QUERYTEMPLATE.....	26
5.1. JPAQUERYTEMPLATE.....	26
5.2. HIBERNATE3QUERYTEMPLATE.....	27
5.3. HIBERNATE2QUERYTEMPLATE.....	27
5.4. JDBCQUERYTEMPLATE.....	28
5.4.1. <i>HSQLDBJdbcQueryTemplate</i>	29
5.4.2. <i>OracleJdbcQueryTemplate</i>	29
5.4.3. <i>DB2JdbcQueryTemplate</i>	29
5.4.4. <i>MySqlJdbcQueryTemplate</i>	30
5.4.5. <i>SQLServer2005JdbcQueryTemplate</i>	30
5.4.6. <i>SQLAnywhereJdbcQueryTemplate</i>	30
5.4.7. <i>PostgreSQL</i>	31
5.4.8. <i>StandardJdbcQueryTemplate</i>	31
CHAPTER 6. VALIDATOR.....	31
CHAPTER 7. GLOBALCONFIG.....	33

7.1. PROPERTIES.....	33
7.1.1. <i>pageSize</i>	33
7.1.2. <i>defaultDataScope</i>	33
7.1.3. <i>pageLinksPattern</i>	33
7.1.4. <i>sortUpImage & sortDownImage</i>	33
7.1.5. <i>urlSuffix</i>	34
7.1.6. <i>errorMessageBundle</i>	34
7.1.7. <i>view</i>	34
7.2. CONFIGURE GLOBALCONFIG IN SPRING.....	35
CHAPTER 8. CHANGE PAGE SIZE FROM PAGE UI.....	35
CHAPTER 9. TAGS.....	36
9.1. <HDP:PAGELINKS>.....	36
9.1.1. <i>Description</i>	36
9.1.2. <i>Attributes</i>	36
9.1.3. <i>How to use</i>	36
9.1.4. <i>Configure PageLinksPattern</i>	36
9.2. <HDP:SORTBY>.....	39
9.2.1. <i>Description</i>	39
9.2.2. <i>Attributes</i>	40
9.2.3. <i>How to use</i>	40
9.2.4. <i>sortUpImage & sortDownImage</i>	40
9.3. <HDP:SELECTSORTBY>.....	41
9.3.1. <i>Description</i>	41
9.3.2. <i>Attributes</i>	41
9.3.3. <i>How to use</i>	41
9.3.4. <i>Support for internationalization and localization</i>	42
9.4. <HDP:PAGESDROPDOWN>.....	42
9.4.1. <i>Description</i>	42
9.4.2. <i>Attributes</i>	42
9.4.3. <i>How to use</i>	43
9.5. <HDP:GOTOPAGE>.....	43
9.5.1. <i>Description</i>	43
9.5.2. <i>Attributes</i>	43
9.5.3. <i>How to use</i>	43
9.5.4. <i>Support for internationalization and localization</i>	44
9.6. <HDP:CURRENTPAGE>.....	44
9.6.1. <i>Description</i>	44
9.6.2. <i>Attributes</i>	44
9.7. <HDP:TOTALPAGES>.....	44
9.7.1. <i>Description</i>	44
9.7.2. <i>Attributes</i>	44
9.8. <HDP:RECORDSCOPE>.....	44
9.8.1. <i>Description</i>	44
9.8.2. <i>Attributes</i>	45
9.9. <HDP:TOTALRECORDS>.....	45
9.9.1. <i>Description</i>	45
9.9.2. <i>Attributes</i>	45
9.10. <HDP:PAGINATIONREQUIRED>.....	45
9.10.1. <i>Description</i>	45
9.10.2. <i>Attributes</i>	45
9.10.3. <i>How to use</i>	45
CHAPTER 10. HOW TO DEPLOY.....	46
10.1. <i>REQUIRED LIBRARIES</i>	46
10.2. <i>CONFIGURATION FILE</i>	46
10.3. <i>WEB.XML</i>	46
CHAPTER 11. INTEGRATE WITH TILES VIEW TECHNOLOGIES.....	47

11.1. INTEGRATE WITH TILES 2	47
11.1.1. Still use the default view (<i>InternalResourceView</i>).....	47
11.1.2. Use <i>Tiles2View</i>	48
11.2. INTEGRATE WITH TILES 1.x (A.K.A. "STRUTS TILES").....	49
11.2.1. Still use the default view (<i>InternalResourceView</i>).....	49
11.2.2. Use <i>TilesView</i>	50
CHAPTER 12. CLEAN SEARCH RESULT DATA.....	50
CHAPTER 13. WORK WITH JSF.....	50
DECLARING <i>PAGINATEDEFINITION</i> INSTANCE.....	50
13.1. DECLARING <i>PAGINATEDEFINITION</i> INSTANCE.....	51
13.1.1. <i>queryTemplate</i> (mandatory).....	51
13.1.2. <i>forwardPath</i> (optional).....	51
13.1.3. <i>attributeName4Data</i> (optional).....	52
13.1.4. <i>pageSize</i> (optional).....	52
13.1.5. <i>dataScope</i> (optional).....	52
13.1.6. <i>resultHandler</i> (optional).....	52
13.2. CREATING SUBCLASS OF <i>PAGINATEFACESACTION</i>	53
13.3. DECLARING SUBCLASS OF <i>PAGINATEFACESACTION</i> IN <i>FACES-CONFIG.XML</i>	54
13.4. MODIFYING WEB DEPLOYMENT DESCRIPTOR (<i>WEB.XML</i>).....	55
13.5. CREATE PAGE (JSP OR FACELET).....	56
13.5.1. Include <i>HDPagination JSF tags</i> definition.....	56
13.5.2. Bind action properties to <i>HTML</i> fields.....	56
13.5.3. Invoke action method of <i>PaginateFacesAction</i>	57
13.5.4. Display search result in table.....	57
13.5.5. Use <i>JSF tags</i>	58
13.6. CHANGE PAGE SIZE FROM PAGE UI.....	58
CHAPTER 14. PAGINATION TAGS FOR JSF.....	59
14.1. <i><HDP:PAGINATELINK></i>	59
14.1.1. Description.....	59
14.1.2. Attributes.....	59
14.1.3. How to use.....	59
14.1.4. Configure <i>PageLinksPattern</i>	60
14.2. <i><HDP:SORTLINK></i>	60
14.2.1. Description.....	60
14.2.2. Attributes.....	60
14.2.3. How to use.....	60
14.2.4. Support for internationalization and localization.....	61
14.3. <i><HDP:SELECTSORT></i>	61
14.3.1. Description.....	61
14.3.2. Attributes.....	62
14.3.3. How to use.....	62
14.3.4. Support for internationalization and localization.....	62
14.4. <i><HDP:SELECTPAGE></i>	62
14.4.1. Description.....	62
14.4.2. Attributes.....	63
14.4.3. How to use.....	63
14.5. <i><HDP:GOTOPAGE></i>	63
14.5.1. Description.....	63
14.5.2. Attributes.....	63
14.5.3. How to use.....	63
14.5.4. Support for internationalization and localization.....	64
CHAPTER 15. PORTLET SUPPORT.....	64
CHAPTER 16. SUPPORTED JDK/J2EE/JSF VERSION.....	64
16.1. JDK VERSION.....	64
16.2. J2EE VERSION.....	65
16.3. JSF VERSION.....	65

Preface

Pagination is a key part in web system design both from UI perspective (it makes UI friendly to user) and performance perspective in the server side (it makes server to service request more rapidly).

Today in java world you can find many frameworks, components, or libraries to deal with the pagination issue in J2EE applications. By leveraging these kinds of common components developers can save a lot of effort to implement the pagination functionalities in their applications, however almost all of these solutions adopt the same strategy - that is, fetching all records that satisfies the input search criteria from database and storing them in memory (typically store them as an attribute of `HttpServletSession` in J2EE Web application), then the built-in UI components (e.g. Tag or `UIComment` in JSF) choose one page of data to display. When users navigate through different pages, the UI components only navigates though the records saved in `session` to pick out part of data for that page, without hitting database again.

This kind of solution works very well if the search result is small, however when data becomes big it can cause a big performance problem because saving huge amount of data in `session` will cost a lot of memory.

For large data search scenario, the idea pagination solution is to load the data of one particular page only from database. And when user navigates to another page (e.g. by clicking "Previous" or "Next" page links), the system should "remember" the previous search criteria and use it together with the passed page number to query against database to load that particular page of data. We call this strategy as "lazy loading" or "loading required only".

Chapter 1. Introduction to HDPagination

1.1. About HDPagination

HDPagination stands for High Data Pagination. It is a framework (or library) to provide an overall solution for pagination search in J2EE with "loading required only" strategy I mentioned above, which can avoid the performance problem faced by the majority of other pagination solutions.

HDPagination relies on spring IoC (Inversion of Control) framework to manage its components (because spring provides such an excellent injection framework to assemble objects that I don't bother to reinvent the wheel, just use it directly). However it is MVC framework independent - that is, it's not tied with any particular MVC frameworks. It can be used in any J2EE Web application whether it's using Struts, Spring-MVC, or other MVC frameworks (or it does not use any web

framework and just uses JSP and normal java classes). The reason lies in the fact that it has its own Servlet class to handle and dispatch http request. Finally, there is one thing worth noting here. As a developer, you only need very basic knowledge of dependency injection about spring framework in order to use and configure HDPagination.

1.2. JSF (JavaServer Faces) or Non-JSF

HDPagination was designed to cater for both JSF (JavaServer Faces) and Non-JSF applications. Non-JSF application means application:

- Using traditional MVC frameworks (e.g. Struts, Spring-MVC etc) which are using JSP & Servlet API to handle HTTP Request and Response.
- Or that directly uses JSP and Servlets technologies (without any MVC framework on top of it), although this is very rare in now days.

JSF, on the other hand, is a component-based, event-driven MVC framework and has its unique lifecycle to handle Request and render Response. Because of this, HDPagination provides different components and tags for these 2 types of applications. **It's very important to choose the right tag library to use in your application. The tag libraries for Non-JSF (called JSP version) and JSF version have similar or same tag names, but different URIs:**

- JSP: <http://www.hdpagination.org/tags>
- JSF: <http://www.hdpagination.org/tags/jsf>

So, when you add the taglib directive on top of JSP files, make sure the correct URI is specified. Otherwise you will get unexpected behaviours.

The Servlet (`org.hdpagination.web.servlet.PaginationServlet`) mentioned in the above [section 1.1](#) is only used by JSP version. So if your application is JSF based, it should not be declared in the web.xml file. So do `PaginationAction` ([Chapter 3 Action](#)) and `org.hdpagination.web.validate.Validator` ([Chapter 6 Validator](#)), they are only used by JSP version.

Although the front-end components used by JSP and JSF versions are different, they share the same back-end components ([Chapter 4 Query Callback](#) and [Chapter 5 Query Template](#)) and some common components ([Chapter 7 GlobalConfig](#)). For example, how to construct an instance of `QueryCallback` and use different types of `QueryTemplate`, according to different data access technologies (JDBC, Hibernate, or JPA) used, are the same for both JSF and Non-JSF version.

So if you are developing a JSF application, unless you want to have an overview of the whole library, you can skip the unrelated chapters and jump to the sections related.

Chapter 2. Why HDPagination

2.1. *Pagination is complex without framework*

To implement the whole functionality of a pagination search in “loading required only” strategy mentioned above, it may include some steps:

Step1. SQL Conversion

The original SQL (which does not take pagination into account) should be converted into a pagination featured SQL so that it only retrieves one particular page of records from database. Different databases have different syntax to support pagination query. For example, if you want to search products from database and the original SQL is:

```
select productId, prodNo, price, madeIn, description from product p where madeIn = ?
and description = ?
```

In oracle the converted pagination SQL would be:

```
select * from (select a.*,rownum myrownum from (
    select productId,prodNo,price,madeIn,description from product p where madeIn
    = ? and description = ?
) a where rownum <= :endRecord ) b where b.myrownum > :startRecord
```

However, in MySQL database, it would be:

```
select productId, prodNo, price, madeIn, description from product p where madeIn = ?
and description = ? limit :startRecord, :pageSize
```

As you can see, each database has different way to support pagination feature for query (some database does not support at all), the developer needs to be aware of this.

Step2. Count total records

After executing the SQL generated in step1, the number of total records for the current search should be counted. Based on above original SQL, the count SQL would be:

```
select count(*) from product p where madeIn = ? and description = ?
```

Step3. Save result data and pagination meta-data

The result data of running the SQL generated in step1 is stored in request so that it can be displayed in the page. The pagination meta-data is stored in session and it generally includes the following information:

- (a) Total records of the search (by running count SQL generated in step 2)
- (b) Current page number
- (c) SQL context, including SQL statement as well as parameters binding. It is saved in session so that the subsequent requests for other pages can restore it to execute the same SQL statement but with different page number.
- (d) Sorting information, including the column by which the data is sorted, whether in ascending or descending order.

