EU-IST Project IST-2003-506826 SEKT
SEKT: Semantically Enabled Knowledge Technologies

D6.3 Interface Definition

Barbara S. Hellbarth-Busch, empolis GmbH
Ralph Traphöner, empolis GmbH

Abstract
This document provides instructional informatioon on how to develop clients on top of the SEKT Integration Platform utilising JSP technology. It is targeted towards a technical audience that has experience in developing JSP applications.

Keyword list: API, JSP Tag Library, Instructions

WP6 Integration
Report PU
Contractual date of delivery 31.12.04 Actual date of delivery 31.1.05

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540
Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592
Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891
Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

University of Innsbruck
Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475
Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006
Madrid
Spain
Tel: +34 913 349 797
Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00
Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912
Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Sirma AI EAD, Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vall` es)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

This document provides instructional informatioon on how to develop clients on top of the SEKT Integration Platform utilising JSP technology. It is targeted towards a technical audience that has experience in developing JSP applications.

Readers looking for instructions on how to implement pipelets for the SEKT Integration Platform are referred to deliverable D6.4 Initial Platform.
.

Contents

1 Glossary and Abbreviations

API	Application Programming Interface
HTML	Hypertext Markup Language
JAR	Archiv format
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
JSTL	JavaServer Pages Tag Library
TLD	Tag Library Description

2 Tag Libraries

Tag Libraries are part and parcel of the JavaServer™ Pages (JSP) specification. Using JSP makes it possible to develop dynamic web pages by using minimal Java code. With the help of Tag Libraries it is much easier to develop dynamic web pages. A Tag Library is a collection of tag classes and a Tag Library Description (TLD). Tag classes are Java classes which implement a dedicated interface.
Taglibs are the mechanism with JSP which allow HTML tags to be associated with Java classes. The JavaServer Pages Tag Library (JSTL) encapsulates as simple tags the core functionality common to many Web applications.

2.1 TagLib specification

For the implementation it is helpful to use as much as possible tags defined in JSTL[1].
Since the specification release JSP 2.0 the *Expression Language* has become a vital part of JSP. *Expression Language* is a powerful language that can be used to perform various actions (conditional, iterator, URL-related…).
So, the number of SEKT tags is small to keep the SEKT TagLib manageable and for an easy use. The use of JSTL facilitates the management of the e:km TagLib.
It is mandatory to use the latest specifications, especially JSP 2.0 and JSTL 1.1. This is important because only in the latest specifications of JSP and JSTL the tight collaboration of the two has achieved a clear partition of functionality and responsibility.

[1] JSTL — JavaServer Pages Standard Tag Library

3 Using the TagLib

For configuring your Web application with the SEKT Tag Library the following steps have to be executed:

Copy the Tag Library descriptor file "ekmTaglib.tld" to the sub-directory of your Web application named /WEB-INF.

Copy the Tag Library JAR files "orenge.jar" and "external.jar" to the sub-directory of your Web application named /WEB-INF/lib.

Copy the properties file "ias.properties" to the sub-directory of your Web application named /WEB-INF/classes and set your JNDI properties in this file.

Note
These informations are necessary to set up the connection between the client and the query server. The JNDI properties have to be set, because a JNDI server bridges the connetion between client and server.

Optionally copy the properties file "api.properties" to the sub directory of your Web application named /WEB-INF/classes and set the SIP model properties in this file.

Add a <taglib> element to your Web application deployment descriptor in /WEB-INF/web.xml like this:

```
<taglib>
<taglib-uri>
http://www.empolis.com/taglib/ekmTaglib.tld
</taglib-uri>
<taglib-location>
/WEB-INF/ekmTaglib.tld
</taglib-location>
</taglib>
```

To use the tags from this library in your JSP pages, add the following directive at the top of each page:

```
<%@ taglib uri=" http://www.empolis.com/taglib/ekmTaglib.tld "
prefix="ekm" %>
```

where "SEKT" is the tag name prefix you should use for the tags from this library. You can change this value to any prefix you like.
To use log4j follow these steps:

Copy the properties file "log4j.properties" to the subdirectory of your web application named/WEB-INF/classes and if you like set your individual log4j properties.