Step4. Display in page

Display the result data in a table using JSP code, JSTL or other tags.

Display page navigation links (such as “Previous”, “Next” page) to allow user to access one particular page of data.

Display other helpful information to allow user to easily identify the location (such as current page number, total pages, records scope of current page).

Apply a clickable link on column header (title) so that user can sort data result by that column.

As you can see, it’s not a simple and straight forward job to implement a pagination search function without any common library or framework. So much tedious work has to be done behind the screen. As a matter of fact, the majority of code is boilerplate code, such as translation of pagination SQL, opening/closing data source connection, counting total records, saving search result, generating and storing pagination meta-data, and displaying page links in pages based on saved pagination meta-data.

HDPagination (High Data Pagination) framework is designed to handle this boilerplate, allowing you to focus on writing query statement and binding its parameters with received input values based on specified business rules. It not only supports JDBC SQL query, but also supports query of some popular ORM tools such as JPA and Hibernate. More details will be discussed in later chapters.

2.2. HDPagination makes pagination easy

Now let’s look at how HDPagination makes it easy to implement pagination search functionality. As I mentioned in previous chapter, the ways to use HDPagination in a JSF and Non-JSF application are different, I am going to go through the steps for both cases separately. Let’s start the Non-JSF application first.

2.2.1. Non-JSF application:

Step1. Create Action

Create an action class which represents the component processing the pagination based search request. As you can see from the below code, you only deal with normal SQL without any database provider specific pagination SQL syntax included. You just concentrate on creating SQL, binding parameters with received input values and mapping each row of ResultSet into an object.

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" select
        productId,prodNo,price,madeIn,description from product p where 1=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append("?");
        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(",");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and description = ? ");
    }

    AbstractJdbcQueryCallback callback = new
        AbstractJdbcQueryCallback(queryStmt.toString()) {
        public void setValues(PreparedStatement ps) throws SQLException {
            int pos = 1;
            for (int i=0;i<madeInSelect.length;i++) {
                ps.setString(pos++, madeInSelect[i]);
            }

            if (desc != null && !desc.trim().equals("")) {
                ps.setString(pos++, desc);
            }
        }

        public Object processRow(ResultSet rs) throws SQLException {
            ProductVO prod = new ProductVO();
            prod.setProductId(new Long(rs.getLong("productId")));
            prod.setPrice(new Integer(rs.getInt("price")));
            prod.setProdNo(rs.getString("prodNo"));
            prod.setMadeIn(rs.getString("madeIn"));
            prod.setDescription(rs.getString("description"));
            return prod;
        }
    };
    return callback;
}
```

Step 2 Configure Action

Declare the action class and other configurations in a spring configure file ‘WEB-INF/hdpagination.xml’. In this case, the database is MySQL, as you can see `org.hdpagination.dataaccess.jdbc.mysql.MySqlJdbcQueryTemplate` is injected into the action class. This gives framework the indication that pagination SQL conversion

should be based on MySQL syntax. However the framework abstracts the detail of SQL conversion from the developer who uses the framework.

```
<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
  <property name="pageSize" value="10"/>
  <property name="defaultDataScope" value="request"/>
  <property name="urlSuffix" value="hdp"/>
  <property name="sortUpImage" value="/images/pagination_up.gif"/>
  <property name="sortDownImage" value="/images/pagination_down.gif"/>

  <property name="pageLinksPattern">
    <ref bean="defaultLinksPattern"/>
  </property>
</bean>

<bean id="defaultLinksPattern" class="org.hdpagination.config.pattern.DefaultPageLinksPattern">
  <constructor-arg type="int" value="10"/>
  <property name="resourceBundleName" value="eg.web.action.PaginalLinks"/>
</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.gjt.mm.mysql.Driver"/>
  <property name="url" value="jdbc:mysql://localhost:3306/test"/>
  <property name="username" value="" />
  <property name="password" value="" />
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.jdbc.mysql.MySqlJdbcQueryTemplate">
  <constructor-arg><ref bean="dataSource" /></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  <property name="id" value="searchProduct" />
  <property name="forwardPath" value="/searchProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>
```

Step3 Modify web.xml

Add the following to WEB-INF/web.xml file:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/hdpagination.xml</param-value>
</context-param>

<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

<servlet>
  <servlet-name>hdpagination</servlet-name>
  <servlet-class>org.hdpagination.web.servlet.PaginationServlet</servlet-class>
  <load-on-startup>3</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>hdpagination</servlet-name>
  <url-pattern>*.hdp</url-pattern>
</servlet-mapping>
```

This allows spring to load the configure file to initialize the related components, and declares the `PaginationServlet` will handle all http requests with URL ending with `‘.hdp’` and dispatch them to specific Action classes for processing.

Step4 Use tags in JSP

To simplify coding in JSP page, HDPagination provides some handy tags to display page links and other html elements (dropdown, text input) allowing user to navigate through different pages. All tags are very easy to use. For instance, to display page links, the code can be:

```
<hdp:pageLinks actionId="searchProduct" resendParams="true" />
```

To display a dropdown for all available pages, the code would be:

```
<hdp:pagesDropdown actionId="searchProduct" resendParams="true" />
```

Selecting the page option in the dropdown list will forward to that particular page.

An important thing worth noting here is that no tag is provided to display the table of search result data. This gives page developer the maximum flexibility as he can choose any technology (e.g. JSP code, JSTL or struts tags) to display the data, mixing with any html tags.

Details will be discussed in later chapters.

2.2.2. JSF application

[Chapter 13](#) describes how to use HDPagination in JSF applications.

Chapter 3. Action

Note: Action is only for Non-JSF application.

Action represents the component processing the pagination based search request. All custom action classes must implement the interface `org.hdpagination.web.action.PaginationAction` directly or indirectly. Each action must have a unique value of its property `‘id’`.

3.1. Abstract action class

It's highly recommended that custom action class extends from abstract class `org.hdpagination.web.action.AbstractPaginationAction` OR `org.hdpagination.web.action.AbstractCommandPaginationAction` rather than implements `PaginationAction` interface directly. If there is no java bean instance (It is called command) that is mapped to the search input form in the page and you want to get input values directly from parameters of `HttpServletRequest` instance directly, and then extend your custom action class from `AbstractPaginationAction`. In this case, you have to implement the method `getQueryCallback (HttpServletRequest request)`. However if there is a java bean instance (It is called command) mapped to the form in the page and you want to get input values from this java bean rather than `HttpServletRequest` parameters, then choose `AbstractCommandPaginationAction` as the parent class of your custom action. And accordingly, another method `getQueryCallback (HttpServletRequest request, Object command)` with 2 arguments should be implemented instead.

For `AbstractPaginationAction` class, there is nothing special and you just invoke method `HttpServletRequest.getParameter (java.lang.String name)` to get input values and perform business logic.

For `AbstractCommandPaginationAction`, however, you have to define the class type of command used either in construct or via spring configuration file. The sample code of declaring class type in construct is as follows:

```
public class SearchProductAction extends AbstractCommandPaginationAction {  
  
    public SearchProductAction() {  
        super(SearchCriteria.class);  
    }  
  
    ...  
    ...  
  
}
```

The command class `SearchCriteria` is defined as follows:

```
public class SearchCriteria {  
    private String[] madeIn;  
    private String description;  
  
    public String[] getMadeIn() {  
        return madeIn;  
    }  
  
    public void setMadeIn(String[] madeIn) {  
        this.madeIn = madeIn;  
    }  
  
    public String getDescription() {  
        return description;  
    }  
  
    public void setDescription(String description) {  
        this.description = description;  
    }  
  
}
```

The idea of introducing `command` is similar to that of `ActionForm` in Struts – that is, values of `command` (javabean) properties are populated with the values of request parameters by the framework automatically when the property name is matched with the parameter name. Very importantly, `HDPagination` only tries to match the properties with type of `java.lang.String` or `java.lang.String[]`. For any property not matched with any request parameter or with different type to `java.lang.String` and `java.lang.String[]`, its value will not be populated.

```
<form name="searchProdForm" action="/searchProduct.hdp" method="post">
  Made In:
  <select name="madeIn" multiple="multiple">
    <option value="China">China</option>
    <option value="USA">USA</option>
    <option value="Australia">Australia</option>
  </select>

  Description:
  <input type="text" name="description" />

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

Take the above JSP as an example; the name of the HTML select field is matched with the property ‘madeIn’ in the command javabean `SearchCriteria`. Since it’s multiple selections, it makes sense to declare the type of property as `java.lang.String[]`. The other HTML text input field is matched with the property ‘description’ and its type is `java.lang.String`. When form is submitted, the command object will be populated with proper property values from the input of the page.

3.2. Configure action

To configure action in spring configure file is as simple as:

```
<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  <property name="id" value="searchProduct" />
  <property name="forwardPath" value="/searchProduct.jsp" />
  <property name="attributeName4Data" value="query_result" />

  <property name="queryTemplate">
    <ref bean="jdbcQueryTemplate" />
  </property>
</bean>
```

Action should be configured as singleton (that’s the default in spring). In fact, all components used by `HDPagination` should be configured as singleton.

3.2.1. Properties

3.2.1.1. id

The value of 'id' property should be unique for each action instance declared in spring application context. The `PaginationServlet` relies on this property to match the action instance with incoming request URL (`ServletPath`). And `Pagination Tags` internally use it to retrieve corresponding pagination information (such as "number of total records", "page number of current page") from `HttpSession` which is saved the first time the query was executed.

It's worth noting that the 'id' property of action is different to the 'id' attribute of the bean element. The 'id' attribute of the bean element is only used by spring framework to manage the bean instances, and `HDPagination` does not use this. **It's the 'id' property of action that `HDPagination` relies on to do all the pagination work.**

This is the 'id' attribute of bean element. It is not used by `HDPagination`.

This is the 'id' property of action. `HDPagination` is based on its value to do the job.

```
<bean id="searchProductAction" class="web.OracleJdbcSearchProductAction">
  <property name="id" value="searchProduct"/>
  <property name="forwardPath" value="/searchProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="jdbcQueryTemplate"/>
  </property>
</bean>
```

3.2.1.2. forwardPath

Property 'forwardPath' specifies the URL path to which it will forward request after execution of query. However, if the **view** used in Action (It relies on the 'view' property of Action OR `GlobalConfig`) is `TilesView` OR `Tiles2View`, it is interpreted as name of Tiles definition. For more information about view, see the section entitled [3.2.1.9](#) and the section entitled [7.1.7](#).

3.2.1.3. attributeName4Data

Property 'attributeName4Data' specifies the name of attribute which holds the search result in the specified scope (e.g. request or session). In this example, the search result (`java.util.List`) is saved as an attribute '**query_result**' in request or session, so you can use `jstl` tag to display the data as follows:

```

<c:forEach var="each" items="\${query_result}">
  <tr>
    <td><div align="center"><c:out value="\${each.prodNo}"/></div></td>
    <td><div align="center"><c:out value="\${each.price}"/></div></td>
    <td><div align="center"><c:out value="\${each.madeIn}"/></div></td>
    <td><div align="center"><c:out value="\${each.description}"/></div></td>
  </tr>
</c:forEach>

```

There are some other optional properties you can use to override the default setting in `GlobalConfig`. The settings in the action level are only valid for the searching function (or page) related to that action.

3.2.1.4. pageSize

It specifies how many records should be displayed in one page. Property 'pageSize' of `GlobalConfig` defines the default value for all pages (see Chapter 6). However if you want the page related to this action to have different number of records to display, this is the place you can set.

3.2.1.5. inputPath

It defines the URL of input page. Like the property '*forwardPath*', if the **view** used in Action (It relies on the '*view*' property of Action or `GlobalConfig`) is `TilesView` or `Tiles2View`, it is interpreted as name of Tiles definition corresponding to the input page. For more information about view, see the section entitled [3.2.1.9](#) or the section entitled [7.1.7](#).

It's used for validation only. If validation fails it will forward back to input URL. Its default value is the same as property '*forwardPath*'. In majority cases, you don't need to set this property.

3.2.1.6. dataScope

Property 'defaultDataScope' of `GlobalConfig` define the default scope ('request', 'session' or 'application') where the search result is to be stored. You can use this property to override the default setting in `GlobalConfig`. Again, this setting is only valid for the searching function (or page) related to the action.

3.2.1.7. searchResultHandler

Plug in custom `searchResultHandler` class to handle search result.

3.2.1.8. validator

Plug in custom `validator` class to do input validation (see [Chapter 6](#)).

3.2.1.9. view

View represents a component which is responsible for rendering page after searching finished. The default view implementation is `InternalResourceView` which just forwards the request to a JSP or other resource URL within the same web application using `RequestDispatcher`. The default view is often sufficient for majority of web applications, and in this case you don't have to configure this property explicitly. However, if you are using Tiles to manage your page layout, you can choose one of other 2 implementations:

TilesView

If your application uses Tiles 1.x (a.k.a. "Struts Tiles") as page template and you want `HPagination` to render a definition rather than forward to a JSP (or other internal resource) directly, then an instance of `TilesView` should be configured to this property. For more information, see the section entitled [11.2 Integrate with Tiles 1.x \(a.k.a. "Struts Tiles"\)](#).

Tiles2View

If your application uses Tile 2 as page template and you want `HPagination` to render a definition rather than forward to a JSP (or other internal resource) directly, then an instance of `Tiles2View` should be configured. For more information, see the section entitled [11.1 Integrate with Tiles 2](#).

It should be noted that if "view" properties in both Action and GlobalConfig are set, the "view" property in Action will take precedence (and the "view" property in GlobalConfig will not apply). For instance, if GlobalConfig has "view" property of Tiles2View and the action is set with InternalResourceView, then this action will use InternalResourceView to render page and other actions use Tiles2View (unless some of actions are set with other view implementations). For more information about "view" property in GlobalConfig, see the section entitled [7.1.7](#).

3.3. Map request to PaginationAction

The URL path from the search request must end with suffix specified by `'urlSuffix'` property of `GlobalConfig`. For instance, if suffix is "hdp" (this is the default value), then URL path of request must be like "xxxx.hdp". To map the request to a `PaginationAction` defined in spring config file, that request's URL path with suffix (e.g. ".hdp") trimmed must have the same value as property `'id'` of that `PaginationAction`.

Considering a `paginationAction` defined as the follows:

```

<bean id="searchProductAction" class="web.OracleJdbcSearchProductAction">
  <property name="id" value="searchProduct" />
  <property name="forwardPath" value="/searchProduct.jsp" />
  <property name="attributeName4Data" value="query_result" />

  <property name="queryTemplate">
    <ref bean="jdbcQueryTemplate" />
  </property>
</bean>

```

The form in jsp page should be like this:

```

<form name="searchProdForm" action="/searchProduct.hdp" method="post">
...
...
  <input type="submit" value="Search" />
</form>

```

Chapter 4. QueryCallback

QueryCallback represents a SQL or Query (ORM) context which includes SQL/Query statement as well as parameters binding. The context will keep the same value across different pages and is stored in session. The framework uses it to remember the SQL/Query statement and binding information when user navigates to different pages so that it can execute the same query except different page number.

To implement the method `getQueryCallback` in custom action class, you have to create an instance of `QueryCallback` based on received input values and return it. Corresponding to different persistence technologies used for the query, there are 4 types of `QueryCallback`:

1. `JdbcQueryCallback` for JDBC API
2. `JPAQueryCallback` for JPA (Java Persistence API)
3. `HBN2QueryCallback` for Hibernate2 ORM framework
4. `HBN3QueryCallback` for Hibernate3 ORM framework

Here, a typical way to generate an instance of `QueryCallback` in `getQueryCallback` method is to instantiate an anonymous inner class extending from one of provided abstract `QueryCallback` classes.

Now let's discuss different types of `QueryCallback` in detail. Before going further, I would like to explain the background of the sample code used in the following sections. The sample source code is based on a simple use case – that is, user searches products by “made in countries” and “product description”. In the search input page,

the field “made in countries” is a multiple select dropdown and the field “description” is a text input. A part of the JSP page is as follows:

```
<form name="searchProdForm" action="/searchProduct.hdp" method="post">

  Made In:
  <select name="madeIn" multiple="multiple">
    <option value="China">China</option>
    <option value="USA">USA</option>
    <option value="Australia">Australia</option>
  </select>

  Description:
  <input type="text" name="description"/>

  <input type="submit" value="Search" />

</form>
```

The command class used by the actions (subclass of `AbstractCommandPaginationAction`) is `SearchCriteria`, which is defined as follows:

```
public class SearchCriteria {
    private String[] madeIn;
    private String description;

    public String[] getMadeIn() {
        return madeIn;
    }

    public void setMadeIn(String[] madeIn) {
        this.madeIn = madeIn;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

Command used by `AbstractCommandPaginationAction` is described in [chapter 3 Action](#).

4.1. *JdbcQueryCallback*

If the search functionality is coded on JDBC API, this is the type of `QueryCallback` instance you should return in your custom action class. Normally you don't have to implement a class of `org.hdpagination.dataaccess.jdbc.JdbcQueryCallback` interface directly. Some abstract classes are provided under the package `org.hdpagination.dataaccess.jdbc` for use:

4.1.1. AbstractJdbcQueryCallback

First of all, you create the SQL statement and pass it as one argument of construct. Then, there are two abstract methods you must implement:

```
public void setValues(PreparedStatement ps) throws SQLException;
```

```
public Object processRow(ResultSet rs) throws SQLException;
```

This first method is to bind parameters of SQL statement with the received search input values. And the second method is to map each row of data from `ResultSet` to an object of any class you prefer, which will be finally added to the returned list (`java.util.List`). A sample code of using `AbstractJdbcQueryCallback` class is as follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" select
        productId,prodNo,price,madeIn,description from product p where 1=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append("?");
        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(",");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and description = ? ");
    }

    AbstractJdbcQueryCallback callback = new
        AbstractJdbcQueryCallback(queryStmt.toString()) {
        public void setValues(PreparedStatement ps) throws SQLException {
            int pos = 1;
            for (int i=0;i<madeInSelect.length;i++) {
                ps.setString(pos++, madeInSelect[i]);
            }

            if (desc != null && !desc.trim().equals("")) {
                ps.setString(pos++, desc);
            }
        }

        public Object processRow(ResultSet rs) throws SQLException {
            ProductVO prod = new ProductVO();
            prod.setProductId(new Long(rs.getLong("productId")));
            prod.setPrice(new Integer(rs.getInt("price")));
            prod.setProdNo(rs.getString("prodNo"));
            prod.setMadeIn(rs.getString("madeIn"));
            prod.setDescription(rs.getString("description"));
            return prod;
        }
    };
    return callback;
}
```

4.1.2. AbstractColumnMapJdbcQueryCallback

This class inherits all features from `AbstractJdbcQueryCallback` except that it provides the default implementation of method `processRow (ResultSet rs)` where each row of data from `ResultSet` is mapped to an instance of `java.util.HashMap`. So as you can imagine you only need to implement one method:

```
public void setValues(PreparedStatement ps) throws SQLException.
```

The same code is like follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" select
        productId,prodNo,price,madeIn,description from product p where 1=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append("?");
        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(",");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and description = ? ");
    }

    AbstractColumnMapJdbcQueryCallback callback = new
        AbstractColumnMapJdbcQueryCallback(queryStmt.toString()) {
        public void setValues(PreparedStatement ps) throws SQLException {
            int pos = 1;
            if (madeIn != null && !madeIn.trim().equals("")) {
                ps.setString(pos++, madeIn);
            }
            if (desc != null && !desc.trim().equals("")) {
                ps.setString(pos++, desc);
            }
        }
    };
    return callback;
}
```

4.1.3. AbstractNoParamBindJdbcQueryCallback

This class also extends from `AbstractJdbcQueryCallback`. It does not provide method to bind parameters (which means there is no parameter in `PreparedStatement`) but you have to implement method `processRow(ResultSet rs)`. The sample code is as follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {

    StringBuffer queryStmt = new StringBuffer(" select
        productId,prodNo,price,madeIn,description from product p where 1=1 ");

    AbstractNoParamBindJdbcQueryCallback callback = new
        AbstractNoParamBindJdbcQueryCallback(queryStmt.toString()) {
        public Object processRow(ResultSet rs) throws SQLException {
            ProductVO prod = new ProductVO();
            prod.setProductId(new Long(rs.getLong("productId")));
            prod.setPrice(new Integer(rs.getInt("price")));
            prod.setProdNo(rs.getString("prodNo"));
        }
    };
}
```

```

        prod.setMadeIn(rs.getString("madeIn"));
        prod.setDescription(rs.getString("description"));
        return prod;
    }
};
return callback;
}

```

4.1.4. DefaultJdbcQueryCallback

As one subclass of `AbstractJdbcQueryCallback`, this class does not provide method to bind parameters (which means there is no parameter in `PreparedStatement`), and map each row of data from `ResultSet` to `java.util.HashMap`. The sample code is as follows:

```

public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {

    StringBuffer queryStmt = new StringBuffer(" select
        productId,prodNo,price,madeIn,description from product p where 1=1 ");

    return new DefaultJdbcQueryCallback(queryStmt.toString());
}

```

4.2. JPAQueryCallback

If the search functionality is coded on JPA (Java Persistence API), this is the type of `QueryCallback` you should return in your custom action class. Similar to `JdbcQueryCallback`, it is not recommended to implement the interface of `org.hdpagination.dataaccess.jdbc.JdbcQueryCallback` directly. Some abstract classes are provided under the package `org.hdpagination.dataaccess.orm.jpa` for use:

4.2.1. AbstractJPAQueryCallback

Similar to `AbstractJdbcQueryCallback`, you have to pass the EJB QL statement as one argument of construct. Then, there are two abstract methods you must implement:

```

abstract public void setValues(Query query);

abstract public List processQueriedResult(List queriedResult);

```

This first method is to bind parameters of EJB QL statement with the received search input values. And the second method is used to do treatment on the result (`java.util.List`) returned by JPA persistence engine before the connection (`EntityManager` in JPA) is closed. A typical case where you may use the second method is to initialize lazy-loading objects in the search result before `EntityManager` is closed. To do this, you just iterate the argument `queriedResult` and do lazy loading

initialization upon each element, then return the processed `queriedResult` in that method. The sample code of using `AbstractJPAQueryCallback` is as follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command)
{
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and p.madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append("'" + madeInSelect[i] + "'");
        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(",");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and p.description = :description ");
    }

    AbstractJPAQueryCallback callback =
        new AbstractJPAQueryCallback(queryStmt.toString()) {
        public void setValues(Query query) {
            if (desc != null && !desc.trim().equals("")) {
                query.setParameter("description", desc);
            }
        }

        public List processQueriedResult(List queriedResult) {
            Iterator it = queriedResult.iterator();
            while (it.hasNext()) {
                Object element = it.next();
                //do lazy loading initialization...
            }
            return queriedResult;
        }
    };

    return callback;
}
```

4.2.2. NotProcessResultJPAQueryCallback

This class inherits all features from `AbstractJPAQueryCallback` except that it provides a default implementation of the method `processQueriedResult(List queriedResult)` – that is, the method returns the argument `queriedResult` directly without any treatment. In the majority of cases, this is the class that should be used if you do not have to worry about the lazy loading issue. The sample code of using this class is as follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and p.madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append("'" + madeInSelect[i] + "'");
    }
}
```

```

        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(",");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and p.description = :description ");
    }

    NotProcessResultJPAQueryCallback callback =
        new NotProcessResultJPAQueryCallback (queryStmt.toString()) {
        public void setValues(Query query) {
            if (desc != null && !desc.trim().equals("")) {
                query.setParameter("description", desc);
            }
        }
    };

    return callback;
}

```

4.2.3. NoParamBindJPAQueryCallback

This class extends from `AbstractJPAQueryCallback` but does not provide method to bind parameters of EJB QL statement with values from input (this means there is no parameter in the EJB QL statement). The sample code of using this class is as follows:

```

public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {

    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");

    NoParamBindJPAQueryCallback callback =
        new NoParamBindJPAQueryCallback (queryStmt.toString()) {
        public List processQueriedResult(List queriedResult) {
            Iterator it = queriedResult.iterator();
            while (it.hasNext()) {
                Object element = it.next();
                //do lazy loading initialization...
            }
            return queriedResult;
        }
    };

    return callback;
}

```

4.2.4. DefaultJPAQueryCallback

This class does not provide method to bind parameters with values from input (this means there is no parameter in the EJB QL statement). Also it does not allow user to process `queriedResult` before `EntityManager` is closed. The sample code of using this class is as follows:

```

public QueryCallback getQueryCallback(HttpServletRequest request, Object command)
{

```

```

StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");
return new DefaultJPAQueryCallback (queryStmt.toString());
}

```

4.3. HBN3QueryCallback

If the search functionality is coded on Hibernate3, this is the type of `QueryCallback` you should return in your custom action class. Like `JPAQueryCallback`, you have some abstract classes under the package

`org.hdpagination.dataaccess.orm.hibernate3` to use:

4.3.1. AbstractHBN3QueryCallback

Similar to `AbstractJPAQueryCallback`, you have to pass the HQL statement as one argument of construct. Then, there are two abstract methods you must implement:

```
abstract public void setValues(Query query);
```

```
abstract public List processQueriedResult(List queriedResult);
```

This first method is to bind parameters of HQL statement with the received search input values. And the second method is used to do treatment on the result (`java.util.List`) returned by `Query` before the connection (`Session` in Hibernate) is closed. A typical case where you may use the second method is to initialize lazy-loading objects in the search result before `Session` is closed. To do this, you just iterate the argument `queriedResult` and do initialization upon each element, then return the processed `queriedResult` in that method. The sample code to use `AbstractHBN3QueryCallback` is as follows:

```

public QueryCallback getQueryCallback(HttpServletRequest request, Object command)
{
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and p.madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append("'" + madeInSelect[i] + "'");
        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(", ");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and p.description = :description ");
    }

    AbstractHBN3QueryCallback callback =
        new AbstractHBN3QueryCallback (queryStmt.toString()) {
        public void setValues(Query query) {
            if (desc != null && !desc.trim().equals("")) {
                query.setParameter("description", desc);
            }
        }
    }
}

```

```

public List processQueriedResult(List queriedResult) {
    Iterator it = queriedResult.iterator();
    while (it.hasNext()) {
        Object element = it.next();
        //do lazy loading initialization...
    }
    return queriedResult;
}
};

return callback

```

4.3.2. NotProcessResultHBN3QueryCallback

This class inherits the same features from `AbstractHBN3QueryCallback` except that it provides a default implementation of the method `processQueriedResult(List queriedResult)` – that is, the method returns the argument `queriedResult` directly without any treatment. Similar to `NotProcessResultJPAQueryCallback`, in the majority of cases, this is the class that should be used if you do not have to worry about the lazy loading issue. The sample code of using this class is as follows:

```

public QueryCallback getQueryCallback(HttpServletRequest request, Object command)
{
    SearchCriteria crit = (SearchCriteria)command;
    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");

    final String[] madeInSelect = crit.getMadeIn();
    queryStmt.append(" and p.madeIn in (");
    for (int i=0;i<madeInSelect.length;i++) {
        queryStmt.append(""+madeInSelect[i]+"");
        if (i < (madeInSelect.length - 1)) {
            queryStmt.append(",");
        }
    }
    queryStmt.append(")");

    final String desc = crit.getDescription();
    if (desc != null && !desc.trim().equals("")) {
        queryStmt.append(" and p.description = :description ");
    }

    NotProcessResultHBN3QueryCallback callback =
        new NotProcessResultHBN3QueryCallback (queryStmt.toString()) {
        public void setValues(Query query) {
            if (madeIn != null && !madeIn.trim().equals("")) {
                query.setParameter("madeIn", madeIn);
            }
            if (desc != null && !desc.trim().equals("")) {
                query.setParameter("description", desc);
            }
        }
    };

    return callback;
}

```

4.3.3. NoParamBindHBN3QueryCallback

This class extends from `AbstractHBN3QueryCallback` except that it does not provide method to bind parameters of HQL statement with values from input (this means there is no parameter in the HQL statement). The sample code of using this class is as follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command) {
    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");

    NoParamBindHBN3QueryCallback callback =
        new NoParamBindHBN3QueryCallback(queryStmt.toString()) {
            public List processQueriedResult(List queriedResult) {
                Iterator it = queriedResult.iterator();
                while (it.hasNext()) {
                    Object element = it.next();
                    //do lazy loading initialization...
                }
                return queriedResult;
            }
        };

    return callback;
}
```

4.3.4. DefaultHBN3QueryCallback

This class does not provide method to bind parameters with values from input (this means there is no parameter in the HQL statement). And it also does not allow user to process `queriedResult` before Session is closed. The same code of using this class is as follows:

```
public QueryCallback getQueryCallback(HttpServletRequest request, Object command)
{
    StringBuffer queryStmt = new StringBuffer(" from ProductVO p where l=1 ");
    return new DefaultHBN3QueryCallback(queryStmt.toString());
}
```

4.4. HBN2QueryCallback

There is almost no difference between `HBN3QueryCallback` and `HBN2QueryCallback` except referencing different Hibernate API.

4.5. Set your own query statement to count records

By default, `HDPagination` use the `'queryStatement'` property (let's call it original statement) in `QueryCallback` to generate a query statement to count total records. For example, if it's a SQL statement, it will use the string `'select count (*)'` to replace the `select` clause before the keyword `'from'` in the original SQL statement to generate a count SQL.

This is probably fine in majority cases; however, in some cases you may need to set the count statement manually without relying on HDPagination's auto-translation. For example, if the 'group by' clause is used in the original statement, the count statement auto-generated by HDPagination will work.

You can set the count statement manually by calling the method:

- `setCountRowsStmt` in class `AbstractJdbcQueryCallback` for JDBC API
- `setCountRecordsQueryStatement` in class `AbstractJPAQueryCallback` for JPA (Java Persistence API)
- `setCountRecordsQueryStatement` in class `AbstractHBN3QueryCallback` for Hibernate3 ORM framework
- `setCountRecordsQueryStatement` in class `AbstractHBN2QueryCallback` for Hibernate2 ORM framework

Importantly, if you set the count statement manually, it should always have the same parameters (even in the same order) as the original statement. Because HDPagination will apply the same parameter values binding to the count statement as the original statement.

Chapter 5. QueryTemplate

Interface `QueryTemplate` represents the query execution engine which runs a pagination based query against different persistence technologies. The persistence technologies include JDBC and ORM (e.g. JPA, Hibernate or TopLink etc.). For JDBC, because each database providers has their own SQL syntax to support the pagination feature, so for all supported database providers there is a corresponding `QueryTemplate` class. In current version, the supported databases include Oracle, DB2, MySQL, HSQLDB, MS SQL Server 2005 (or later version), Sybase SQL Anywhere and PostgreSQL.

To allow the developed custom action class to work correctly against particular persistence technology or database, you have to make sure the correct `QueryTemplate` instance is injected into this action instance via spring configuration file.

5.1. JPAQueryTemplate

`JPAQueryTemplate` class should be used if the query is run against JPA. The way to inject a `JPAQueryTemplate` instance into action in spring configuration file is as follows:

```
<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="examplePersistence"/>
  <property name="loadTimeWeaver">
    <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>
  </property>
</bean>
```

```

</property>
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.orm.jpa.JPAQueryTemplate">
  <constructor-arg><ref bean="entityManagerFactory"/></constructor-arg>
</bean>

<bean id="queryProductAction" class="eg.web.action.jpa.QueryProductAction">
  <property name="id" value="queryProduct"/>
  <property name="forwardPath" value="/shopping/queryProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>

```

As you can see, it is very simple to configure `JPAQueryTemplate` instance and inject it into action instance in spring if `EntityManagerFactory` of JPA is already configured somewhere.

5.2. Hibernate3QueryTemplate

`Hibernate3QueryTemplate` class should be used if the query is run against Hibernate3. The way to inject a `Hibernate3QueryTemplate` instance into action in spring configuration file is as follows:

```

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
  <property name="url" value="jdbc:hsqldb:hsqldb://localhost/xdm"/>
  <property name="username" value="sa"/>
  <property name="password" value="" />
</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="mappingResources">
    <list>
      <value>eg/vo/ProductVO-hbn3.hbm.xml</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <value>hibernate.dialect=org.hibernate.dialect.HSQLDialect</value>
  </property>
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.orm.hibernate3.Hibernate3QueryTemplate">
  <constructor-arg><ref bean="sessionFactory"/></constructor-arg>
</bean>

<bean id="queryProductAction" class="eg.web.action.hbn3.QueryProductAction">
  <property name="id" value="queryProduct"/>
  <property name="forwardPath" value="/shopping/queryProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>

```

5.3. Hibernate2QueryTemplate

Hibernate2QueryTemplate class should be used if the query is run against Hibernate2. The way to inject a Hibernate2QueryTemplate instance into action in spring configuration file is as follows:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
  <property name="url" value="jdbc:hsqldb:hsqldb://localhost/xdm"/>
  <property name="username" value="sa"/>
  <property name="password" value="" />
</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
  <property name="dataSource">
    <ref bean="dataSource"/>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">net.sf.hibernate.dialect.HSQLDialect</prop>
    </props>
  </property>
  <property name="mappingResources">
    <list>
      <value>eg/vo/ProductVO-hbn2.hbm.xml</value>
    </list>
  </property>
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.orm.hibernate2.Hibernate2QueryTemplate">
  <constructor-arg><ref bean="sessionFactory"/></constructor-arg>
</bean>

<bean id="queryProductAction" class="eg.web.action.hbn3.QueryProductAction">
  <property name="id" value="queryProduct"/>
  <property name="forwardPath" value="/shopping/queryProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean=" queryTemplate "/>
  </property>
</bean>
```

5.4. JdbcQueryTemplate

JdbcQueryTemplate class is used when the query is run against JDBC API. However you can not use this class directly because it is an abstract class. You have to use its subclasses instead which correspond to the particular database providers (e.g. Oracle). The way to configure in spring is as follows:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
  <property name="url" value="jdbc:hsqldb:hsqldb://localhost/xdm"/>
  <property name="username" value="sa"/>
  <property name="password" value="" />
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.jdbc.hsqldb.HSQLDBJdbcQueryTemplate">
  <constructor-arg><ref bean="dataSource"/></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  <property name="id" value="searchProduct"/>
  <property name="forwardPath" value="/searchProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>
```

```
<property name="queryTemplate">
  <ref bean="queryTemplate"/>
</property>
</bean>
```

HSQldb

5.4.1. HSQldbJdbcQueryTemplate

The above is a sample configuration for HSQldb database. If you want to configure for other databases, just modify to inject different class as follows:

5.4.2. OracleJdbcQueryTemplate

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  .....
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.jdbc.oracle.OracleJdbcQueryTemplate">
  <constructor-arg><ref bean="dataSource"/></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  .....
  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>
```

Oracle

5.4.3. DB2JdbcQueryTemplate

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  .....
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.jdbc.db2.DB2JdbcQueryTemplate">
  <constructor-arg><ref bean="dataSource"/></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  .....
  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>
```

DB2

5.4.4. MySQLJdbcQueryTemplate

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    .....
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.jdbc.mysql.MySQLJdbcQueryTemplate">
    <constructor-arg><ref bean="dataSource" /></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
    .....
    <property name="queryTemplate">
        <ref bean="queryTemplate" />
    </property>
</bean>
```

MySQL

5.4.5. SQLServer2005JdbcQueryTemplate

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    .....
</bean>

<bean id="queryTemplate"
    class="org.hdpagination.dataaccess.jdbc.sqlserver.SQLServer2005JdbcQueryTemplate">
    <constructor-arg><ref bean="dataSource" /></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
    .....
    <property name="queryTemplate">
        <ref bean="queryTemplate" />
    </property>
</bean>
```

MS SQL Server 2005

5.4.6. SQLAnywhereJdbcQueryTemplate

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    .....
</bean>

<bean id="queryTemplate"
    class="org.hdpagination.dataaccess.jdbc.sybase.SQLAnywhereJdbcQueryTemplate">
    <constructor-arg><ref bean="dataSource" /></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
    .....
    <property name="queryTemplate">
        <ref bean="queryTemplate" />
    </property>
</bean>
```

Sybase SQL Anywhere

5.4.7. PostgreSQL

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    .....
</bean>

<bean id="queryTemplate"
    class="org.hdpagination.dataaccess.jdbc.postgresql.PostgreSQLQueryTemplate">
    <constructor-arg><ref bean="dataSource" /></constructor-arg>
</bean>

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
    .....
    <property name="queryTemplate">
        <ref bean="queryTemplate" />
    </property>
</bean>
```

PostgreSQL

5.4.8. StandardJdbcQueryTemplate

If you can not find matched `JdbcQueryTemplate` in above list for the database your application is using, however this database supports standard SQL syntax (`ROW_NUMBER() OVER`) for pagination, then you can use `StandardJdbcQueryTemplate`.

Chapter 6. Validator

Note: this is only for Non-JSF application.