3.2 Required Libraries

The following JAR-files are required in folder /WEB-INF/lib to use the SEKT Tag Library:

orenge.jar, external.jar
containing all classfiles that are needed for SIP

jstl.jar, standard.jar
to support JSTL

`jbossall-client.jar` (only orenge 5.x)
to support JNDI services

4 Configuration Files

In this section the configuration of the SIP API[2] and the SIP file, which contains all access information for the search engine, is described.

4.2 API configuration file

The SIP TagLib uses the SIP API to communicate with the search engine. So you can use the API configuration file "`api.properties`" to set some global search parameters for your Web application. This configuration file has to be placed in the sub-directory of the Web application named `WEB-INF/classes`. All these properties settings and even the usage of this file are optional.

locale
sets the default language of the search results

Example
```
locale=en_US
```

request.pipeline
the pipeline that is used for request if no other pipeline is given

Example
```
request.pipeline=MainPipe
```

request.aggregate
the query class that is used for the request if no other query class is given

Example
```
request.aggregate=Recipe.V1
```

request.parameter.general
set some general parameters for the query

Example 1
```
request.parameter.general.language=de
```
Sets general parameter language to value 'de'

Example 2
```
request.parameter.general.printResult=append
```
Sets general parameter printResult to value 'append'

Example 3
```
request.parameter.general.languages.1=de
request.parameter.general.languages.2=en
```
Set the general parameters language to values 'de' and 'en'

request.parameter.pipelet
sets pipelet parameters for the query

Example 1
```
request.parameter.pipelet.TermFinderPipelet.maxCount=1000
```
Sets parameter maxCount for Pipelet `TermFinderPipelet` to value '1000'

Example 2
```
request.parameter.pipelet.multiRetrieverPipelet.useRetrievers.1=smar
tcookingIndexPipelet
request.parameter.pipelet.multiRetrieverPipelet.useRetrievers.2=smar
tcookingIndexDEPipelet
```
Sets parameter useRetrievers for pipelet multiRetrieverPipelet to values 'smartcookingIndexPipelet' and 'smartcookingIndexDEPipelet'

[2] API — Application Programming Interface

response.pipelet
set the pipelet-ids for the main services (textminer, retrieval, dialog, decisiontree)

Example 1
```
response.pipelet.textminer=textminerPipelet
```
The pipelet-id for the textminer service is "`textminerPipelet`".
Example 2
```
response.pipelet.resultlist=multiRetrieverPipelet
```
The pipelet-id for the retrieval service is "`multiRetrieverPipelet`".
Example 3
```
response.pipelet.dialog=dialogPipelet
```
The pipelet-id for the dialog service is "`dialogPipelet`".
Example 4
```
response.pipelet.decisiontree=decisionTreePipelet
```
The pipelet-id for the decisiontree service is "`decisionTreePipelet`".

format.date, **format.time**, **format.timestamp**
set the output-format of date, time, and timestamp values as patterns

Possible values:

Pattern	Output
none	2004-08-09
short	8/9/04
medium	Aug 9, 2004
long	August 9, 2004

Pattern	Output
"#yyy.MM.dd G 'at' HH:mm:ss z"	2001.0.04 AD at 00:00:00 PDT
"#EEE, MMM d, "yy"	Wed, Jul 4, '01
"#h:mm a"	00:00 PM
"#hh 'o''clock' a,zzzz"	00 o'clock PM, Pacific Daylight Time
"#K:mm a, z"	0:00 PM, PDT
"#yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 00:00 PM
"EEE,d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 00:00:00 – 0700
"yyMMddHHmmssZ"	010704000000-0700