Sometimes input validation is required in search functionality, and `HDPagination` provides a way to plug-in a custom `Validator`. The steps to do custom validation are as follows:

1. Create a custom `validator` class which implements the interface `org.hdpagination.web.validate.Validator`.
2. Create resource bundler file containing error message definition and inject it into `globalConfig` (`errorMessageBundle` property of `org.hdpagination.config.GlobalConfig`) in spring configure file like:

```
<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
    .....
    <property name="errorMessageBundle" value="eg.web.action.ErrorMessage" />
    ....
</bean>
```

3. In the method `validate(HttpServletRequest request, Object command)` of custom `validator` class created in step 1, perform validation, if no error is found then return null or an empty instance of `org.hdpagination.web.validate.ErrorMessages`.
4. If error is found in validation, call `addError(String errorKey)` method of `org.hdpagination.web.validate.ErrorMessages`, providing value `errorKey` which is defined in resource bundle file created in step 2.
5. Finally use `errors` tag to display error messages somewhere in jsp file like follows:

```
<hdp:errors/>
```

The sample code of custom `validator` is as follows:

```
public class SearchProdValidator implements Validator {
    public ErrorMessages validate(HttpServletRequest request, Object command) {
        ErrorMessages errors = new ErrorMessages();
        SearchCriteria crit = (SearchCriteria)command;
        if (crit.getMadeIn() == null || crit.getMadeIn().length == 0) {
            errors.addError("noMadeInSelected");
        }
        return errors;
    }
}
```

If the validation error messages do not need to support internationalization, you can call another method `addErrorWithoutKey` of `ErrorMessages` to add an error message which will be rendered directly by `<hdp:errors/>` tag. The above custom validator will be as follows:

```
public class SearchProdValidator implements Validator {
    public ErrorMessages validate(HttpServletRequest request, Object command) {
        ErrorMessages errors = new ErrorMessages();
        SearchCriteria crit = (SearchCriteria)command;
        if (crit.getMadeIn() == null || crit.getMadeIn().length == 0) {
            errors.addErrorWithoutKey("At least one country need to be selected");
        }
        return errors;
    }
}
```

By doing this way, the error message resource bundler file configured in step 2 above will not be used so you can skip that step if it's not used by other validtors.

The way to inject this custom `validator` into action in spring configure file is as follows:

```
<bean id="searchProdValidator" class="eg.web.action.SearchProdValidator"/>
```

```
<bean id="searchProductAction" class="web.OracleJdbcSearchProductAction">
  <property name="id" value="searchProduct"/>
  <property name="forwardPath" value="/searchProduct.jsp"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="jdbcQueryTemplate"/>
  </property>

  <property name="validator">
    <ref bean="searchProdValidator"/>
  </property>
</bean>
```

Chapter 7. GlobalConfig

7.1. Properties

GlobalConfig contains the pagination related configure information. It has the following properties:

7.1.1. pageSize

It defines how many records should be displayed in one page.

7.1.2. defaultDataScope

It defines the default scope where the search result is stored. The valid values of scope include 'request', 'session' and 'application'. If the 'dataScope' property of action is not set, this default scope will be used instead. 'request' is highly recommended to be used as the default scope.

7.1.3. pageLinksPattern

It defines the layout or style of "Pagination Control" links in page. More detailed information will be discussed in later chapter.

7.1.4. sortUpImage & sortDownImage

Property 'sortUpImage' defines the location (relative to the context path of web application) of image file which indicates the search results are sorted in ascending order.

Property 'sortDownImage' defines the location (relative to the context path of web application) of image file which indicates the search results are sorted in descending order.

7.1.5. urlSuffix

Define the suffix (after '.') of URL that will be dispatched to the action instances for processing. In another word, only request whose URL path ends with such suffix are thought to be a pagination search request.

Note:

This property is only used by JSP pagination. JSF pagination does not need this.

7.1.6. errorMessageBundle

It defines the resource bundle file which defines error messages for input validation.

Note:

This property is only used by JSP pagination. JSF pagination does not need this.

7.1.7. view

View represents a component which is responsible for rendering page after searching finishes. The default view implementation is `InternalResourceView` which just forwards the request to a JSP or other resource URL within the same web application using `RequestDispatcher`. The default view is often sufficient for majority of web applications where you don't have to configure this property explicitly. However, if you are using Tiles to manage your page layout, you can choose one of the other 2 implementations:

TilesView

If your application uses Tiles 1.x (a.k.a. "Struts Tiles") as page template and you want HDPagination to render a definition rather than forward to a JSP (or other internal resource) directly, then an instance of `TilesView` should be configured to this property. For more information, see the section entitled [11.2 Integrate with Tiles 1.x \(a.k.a. "Struts Tiles"\)](#).

Tiles2View

If your application uses Tile 2 as page template and you want HDPagination to render a definition rather than forward to a JSP (or other internal resource) directly, then an instance of `Tiles2View` should be configured. For more information, see the section entitled [11.1 Integrate with Tiles 2](#).

It should be noted that the "view" property here (`GlobalConfig`) has the same usage as that in `Action` (see the section entitled [3.1.2.9](#)), except that the setting here applies to all `Actions` in application while the setting at the `Action` only takes effect on that `Action` itself. In short, if "view" properties in both `Action` and `GlobalConfig` are set, the "view" property in `Action` will take precedence (and the "view" property in `GlobalConfig` will not apply)

Note:

This property is only used by JSP pagination. JSF pagination does not need this.

7.2. Configure GlobalConfig in spring

To configure GlobalConfig is just to declare a bean of org.hdpagination.config.GlobalConfig in spring configure file as follows:

```
<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
  <property name="pageSize" value="10"/>
  <property name="defaultDataScope" value="request"/>
  <property name="urlSuffix" value="hdp"/>
  <property name="sortUpImage" value="/images/pagination_up.gif"/>
  <property name="sortDownImage" value="/images/pagination_down.gif"/>
  <property name="errorMessageBundle" value="eg.web.action.ErrorMessage"/>
  <property name="pageLinksPattern">
    <ref bean="pageLinksPatternYahoo"/>
  </property>
</bean>

<bean id="pageLinksPatternYahoo"
      class="org.hdpagination.config.pattern.YahooPageLinksPattern">
  <constructor-arg type="int" value="5"/>
  <property name="resourceBundleName" value="eg.web.action.PaginalLinks"/>
  <property name="displayFirstPageLink" value="false"/>
  <property name="displayLastPageLink" value="false"/>
</bean>
```

It's worth noting here that the 'id' attribute of bean element can be any value and the bean should be singleton (that's the default in spring).

Chapter 8. Change page size from page UI

Since version 1.2, HDPagination began to support the use case that the user can change the page size dynamically from the UI input. A special HTTP request parameter name ('page_size') is reserved to achieve this purpose. The way to use this parameter in a JSP page is as follows:

```
<form name="searchProdForm" action="/searchProduct.hdp" method="post">
  Description: <input type="text" name="description"/>
  ...
  <%
    String pageSize = request.getParameter("page_size") == null ? "" :
      request.getParameter("page_size");
  %>
  <input type="text" name="page_size" value="<%=pageSize%>" size="2" />
  <input type="submit" value="Search" />
</form>
```

By doing this, if the user types the number in the text input field and submits the search form, the search results will be paginated based on that filled page size. As a matter of fact, it can be other type of HTML input, such as select (this is probably more user friendly).

If you prefer to using a different parameter name to pass the page size, you can override the default reserved parameter name ('page_size') by setting the property 'pageSizeParamName' of GlobalConfig in the spring application context file.

Chapter 9. Tags

To include HDPagination JSP tag definition, add the following directive on top of the JSP file:

```
<%@ taglib uri="http://www.hdpagination.org/tags" prefix="hdp" %>
```

Here, <http://www.hdpagination.org/tags> represents the unique URI for HDPagination JSP tags.

9.1. <hdp:pageLinks>

9.1.1. Description

This JSP Tag is used to render pagination navigation links (e.g. Previous Next 1 2 3 4 5 6 7 8 First Last).

9.1.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.1.3. How to use

The way to use this tag in jsp is as the following:

```
<hdp:pageLinks actionId="searchProduct" resendParams="true" />
```

9.1.4. Configure PageLinksPattern

You can choose different patterns to display different style or layout of the "Pagination Control" links (e.g. navigation links such as “Previous”, “Next”, “First”,

“Last” or page links like 1 2 3 4 5 6 7 8). There are a few patterns available for selection:

- `org.hdpagination.config.pattern.OnlyNavigatePageLinksPattern`
- `org.hdpagination.config.pattern.DefaultPageLinksPattern`
- `org.hdpagination.config.pattern.YahooPageLinksPattern`

They share some common features and rules:

1. When on first page of results do not display the “Previous” and “First” page links.
2. When on the last page of results do not display the “Next” and “Last” page links.
3. You can choose not to display “First” page link always by setting property ‘displayFirstPageLink’ with “false”.
4. You can choose not to display “Last” page link always by setting property ‘displayLastPageLink’ with “false”.
5. The label of navigation page links (“First”, “Previous”, “Next” and “Last”) is configured in a resource bundle file whose location is defined by the property ‘resourceBundleName’. For instance, if the value of ‘resourceBundleName’ is “PaginalLinks”, then the content of PaginalLinks.properties can be as follows:

```
firstPage=First
lastPage=Last
previousPage=Previous
nextPage=Next
```

As a result, the navigation page links will display as follows:

[First](#) [Previous](#) [Next](#) [Last](#)

Meanwhile, it supports internationalization.

OnlyNavigatePageLinksPattern

This pattern displays 'Pagination Control' links as the following pattern or style:

[First](#) [Previous](#) [Next](#) [Last](#)

To configure this pattern is as simple as:

```
<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
  <property name="pageSize" value="10"/>
  <property name="defaultDataScope" value="request"/>
  <property name="urlSuffix" value="hdp"/>
  <property name="sortUpImage" value="/images/pagination_up.gif"/>
  <property name="sortDownImage" value="/images/pagination_down.gif"/>
  <property name="errorMessageBundle" value="eg.web.action.ErrorMessage"/>
</bean>
```

```

<property name="pageLinksPattern">
  <ref bean="pageLinksPattern" />
</property>
</bean>

<bean id="pageLinksPattern"
      class="org.hdpagination.config.pattern.OnlyNavigatePageLinksPattern">
  <property name="resourceBundleName" value="eg.web.action.PaginalLinks" />
  <property name="displayFirstPageLink" value="false" />
  <property name="displayLastPageLink" value="false" />
</bean>

```

DefaultPageLinksPattern

This pattern displays 'Pagination Control' links as the following pattern or style:

[First](#) [Previous](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [Next](#) [Last](#)

Apart from the common rules mentioned above, its special rules for displaying page links can be described as follows:

1. The page links contain a maximum set of page links defined by its 'maxPageLinks2Display' property. If fewer pages of results exist, only show page links for the available pages.
2. The page links should always start at '1', $1 + \text{maxPageLinks2Display}$, $1 + \text{maxPageLinks2Display} * 2$, ... $1 + \text{maxPageLinks2Display} * n$. For example if 'maxPageLinks2Display' is 10, then it should always start at '1' when on page between 1 and 10, start at '11' when on page between 11 and 20.
3. It does not display a hyperlink for the current page in the page links.

To configure this pattern is as follows:

```

<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
  <property name="pageSize" value="10" />
  <property name="defaultDataScope" value="request" />
  <property name="urlSuffix" value="hdp" />
  <property name="sortUpImage" value="/images/pagination_up.gif" />
  <property name="sortDownImage" value="/images/pagination_down.gif" />
  <property name="errorMessageBundle" value="eg.web.action.ErrorMessage" />

  <property name="pageLinksPattern">
    <ref bean="pageLinksPattern" />
  </property>
</bean>

<bean id="pageLinksPattern"
      class="org.hdpagination.config.pattern.DefaultPageLinksPattern">
  <constructor-arg type="int" value="10" />
  <property name="resourceBundleName" value="eg.web.action.PaginalLinks" />
  <property name="displayFirstPageLink" value="true" />
  <property name="displayLastPageLink" value="true" />
</bean>

```

YahooPageLinksPattern

This pattern displays 'Pagination Control' links as the following pattern or style:

[First](#) [Previous](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [Next](#) [Last](#)

Apart from the common rules mentioned above, its special rules for displaying page links can be described as follows:

1. The page links contain a maximum set of page links defined by its 'maxPageLinks2Display' property. If fewer pages of results exist, only show page links for the available pages.
2. When on pages 1 - (maxPageLinks2Display/2+1), the page links should always start at '1'. When on any page after (maxPageLinks2Display/2+1), the page links should start at the current page minus maxPageLinks2Display/2. For example, if maxPageLinks2Display is 10, then when on pages 1 - 6, the page links should always start at '1'. When on any page after 6 (7 and onward), the page links should start at the current page minus 5. e.g., when on page 7, the first page will be 2 (7 - 5 = 2) and the last page will be 11 (still shows 10 pages).
3. It does not display a hyperlink for the current page in the page links.

To configure this pattern is as follows:

```
<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
  <property name="pageSize" value="10"/>
  <property name="defaultDataScope" value="request"/>
  <property name="urlSuffix" value="hdp"/>
  <property name="sortUpImage" value="/images/pagination_up.gif"/>
  <property name="sortDownImage" value="/images/pagination_down.gif"/>
  <property name="errorMessageBundle" value="eg.web.action.ErrorMessage"/>

  <property name="pageLinksPattern">
    <ref bean="pageLinksPattern"/>
  </property>
</bean>

<bean id="pageLinksPattern"
      class="org.hdpagination.config.pattern.YahooPageLinksPattern">
  <constructor-arg type="int" value="10"/>
  <property name="resourceBundleName" value="eg.web.action.PaginalLinks"/>
</bean>
```

9.2. <hdp:sortBy>

9.2.1. Description

This JSP Tag is used to apply a hyperlink (<a>) to the column header in the search results table. This allows the user to sort the search results by this column. By default, when the column header is clicked and the current search result is not sorted by that column, then the result will be displayed in ascending order. The search results will be

toggled between ascending and descending order every time the column header is clicked.

9.2.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.2.3. How to use

The way to use this tag in JSP is as the following:

```
<table width="80%" border="1" cellpadding="5" cellspacing="0">
  <tr>
    <td class="tableHeading" width="16%">
      Product No
    </td>
    <td class="tableHeading" width="22%">
      <hdp:sortBy actionId="searchProduct" sortedColumn="p.price" resendParams="true">
        Price
      </hdp:sortBy>
    </td>
    <td class="tableHeading" width="18%">
      Made In
    </td>
    <td class="tableHeading" width="18%">
      Description
    </td>
  </tr>

  <c:forEach var="each" items="${query_result}">
    <tr>
      <td><div align="center"><c:out value="${each.prodNo}"/></div></td>
      <td><div align="center"><c:out value="${each.price}"/></div></td>
      <td><div align="center"><c:out value="${each.madeIn}"/></div></td>
      <td><div align="center"><c:out value="${each.description}"/></div></td>
    </tr>
  </c:forEach>
</table>
```

In the sample code above, the ‘sortBy’ tag is applied upon the “Price” column.

9.2.4. sortUpImage & sortDownImage

As I have mentioned in Chapter 6.4, these 2 properties of GlobalConfig are used to specify the location of images which indicate the result is sorted by this column in ascending or descending order. For example, it will be displayed as the following screen if the result is sorted by “Price” column in ascending order:

Search Result			
Product No	▲Price	Made In	Description
6	20\$	China	Silk
1	30\$	China	Cotton
2	35\$	China	Cotton

1 2 3 4 <Next>

By clicking the links of “Price” again, search results will change to descending order as follows:

Search Result			
Product No	Price	Made In	Description
11	430\$	USA	Silk
9	330\$	USA	Silk
7	230\$	USA	Cotton

1 2 3 4 <Next>

9.3. <hdp:selectSortBy>

9.3.1. Description

This tag is used to render a dropdown (<select>) to allow the user to sort the search results by specified column in either ascending or descending order.

This tag is an alternative of the <hdp:sortBy> tag which applies a sorting hyperlink to the column header in a search results table. This tag can be useful in some cases where there is no column header in the search results table (as a result you can not apply the <hdp:sortBy> tag to the column header).

The <hdp:selectSortBy> tag should be used together with <hdp:sortByOption> tag. <hdp:sortByOption> tags must be nested inside <hdp:selectSortBy> tag and each <hdp:sortByOption> tag will render as a <option> HTML tag.

9.3.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.3.3. How to use

The way to use this tag in jsp is as the following:

```
<hdp:selectSortBy actionId="searchProduct" resendParams="true">
  <hdp:sortByOption sortedColumn="p.price" ascending="true" label="Price (Low - High)"/>
  <hdp:sortByOption sortedColumn="p.price" ascending="false" label="Price (High - Low)"/>
  <hdp:sortByOption sortedColumn="p.madeIn" label="Made In"/>
  <hdp:sortByOption sortedColumn="p.description" label="Description"/>
</hdp:selectSortBy>
```

The sample code above renders as a HTML select dropdown with 4 options. An HTML onchange event will be fired after changing of option selection. As a result, the search results will be resorted by the specified column in ascending or descending order as specified.

9.3.4. Support for internationalization and localization

In an application supporting multiple languages, the dropdown select options should display different language labels based on the locale in client's request. Since the attribute 'label' in `<hdp:sortByOption>` tag accepts a run time expression ('rtexprvalue' is set true), it opens the door to support internationalization and localization. For example, you can utilize JSTL to fetch resource bundle message by doing the following steps:

Add the following to web.xml:

```
<context-param>
  <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
  <param-value>eg.web.action.LocalizationResources</param-value>
</context-param>
```

In the JSP file, get resource bundle message by calling the method `getLocalizedMessage` in class `javax.servlet.jsp.jstl.fmt.LocaleSupport` (which is provided by JSTL), then use JSP code to assign the fetched message to the attribute 'label' of `<hdp:sortByOption>` tag. The code is as follows:

```
<%
String madeInMsg = LocaleSupport.getLocalizedMessage(pageContext, "madeIn");
String descMsg = LocaleSupport.getLocalizedMessage(pageContext, "desc");
%>
<hdp:selectSortBy actionId="searchProduct" resendParams="true">
  <hdp:sortByOption sortedColumn="p.madeIn" label="<%=madeInMsg%"/>
  <hdp:sortByOption sortedColumn="p.description" label="<%=descMsg%"/>
</hdp:selectSortBy>
```

Please keep in mind, this is just one example of how to use resource bundle message in the tag to support internationalization and localization. There are many other ways to work out this issue and the discussion of internationalization in J2EE applications is beyond the scope of this document.

9.4. `<hdp:pagesDropdown>`

9.4.1. Description

This JSP Tag is used to render a dropdown (`<select>`) with all available pages, which allows the user to navigate to different pages.

9.4.2. Attributes

Refer to the JSP taglib document (which is under the directory "doc/jsp-taglib" after unzipping the distribution).

9.4.3. How to use

The way to use this tag in jsp is as the following:

```
<hdp:pagesDropdown actionId="searchProduct" resendParams="true"/>
```

As a result, the dropdown will displays like:



9.5. <hdp:gotoPage>

9.5.1. Description

This JSP Tag is used to render a text box along with either a submit button, hyperlink, or image link. The button, hyperlink, or image link is used to submit the request and navigate to the page specified in the text box.

The decision to render the button, hyperlink, or image link relies on the following rule:

- If property 'imgSrc' is specified, then an image link (tag with enclosing <a> tag) is rendered.
- Otherwise, if property 'linkLabel' is specified, the hyperlink (text with enclosing tag <a>) is rendered.
- If none of these are specified, the button is rendered by default.

9.5.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.5.3. How to use

The way to use this tag in JSP page is as the following:

```
<hdp:gotoPage actionId="searchProduct" resendParams="true" buttonLabel="go"/>
```

It will display like:



9.5.4. Support for internationalization and localization

Like the attribute 'label' in `<hdp:sortByOption>` tag, attributes 'invalidPageAlert', 'exceedMaxPageAlert', 'buttonLabel', and 'linkLabel' in this tag accept a run time expression ('rtexprvalue' is set true), so you can use the same way as described in section [9.3.4](#) to support internationalization and localization.

9.6. `<hdp:currentPage>`

9.6.1. Description

This JSP Tag is used to display the current page number (or called index of the current page). The index starts from 1 - that is, it displays 1 for the first page.

9.6.2. Attributes

Refer to the JSP taglib document (which is under the directory "doc/jsp-taglib" after unzipping the distribution).

9.7. `<hdp:totalPages>`

9.7.1. Description

This JSP Tag is used to display the count of pages - that is, how many pages of records in the search results.

9.7.2. Attributes

Refer to the JSP taglib document (which is under the directory "doc/jsp-taglib" after unzipping the distribution).

9.8. `<hdp:recordScope>`

9.8.1. Description

This JSP tag is used to display the scope of records in current page - that is, index (starting from 1) of the first and last records in the current page.

For example, if the page size is defined to 10, then it will display "11 - 20" for the second page if there are more than (including) 20 records totally in the search result.

9.8.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.9. *<hdp:totalRecords>*

9.9.1. Description

This JSP Tag is used to display the count of pages - that is, how many pages of records in the search results.

9.9.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.10. *<hdp:paginationRequired>*

9.10.1. Description

This JSP tag is used as an outer tag of other pagination information display tags (*<hdp:currentPage>*, *<hdp:totalPages>*, *<hdp:totalRecords>* and *<hdp:recordScope>*).

The purpose of this tag is to check if search result can fit within one page. If that's the case, then its enclosed content (including what is generated by the enclosed tags) will not be rendered.

9.10.2. Attributes

Refer to the JSP taglib document (which is under the directory “doc/jsp-taglib” after unzipping the distribution).

9.10.3. How to use

The way to use this tag in jsp is as the following:

```
<hdp:paginationRequired actionId="searchProduct" >
  <TR>
    <TD>
      Page&nbsp;
      <hdp:currentPage actionId="searchProduct" />
      &nbsp;of&nbsp;
      <hdp:totalPages actionId="searchProduct" />
      &nbsp;&nbsp;
      <hdp:pagesDropdown actionId="searchProduct" resendParams="true" />
      &nbsp;&nbsp;
    </TD>
  </TR>
</hdp:paginationRequired >
```


(c) Add pagination servlet and its URL mapping

```
<servlet>
  <servlet-name>hdpagination</servlet-name>
  <servlet-class>org.hdpagination.web.servlet.PaginationServlet</servlet-class>
  <load-on-startup>3</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>hdpagination</servlet-name>
  <url-pattern>*.hdp</url-pattern>
</servlet-mapping>
```

If property 'urlSuffix' of GlobalConfig is set with value other than 'hdp', you have to modify <url-pattern> correspondingly.

If it's used in a JSF application, you do not need to configure PaginationServlet.

Chapter 11. Integrate with Tiles View Technologies

By default, HDPagination just uses JSP to display the searching result by forwarding the request to a JSP file or other internal resource which is specified in the property 'forwardPath' of Action. For more information, see the section entitled [3.2. Configure Action](#).

However, HDPagination does also support the integration with Tiles view technologies.

11.1. Integrate with Tiles 2

This section focuses on HDPagination's support for Tiles 2 (the standalone version of Tiles). To allow HDPagination to work with Tiles 2 framework, there are 2 available options:

11.1.1. Still use the default view (InternalResourceView)

This view is for JSP file or other internal resource in the web application. Because it's the default view implementation in HDPagination, so you don't have to do any view configuration for you application. However, you probably can not use a JSP file path directly as the value of property 'forwardPath' of Action.

For example, if you are using `org.apache.tiles.web.util.TilesDispatchServlet` to handle all Tiles related request by configuring it in `web.xml` as follows:

```
<servlet>
```

```

    <servlet-name>Tiles Dispatch Servlet</servlet-name>
    <servlet-class>
      org.apache.tiles.web.util.TilesDispatchServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Tiles Dispatch Servlet</servlet-name>
    <url-pattern>*.tiles</url-pattern>
  </servlet-mapping>

```

And the name of definition corresponding to the searching page is "searchProduct". Then the property 'forwardPath' of Action should be specified as "/searchProduct.tiles" as follows:

```

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  <property name="id" value="searchProduct"/>
  <property name="forwardPath" value="/searchProduct.tiles"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>

```

11.1.2. Use Tiles2View

To use Tiles2View, you have to declare it as a bean in the configuration file as follows:

```

<bean id="tiles2View" class="org.hdpagination.web.view.Tiles2View"/>

```

Then set this as the property 'view' of GlobalConfig or Action. The difference between these 2 approaches is that the former takes effect across all the Actions in the application, while the later only applies to that particular Action. Configuring view in GlobalConfig level is as follows:

```

<bean id="globalConfig" class="org.hdpagination.config.GlobalConfig">
  .....
  <property name="view">
    <ref bean="tiles2View"/>
  </property>
</bean>

```

While configuration of view in Action level is like:

```

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  .....
  <property name="view">
    <ref bean="tiles2View"/>
  </property>
</bean>

```

After doing this, the property 'forwardPath' of Action must be set with the name of definition rather than the path of JSP file. For example, if the name of definition corresponding to the searching page is "searchProduct", the action should be configured as follows:

```
<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
    .....
    <property name="forwardPath" value="searchProduct" />
</bean>
```

11.2. Integrate with Tiles 1.x (a.k.a. "Struts Tiles")

This section focuses on HDPagination's support for Tiles 1.x (a.k.a. "Struts Tiles", as shipped with Struts 1.1+). Like Tiles 2, to make HDPagination to work with Tiles 1.x (a.k.a. "Struts Tiles"), there are 2 available options:

11.2.1. Still use the default view (InternalResourceView)

Similar to Tiles 2 (for more information, see the section entitled [11.1.1](#)), so you don't have to do any view configuration for you application. And similarly, do not set the property 'forwardPath' of Action with JSP file path directly. Depending on how you configure struts and tiles to work in your application, you need to set property 'forwardPath' with a correct URL so that page mapped to the tiles definition can be rendered correctly. Think about a web application which uses struts as MVC framework, and the URL mapping is defined in web.xml as follows:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

And the related struts configuration in struts-config.xml is as follows:

```

<action-mappings>
  <action path="/definitionForward"
    type="org.apache.struts.tiles.actions.DefinitionDispatcherAction"
    parameter="def">
    <forward name="success" path="" />
  </action>
</action-mappings>

<plug-in className="org.apache.struts.tiles.TilesPlugin">
  <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml"/>
</plug-in>

```

Suppose the name of definition corresponding to the searching page is "searchProduct". Then the property 'forwardPath' of Action should be specified with the URL "/definitionForward.do?def=searchProduct" as follows:

```

<bean id="searchProductAction" class="eg.web.action.jdbc.SearchProductAction">
  <property name="id" value="searchProduct"/>
  <property name="forwardPath"
    value="/definitionForward.do?def=searchProduct"/>
  <property name="attributeName4Data" value="query_result"/>

  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
</bean>

```

11.2.2. Use TilesView

The way to use TilesView (`org.hdpagination.web.view.TilesView`) is exactly the same as Tiles2View (`org.hdpagination.web.view.Tiles2View`) in [section 11.1.2](#). Just replace Tiles2View with TilesView.