Examples
```
format.date=#dd.MM.yyyy
format.time=#HH:mm:ss
format.timestamp=long
```

format.date.parse.pattern
Set a list of patterns that are recognized and parsed as a date when inputting date-, time- or timestamp-values

Examples

```
format.date.parse.pattern.1=M/d/yy
format.date.parse.pattern.2=MM/dd/yy
format.date.parse.pattern.3=M/d/yyyy
format.date.parse.pattern.4=MM/dd/yy
format.date.parse.pattern.5=d.M.yy
format.date.parse.pattern.6=dd.MM.yy
format.date.parse.pattern.7=d.M.yyyy
format.date.parse.pattern.8=dd.MM.yyyy
format.date.parse.pattern.9=MMM d, yyyy
format.date.parse.pattern.10=MMMMM d, yyyy
format.date.parse.pattern.11=EEE, MMM d, ''yy
format.date.parse.pattern.12=EEE, d MMM yyyy
format.date.parse.pattern.13=EEE MMM dd HH:mm:ss z yyyy
```

format.number
Set the output-format of number-values.
Possible values:

Pattern	Output
true	1,561.00 (English) or 1.561,00 (German)
false	1561.00 (English) or 1561,00 (German)

Example
```
number.format=true
```

4.2 SIP configuration file

The SIP configuration file contains access information for the search engine. The following properties have to be set for connecting the search engine. The required information is different in orenge 4.x and orenge 5.x.

4.2.1 orenge 4.x

empolis.ias.host
sets the host of the query server as servername or IP

Example
```
empolis.ias.host=127.0.0.1
```

empolis.ias.port
sets the port of the query server

Example
```
empolis.ias.port=6201
```

empolis.ias.maxrepeats
sets the number of repeats, when the connection fails

Example
```
empolis.ias.maxrepeats=3
```

empolis.ias.filepipeline
sets the name of the pipeline to fetch the SIP model from

Example
```
empolis.ias.filepipeline=FileAccessPipeline
```

4.2.2 SIP 5.x

java.naming.factory.inital
sets the JNDI initial context factory

Example

```
java.naming.factory.inital=org.jnp.interface.NamingContextFactory
```

java.naming.provider.url
sets the JNDI context for ProcessManager lookup

Example

```
java.naming.provider.url=jnp://afa-07468:1099/SmartcookingServices
```

empolis.ias.pmgr
sets the JNDI name of ProcessManager

Example

```
empolis.ias.pmgrAlone
```

Add names of failover ProcessManagers like this:

```
empolis.ias.pmgr.0=pmgr2
empolis.ias.pmgr.1=yetAnotherPmgr
```

empolis.ias.maxrepeats
sets the number of repeats, when the connection fails

Example

```
empolis.ias.maxrepeats=3
```

empolis.ias.filepipeline
sets the name of the pipeline to fetch the SIP model from

Example

```
empolis.ias.filepipeline=FileAccessPipeline
```

5 Request Manager Bean

The RequestManager Bean is a helper to create and execute search requests. To achieve this it reads the properties from the configuration files 'api.properties' and 'ias.properties'.
The RequestManager Bean is an instance of com.empolis.orenge.api.RequestManager and therefore is initialized in the JSP-page like this:

```
<jsp:useBean
id="rm"
scope="request"
class="com.empolis.orenge.api.RequestManager" />
```

The api- and ias-configuration files have to be set as bean properties:

```
      <jsp:setProperty
name="rm"
property="apiConfigurationFile"
value="/api.properties" />
      <jsp:setProperty
name="rm"
property="propertyFile"
value="/ias.properties" />
```

Please set these two properties in this order always:

1. apiConfigurationFile (optional)
2. propertyFile (required)

6 Supported Tags

The available tags can be divided into three main groups: set-, get- and print-tags. Additionally there are two special tags for creating and executing the actual request.

6.1 Tags for Creating and Executing the Request

- createRequest
- executeRequest

6.2 Set-Tags

All set-tags are used to set query information like filters, attribute values, and (pipelet) parameters. This information is stored in the request object which is used by the requestmanager bean to generate the actual search query. So these tags have to be processed **before** executing the request.