Chapter 12. Clean search result data

If the search result is stored in scope other than "request" (such as "session" or "application" which is very rare), in some cases it may be necessary to clean data from time to time. The utility class `org.hdpagination.web.WebUtils` aims to provide the methods to clean data when necessary. For more information, refer its API document.

Chapter 13. Work with JSF

Since version 1.1, HDPagination starts to support JSF (JavaSever Faces). Using HDPagination in JSF typically requires the following tasks:

- Declaring PaginateDefinition instance
- Creating subclass of PaginateFacesAction

- Declaring subclass of `PaginateFacesAction` in `faces-config.xml`
- Modifying web deployment descriptor (`web.xml`)
- Creating web pages page (JSP or Facelet) using `HDPagination` tags

In the following sections, the above tasks will be described through the process of creating a simple functional use case. The example is to allow the user to search the products and display the result in a paginated way.

13.1. Declaring `PaginateDefinition` instance

Declare a bean instance of `org.hdpagination.web.jsf.PaginateDefinition` in spring application context file (e.g. `hdpagination.xml`, or it can be added to an existing configuration file). `PaginateDefinition` defines the pagination related configuration for each search function and consists of the following properties:

13.1.1. `queryTemplate` (mandatory)

This property specifies the type of query execution engine which represents different types of data access technologies used (e.g. JDBC, Hibernate and JPA). See [Chapter 5](#) for more information about `QueryTemplate`. The configuration snippet is as follows:

```
<bean id="searchProdPage" class="org.hdpagination.web.jsf.PaginateDefinition">
  <property name="queryTemplate">
    <ref bean="queryTemplate"/>
  </property>
  <property name="attributeName4Data" value="products"/>
</bean>

<bean id="queryTemplate" class="org.hdpagination.dataaccess.jdbc.hsqldb.HSQLDBJdbcQueryTemplate">
  <constructor-arg>
    <ref bean="dataSource"/>
  </constructor-arg>
</bean>
```

13.1.2. `forwardPath` (optional)

This property specifies outcome of action method “`execute`” in the subclass of `PaginateFacesAction`. The outcome value eventually determines the JSF navigation flow. As specified by JSF, if the outcome returned after executing the action method is null, the same page is redisplayed, which covers the majority of cases where search input shares the same page with search result. So normally you do not need to set this property. Only for some cases where the page containing the “search submit” button is different to the page displaying the search result, you need to set this property.

13.1.3. attributeName4Data (optional)

This property specifies the name of attribute which holds the search result in the specified scope (e.g. request or session). For example, if the property is set as *'products'*, then the search result (*java.util.List*) will be saved as an attribute *'products'* in request or session, so you can use JSF html tag (*dataTable*) to display the data as follows:

```
<h:dataTable id="requestList" value="#{products}" var="item">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Product No"/>
    </f:facet>
    <h:outputText value="#{item.prodNo}"/>
  </h:column>
  .....
</h:dataTable>
```

This property is optional, and if it's not set then the default value "query_result" is used.

13.1.4. pageSize (optional)

This property specifies how many records should be displayed in one page. Property 'pageSize' of *GlobalConfig* defines the default value for all pages (see [Chapter 7](#) for details about *GlobalConfig*). However if you want the page related to this *PaginateDefinition* to have different page size (records per page) to other pages, this is the place you can set.

13.1.5. dataScope (optional)

Property 'defaultDataScope' of *GlobalConfig* define the default scope ('request', 'session' or 'application') where the search result is to be stored. You can use this property to override the default setting in *GlobalConfig*.

13.1.6. resultHandler (optional)

This property is to set the handler to deal with the search result. Under some special circumstances, it's necessary to save the search result in a custom way rather than relying on framework to save it in the specified scope (request or session). This property is to cater for this requirement, allowing user to plug-in Custom Result Handler implementation class to handle the search result. An example of this would be the integration with Seam to use its Out-jection to store search result.

13.2. Creating Subclass of *PaginateFacesAction*

Create a JSF managed bean which extends from class

`org.hdpagination.web.jsf.PaginateFacesAction`, binding its properties with corresponding search criteria HTML fields in search input page (this is just a standard way of JSF value binding between java bean and UI input).

This JSF managed bean should implement the abstract method `getQueryCallback` specified in class `org.hdpagination.web.jsf.PaginateFacesAction`, where a `QueryCallback` instance is dynamically created based on the values of bean's properties which reflect the search criteria input. A typical way of creating the `QueryCallback` instance is to use anonymous inner class, please refer to the provided example code to find more information about this. And similar to the way `HDPagination` handle JSP application, different type of `QueryCallback` should be used based on underlying data access technologies used (JDBC, Hibernate or JPA). It is configured by the 'queryTemplate' property of `PaginateDefinition` (see [13.1.1](#)). Refer to [Chapter 4](#) for more information about different types of `QueryCallback`. Here is an example that shows how to create a subclass of `PaginateFacesAction`:

```
public class SearchProdAction extends PaginateFacesAction {
    private String prodNo;
    private String madeIn;
    private String desc;

    public String getProdNo() {
        return prodNo;
    }

    public void setProdNo(String prodNo) {
        this.prodNo = prodNo;
    }

    public String getMadeIn() {
        return madeIn;
    }

    public void setMadeIn(String madeIn) {
        this.madeIn = madeIn;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public QueryCallback getQueryCallback() {
        StringBuffer queryStmt = new StringBuffer(" select
            productId,prodNo,price,madeIn,description from product p where 1=1 ");
        if (prodNo != null && !prodNo.trim().equals("")) {
            queryStmt.append(" and prodNo = ? ");
        }
        if (madeIn != null && !madeIn.trim().equals("")) {
            queryStmt.append(" and madeIn = ? ");
        }
        if (desc != null && !desc.trim().equals("")) {
            queryStmt.append(" and description = ? ");
        }
        AbstractJdbcQueryCallback callback =
            new AbstractJdbcQueryCallback(queryStmt.toString(), "p.madeIn") {
                public void setValues(PreparedStatement ps) throws SQLException {
```

```

int pos = 1;
if (prodNo != null && !prodNo.trim().equals("")) {
    ps.setString(pos++, prodNo);
}
if (madeIn != null && !madeIn.trim().equals("")) {
    ps.setString(pos++, madeIn);
}
if (desc != null && !desc.trim().equals("")) {
    ps.setString(pos++, desc);
}
}

public Object processRow(ResultSet rs) throws SQLException {
    ProductVO prod = new ProductVO();
    prod.setProductId(new Long(rs.getLong("productId")));
    prod.setPrice(new Integer(rs.getInt("price")));
    prod.setProdNo(rs.getString("prodNo"));
    prod.setMadeIn(rs.getString("madeIn"));
    prod.setDescription(rs.getString("description"));

    return prod;
}
};
return callback;
}
}

```

Under some special circumstances, it may be necessary to perform cross field validation (validate multiple fields). In that case, the method `validate(FacesContext context)` should be overridden to perform the custom validation. When validation fails, the method should return false and error messages should be added to `FacesContext` (just as the normal way to add error messages in a JSF application). The sample code snippet is as follows:

```

protected boolean validate(FacesContext context) {
    if ("USA".equals(this.madeIn) && "USA".equals(this.desc) ){
        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
            "made in can not equal to desc", "made in can not equal to desc");
        context.addMessage(null, message);
        return false;
    } else {
        return true;
    }
}
}

```

13.3. Declaring Subclass of `PaginateFacesAction` in `faces-config.xml`

Declare the java bean created in [13.2](#) as a JSF managed bean in the file “`faces-config.xml`”, and inject the corresponding instance of `PaginateDefinition` (defined in Spring application context) into that java bean. To allow JSF to resolve its variables from the Spring application context, the default JSF variable resolver need to be replaced by Spring’s `DelegatingVariableResolver`. The snippet of “`faces-config.xml`” is as follows:

```

<managed-bean>
  <managed-bean-name>searchProdAction</managed-bean-name>
  <managed-bean-class>eg.web.jsf.SearchProdAction</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>paginateDefinition</property-name>
    <value>#{searchProdPage}</value>
  </managed-property>
</managed-bean>
..
..
..
<application>
  <variable-resolver>
    org.springframework.web.jsf.DelegatingVariableResolver
  </variable-resolver>
</application>

```

For more information about the integration of Spring with JSF, and Spring's `DelegatingVariableResolver`, refer to the reference document from Spring framework.

13.4. Modifying web deployment descriptor (*web.xml*)

Nothing special about `HDPagination` needs to be added to `web.xml`. All you need to define is the `FacesServlet` (like the way normal JSF applications do), Spring's `ContextLoaderListener` and “`contextConfigLocation`” context-param (allowing Spring to load `HDPagination` related configuration file). A sample snippet of `web.xml` is as follows:

```

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/hdpagination.xml
    /WEB-INF/dataAccessContext-hsqldb.xml
  </param-value>
</context-param>

<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<servlet id="FacesServlet">
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.faces</url-pattern>
</servlet-mapping>

```

13.5. Create page (JSP or Facelet)

Now, you can start to use HDPagination JSF tag libraries to build your page. Usually, for a JSF application, you can choose JSP or Facelet to display the page. HDPagination tags have been tested to work for both view technologies. Follow the steps listed bellow to implement pagination search page:

13.5.1. Include HDPagination JSF tags definition

- If JSP is used to view the page, add the following directive on top of the JSP file:

```
<%@ taglib uri="http://www.hdpagination.org/tags/jsf" prefix="hdp" %>
```

Here, <http://www.hdpagination.org/tags/jsf> represents the unique URI for HDPagination JSF tag (JSP view)

- If Facelet is used to view the page, define the root html tag in XHTML file as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:hdp="http://www.hdpagination.org/tags/facelet">
```

Here, <http://www.hdpagination.org/tags/facelet> represents the unique URI for HDPagination JSF tag (Facelet view)

13.5.2. Bind action properties to HTML fields

Bind the values of corresponding search criteria HTML fields in search input page to the properties defined in the subclass of `PaginateFacesAction` (created in [13.2](#)) through Unified Expression Language (EL). This is just a standard way of JSF value binding between java bean properties and UI input, nothing special to HDPagination. The snippet of sample code is as follows:

```

<h:form id="form" >
<table width="80%" border="1">
  <tr valign="baseline">
    <td>Product No:</td>
    <td>
      <h:inputText id="prodNo" value="#{searchProdAction.prodNo}"/>
    </td>
    <td>Made In:</td>
    <td>
      <h:inputText id="madeIn" value="#{searchProdAction.madeIn}">
        <f:validateLength maximum="5"/>
      </h:inputText>
    </td>
    <td>Description</td>
    <td>
      <h:inputText id="description" value="#{searchProdAction.desc}"/>
    </td>
  </tr>

  <tr valign="baseline">
    <td colspan="4">
      <h:commandButton action="#{searchProdAction.execute}" value="Search" />
    </td>
  </tr>
</table>
...
..

```

13.5.3. Invoke action method of PaginateFacesAction

Use Method Binding Expression to associate the “action” property of `commandButton` or `commandLink` with the “execute” method of `PaginateFacesAction` subclass you created in [13.2](#). Sample snippet code is as follows:

```

<h:commandButton action="#{searchProdAction.execute}" value="Search" />

```

13.5.4. Display search result in table

Use `<h:dataTable>` or other extended JSF `dataTable` tags to display the search result. This is purely JSF related and all you need to be aware of is the attribute name of search result (`java.util.List`) which is saved by HDPagination framework in the specified scope (‘request’ or `PaginateDefinition` ‘session’). For example, if property “attributeName4Data” of `PaginateDefinition` is set as “products”, then the snippet code will be as follows:

```

<h:dataTable id="requestList" value="#{products}" var="item">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Product No"/>
    </f:facet>
    <h:outputText value="#{item.prodNo}"/>
  </h:column>
  .....
```

```
</h:dataTable>
```

13.5.5. Use JSF tags

HDPagination ships with some useful pagination tags for JSF application. **To make them work, they must be used inside html form tag (e.g. <h:form>).** The way to use tags is as follows:

```
<hdp:paginateLink pagiDefinitionId="searchProdPage" isPaginatedVar="_isPaginated"
  pageIndexVar="_pageIndex" pagesVar="_pages"
  recordsVar="_records" firstResultVar="_firstResult"
  lastResultVar="_lastResult" />
<br/><br/>
<hdp:selectPage pagiDefinitionId="searchProdPage" />
<br/><br/>
<hdp:goToPage pagiDefinitionId="searchProdPage" buttonLabel="Goto" />
<br/><br/>
<h:panelGroup rendered="#{_isPaginated}">
  <h:outputText value="Page " />
  <h:outputText value="#{_pageIndex}" />
  <h:outputText value=" of " />
  <h:outputText value="#{_pages}" />
  <h:outputText value=" Records " />
  <h:outputText value="#{_firstResult}" />
  <h:outputText value=" - " />
  <h:outputText value="#{_lastResult}" />
  <h:outputText value=" of total " />
  <h:outputText value="#{_records}" />
  <h:outputText value=" records" />
</h:panelGroup>
```

The next chapter ([Chapter 14](#)) describes more details about tags.

13.6. Change page size from Page UI

Since version 1.2, a new property 'pageSize' was added to the class `PaginateFacesAction`. This allows the user to change the page size (number of records per page) from the searching input page.

Use value binding expression to associate this property with a UI Component's value in the searching input page. If this property is not associated with any UI Component in the searching input page, the page size defined in `PaginateDefinition`, the page size defined in `GlobalConfig`, or the default value (10) will be used.

The way to associate this property with a UI Component's value is as follows:

```
<h:selectOneMenu id="pageSize" value="#{searchProdAction.pageSize}">
  <f:selectItems value="#{selectItemsBean.pageSizeSelectItems}" />
</h:selectOneMenu>
```

Description of JSF tags `h:selectOneMenu` and `f:selectItems` is beyond the scope of this document. Please refer to JSF taglib document for details.

The managed bean used in the above snippet to populate HTML select options is as follows:

```
public class SelectItemsBean {  
  
    public List getPageSizeSelectItems() {  
        List items = new ArrayList();  
        items.add(new SelectItem(Integer.valueOf("3"), "3"));  
        items.add(new SelectItem(Integer.valueOf("5"), "5"));  
        items.add(new SelectItem(Integer.valueOf("10"), "10"));  
  
        return items;  
    }  
}
```

Chapter 14. Pagination Tags for JSF

14.1. `<hdp:paginateLink>`

14.1.1. Description

This JSF Tag is used to render pagination navigation links (e.g. Previous Next 1 2 3 4 5 6 7 8 First Last).

14.1.2. Attributes

Refer to the JSF taglib document (which is under the directory “doc/jsf-taglib” after unzipping the distribution).

14.1.3. How to use

The way to use this tag in page is as follows:

```
<hdp:paginateLink pagiDefinitionId="searchProdPage"  
    isPaginatedVar="_isPaginated" pageIndexVar="_pageIndex"  
    pagesVar="_pages" recordsVar="_records" firstResultVar="_firstResult"  
    lastResultVar="_lastResult" />  
<br/><br/>  
<h:panelGroup rendered="#{_isPaginated}">  
    <h:outputText value="Page " />  
    <h:outputText value="#{_pageIndex}" />  
    <h:outputText value=" of " />  
    <h:outputText value="#{_pages}" />  
    <h:outputText value=" Records " />  
    <h:outputText value="#{_firstResult}" />  
    <h:outputText value=" - " />  
    <h:outputText value="#{_lastResult}" />  
    <h:outputText value=" of total " />
```

```
<h:outputText value="#{_records}" />
<h:outputText value=" records" />
</h:panelGroup>
```

14.1.4. Configure PageLinksPattern

This is the same as the tag (pageLinks) for JSP. See [9.1.4](#) for detailed information.

One thing different to JSP application is that:

If special HTML characters (e.g. “<” and “>”) are used in ResourceBundle file (which is defined as property “resourceBundleName” of PageLinksPattern), then property “escapeHtmlChar” should set as false specifically as follows:

```
<bean id="pageLinksPatternYahoo" class="org.hdpagination.config.pattern.YahooPageLinksPattern">
  <constructor-arg type="int" value="10"/>
  <property name="resourceBundleName" value="eg.web.jsf.PaginalLinks"/>
  <property name="displayFirstPageLink" value="true"/>
  <property name="displayLastPageLink" value="true"/>
  <property name="escapeHtmlChar" value="false"/>
</bean>
```

14.2. <hdp:sortLink>

14.2.1. Description

This JSF Tag is used to render the column header with hyperlink (<a>) in a search results table. This allows the user to sort the search results by this column.

By default, when the column header is clicked and the current search result is not sorted by that column, then the result will be displayed in ascending order. The search results will be toggled between ascending and descending order every time the column header is clicked.

14.2.2. Attributes

Refer to the JSF taglib document (which is under the directory “doc/jsf-taglib” after unzipping the distribution).

14.2.3. How to use

The way to use this tag in page is as follows:

```

<h:dataTable id="requestList" value="#{products}" var="item">
    ..
<h:column>
    <f:facet name="header">
        <hdp:sortLink pagiDefinitionId="searchProdPage" sortedColumn="p.price" value="Price"/>
    </f:facet>
    <h:outputText value = "#{item.price}" />
</h:column>

<h:column>
    <f:facet name="header">
        <hdp:sortLink pagiDefinitionId="searchProdPage" sortedColumn="p.madeIn" value="Made In"/>
    </f:facet>
    <h:outputText value = "#{item.madeIn}" />
</h:column>
    ..
</h:dataTable>

```

14.2.4. Support for internationalization and localization

Since all its attributes except ‘sortedColumn’ accept EL expressions, it’s easy to render the column header (title) as a localization aware message. For example, you can do as follows:

Add the following line to the top of JSP or XHTML file:

```
<f:loadBundle basename="eg.web.jsf.LocalizationResources" var="bundle"/>
```

Then use value binding expression to assign the bundle message to ‘value’ attribute of `<hdp:sortLink>` tag:

```
<hdp:sortLink pagiDefinitionId="searchProdPage" sortedColumn="p.price" value="#{bundle.price}"/>
```

Here, ‘price’ is the message key defined in resource bundle file ‘eg.web.jsf.LocalizationResources’.

14.3. `<hdp:selectSort>`

14.3.1. Description

This JSF tag is used to render a dropdown (`<select>`) to allow the user to sort the search results by specified column in either ascending or descending order.

This tag is an alternative of the [<hdp:sortLink>](#) tag which applies a sorting hyperlink to the column header in a search results table. This tag can be useful in some cases

where there is no column header in the search results table (as a result you can not apply the [<hdp:sortLink>](#) tag to the column header).

The `<hdp:selectSort>` tag should be used together with `<hdp:sortOption>` tag. `<hdp:sortOption>` tags must be nested inside `<hdp:selectSort>` tag and each `<hdp:sortOption>` will render as a `<option>` HTML tag.

14.3.2. Attributes

Refer to the JSF taglib document (which is under the directory “doc/jsf-taglib” after unzipping the distribution).

14.3.3. How to use

The way to use this tag in page is as follows:

```
<hdp:selectSort pagiDefinitionId="searchProdPage">
  <hdp:sortOption sortedColumn="p.price" ascending="true" label="Price (Low - High)"/>
  <hdp:sortOption sortedColumn="p.price" ascending="false" label="Price (High - Low)"/>
  <hdp:sortOption sortedColumn="p.madeIn" label="Made In"/>
  <hdp:sortOption sortedColumn="p.description" label="Description"/>
</hdp:selectSort>
```

14.3.4. Support for internationalization and localization

Similar to `<hdp:sortLink>` tag, you can use value binding expression to assign the bundle message to ‘label’ attribute of `<hdp:sortOption>` tag:

```
<hdp:selectSort pagiDefinitionId="searchProdPage">
  <hdp:sortOption sortedColumn="p.price" ascending="true" label="#{bundle.priceLowHigh}"/>
  <hdp:sortOption sortedColumn="p.price" ascending="false" label="#{bundle.priceHighLow}"/>
  ...
</hdp:selectSort>
```

[See 14.2.4](#)

14.4. `<hdp:selectPage>`

14.4.1. Description

This JSF Tag is used to render a dropdown (`<select>`) with all available pages, which allows the user to navigate to different pages.

14.4.2. Attributes

Refer to the JSF taglib document (which is under the directory “doc/jsf-taglib” after unzipping the distribution).

14.4.3. How to use

The way to use this tag in page is as follows:

```
<hdp:selectPage pagiDefinitionId="searchProdPage" />
```

14.5. <hdp:goToPage>

14.5.1. Description

This JSF Tag is used to render a text box along with either a submit button, hyperlink, or image link. The button, hyperlink, or image link is used to submit the request and navigate to the page specified in the text box.

The decision to render the button, hyperlink, or image link relies on the following rule:

- If property 'imgSrc' is specified, then an image link (tag with enclosing <a> tag) is rendered.
- Otherwise, if property 'linkLabel' is specified, the hyperlink (text with enclosing tag <a>) is rendered.
- If none of these are specified, the button is rendered by default.

14.5.2. Attributes

Refer to the JSF taglib document (which is under the directory “doc/jsf-taglib” after unzipping the distribution).

14.5.3. How to use

The way to use this tag in page is as follows:

```
<hdp:goToPage pagiDefinitionId="searchProdPage" buttonLabel="Goto" />
```

14.5.4. Support for internationalization and localization

Similar to `<hdp:sortLink>` tag, you can use value binding expression to assign the bundle message to attributes `'invalidPageAlert'`, `'exceedMaxPageAlert'`, `'buttonLabel'`, and `'linkLabel'`:

```
<hdp:goToPage pagiDefinitionId="searchProdPage" buttonLabel="#{bundle.goTo}"
  invalidPageAlert="#{bundle.invalidPageNo}" exceedMaxPageAlert="#{bundle.pageNoExceeds}"/>
```

[See 14.2.4](#)

Chapter 15. Portlet support

HDPagination supports JSF application running under Portlet environment whether it's Portlet 1.0 (JSR 168) or Portlet 2.0 (JSR 286). All its JSF components and tags were tested to be running successfully under the standard Portlet Container (such as Pluto, Jetspeed and GlassFish Portal Server etc).

To run the JSF web application under Portlet Container, you may need a JSF Portlet Bridge to abridge the gap between the JSF request/response and Portlet request/response processing lifecycle. The discussion about JSF Portlet Bridge is beyond the scope of this document, there are a few open source projects addressing this issue and go to their websites to get more information.

Unfortunately, HDPagination does not support Non-JSF (JSP version) application for Portlet environment.

Chapter 16. Supported JDK/J2EE/JSF version

16.1. JDK version

JDK 1.4 or later version is required to use HDPagination library. For application servers which is still using JDK1.4 (e.g. Webshphere 6.0), the jar named as "hdpagination-for-jdk14.jar" should be used. For application servers using JDK 1.5 or later version, the jar named as "hdpagination.jar" should be used.

Please note, if you intend to integrate with Tiles 2, JDK 1.5 or later is required, which is required by Tiles 2.

16.2. J2EE version

HDPagination framework needs:

- JSP 1.2 or later version
- Servlet 2.3 or later version

16.3. JSF version

If your application is still using JSF1.0, please use the jar named as “hdpagination-for-jsf1.0.jar”. If JSF1.1, JSF1.2, or later version is used in your application, the jar named as “hdpagination.jar” should be used.

16.4. Spring version

No specific requirement for Spring framework version. HDPagination library can be working in any version of Spring framework.