- setGeneralParameterValue
- setPipeletParameterValue
- setAttributeValue
- setIntervalFilter
- setEnumerationFilter
- setHasElementsFilter
- setImportance
- setUseMeasure
- setExpandwildcards

6.2.1 Resolving Values in Set-Tags

The main task of all set-tags is to deliver a given value to the request object. The attribute *'source'* is required in most of the set-tags (we have the attributes *'min'* and *'max'* instead of *'source'* in setIntervalFilter). The given string-value is used as an identifier to find the actual value either as *request-parameter* or as *context-attribute*.
This means, that this identifier refers to the request-parameter returned by a call to `ServletRequest.getParameter(source)` or — if no value was found — to the JSP scoped variable returned by a call to `PageContext.findAttribute(source)`. The received value can reside in request-parameters or any of the four JSP scopes: *page*, *request*, *session*, or *application*, whereas the first hit will be returned.

6.3 Get-Tags

Get-tags provide opportunities to receive data and store it within an individually named variable. The get-tags provide information about the current search result, so they need to be called **after** executing the search request.

- getQueryObject
- getInput
- getResult
- getDialogResult

- getDialogAnswersForQuestion
- getAnalysisMisspelledWords
- getAnalysisSuggestions
- getAttributeValue
- getPropertyValue
- getSimilarTerms (only SIP.5.x)
- getTotalNumberOfHits

6.4 Print-Tags

Print-tags also deliver data but instead of storing it to a variable the data is immediately printed out to the HTML output of the JSP.

- printAttributeValue
- printAttributeMarkup
- printConcept
- printPropertyValue
- printAnalysisMisspelledWord
- printAnalysisSuggestion
- checkDialogAnswerType (is used in combination with printDialog…Answer tags)
- printDialogEnumerationAnswer
- printDialogIntervalAnswer
- printSearchcontext
- printRetrievalSim
- printTotalNumberOfHits

7 Description of Tags

7.1 Creating and Executing the Request createRequest

This tag creates a request object (instance of *com.empolis.orenge.api.Request*) and writes it to the page scoped variable with the name given in tag attribute *'var'*. If attributes *'pipeline'*, *'queryClass', 'serviceClass'* or *'view'* are given, the respective values are set for the actual request, whereas *'pipeline'* is the name of the search pipeline, *'queryClass'* is the name of the aggregate for the query object, *'serviceClass'* is the name of the aggregate for the service-object, and *'view'* is the name of the used view. If *'pipeline'* and *'queryClass'* are not given, the respective values from API configuration file will be used.

Attribute	Use	Default	Description	Value(s)
requestManager	required	—	RequestManagerBean, that handles the requests	RequestManagerBean
var	required	—	Name for the identifier of the received result	String
pipeline	optional	—	Name of the pipeline used for the search request	String
queryClass	optional	—	Name of the aggregate used for the search request's query-object	String
serviceClass	optional	Same as queryClass	Name of the aggregate used for the search request's service-object	String
view	optional	—	Name of the view used for the search request	String

Example
Samplecode in processing page

```
<jsp:useBean
id="rm"
scope="request"
class="com.empolis.orenge.api.RequestManager">
[…]
</jsp:useBean>
<ekm:createRequest
requestManager ="${rm}"
var="request" />
```

This tag creates a new request-object, with pipeline and queryClass given in API configuration file.

executeRequest

After having set all information for the query (see section "Setting the Query (Set-Tags)") we can execute the search request by calling the tag *executeRequest* with requestmanager and request as tag attributes. The resulting response object (instance of *com.empolis.orenge.api.Response*) is written to the variable with the name given in attribute *'var'*.

Attribute	Use	Default	Description	Value(s)
requestManager	required	—	RequestManagerBean, that handles the requests	RequestManagerBean

request	required	—	The request-object that represents the query	Request-Object
var	required	—	Name for the identifier of the received result	String

Example
Samplecode in processing page

```
<jsp:useBean
id="rm"
scope="request"
class="com.empolis.orenge.api.RequestManager">
[…]
</jsp:useBean>
<ekm:createRequest
requestManager ="${rm}"
var="request" />
```

(pass query information to request-object (see section “Setting the Query (Set-Tags)”)

```
<executeRequest var="response" requestManager="${rm}"
request="${request}" />
```

In this example a query with the information from request object *'request'* is sent to the server. After that the received results are available over the response object *'response'*.

7.2 Setting the Query (Set-tags)

The customizing of queries is divided in the setting of parameters, attributes, filters, dynamic weighting, similarity measures, and wildcards.

7.2.1 Parameters

In each query we can set pipelet parameters either for single pipelets or for all pipelets in a pipeline at once.
In other words parameters of pipelets such as *ServicePipelets*, *Orengelets,* and *Conditionlets* can be changed or set for the current retrieval in the pipeline.

setGeneralParameterValue
This tag passes a value for a named parameter to the request object (instance of *com.empolis.orenge.api.Request*).
This means that the given value applies to all pipelets that use the actual parameter.

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
Param	required	—	Name of the parameter that has to be set	String
source	optional	—	Name of the request-parameter or context-attribute that represents the actual value for the parameter.	String

Example
Samplecode in searchform

```
<input type="hidden" name="defLanguage" value="de"/>
```

Samplecode in processing page

```
<ekm:createRequest
requestManager ="${rm}"
      var="request" />
[…]
<ekm:setGeneralParameterValue
request="${request}"
param="language"
source="defLanguage" />
```

This tag sets the parameter *'language'* for all pipelets to the value that is specified by the identifier in the source attribute. So in this example it will be resolved to *'de'* (see section "Resolving Values in Set-Tags").
The resolved value should be a string, a collection of strings or a string array.
If the identifier does not exist in any of the legal scopes the parameter will not be passed to request object.

Example
Samplecode in searchform

```
<input type="hidden" name="threshold" value="0.5" />
```

Samplecode in processing page:

```
<ekm:createRequest
requestManager ="${rm}"
     var="request" />
[…]
<ekm:setGeneralParameterValue
request="${request}"
param="threshold" />
```

This tag sets the parameter *'threshold'* for all pipelets. As no *'source'*-attribute is given in the tag, the name of the actual parameter is used as identifier for the value. So in this example it will be resolved to *'0.5'*(see section "Resolving Values in Set-Tags").
If the identifier does not exist in any of the legal scopes the parameter will not be passed to request-object.

setPipeletParameterValue
This tag passes a value for a named pipelet parameter to the request object (instance of *com.empolis.orenge.api.Request*).
This means that the actual value only applies to the given pipelet.

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
pipeletID	required	—	Name of pipelet	String
param	required	—	Name of the parameter that has to be set.	String
source	optional	—	Name of the request-parameter or context-attribute that represents the actual value for the parameter	String

Example
Samplecode in searchform

```
<input type="hidden" name="retrievers"
value="smartcookingIndexPipelet"/>
<input type="hidden" name="retrievers"
value="smartcookingIndexDEPipelet"/>
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setPipeletParameterValue
request="${request}"
pipeletID="multiRetrieverPipelet"
param="useRetrievers"
source="retrievers" />
```

This tag sets the parameter *'useRetrievers'* for pipelet "*multiRetrieverPipelet*" to the value that is specified by the identifier in the source attribute. So in this example it will be resolved to values *'smartcookingIndexPipelet'* and *'smartcookingIndexDEPipelet'* (see section "Resolving Values in Set-Tags").
The resolved value should be a string, a collection of strings or a string array.
If the identifier does not exist in any of the legal scopes the parameter will not be passed to request object.
If no *'source'*-attribute was given, the name of the parameter (in this case *'useRetrievers'*) would be used as identifier for the actual value.

7.2.2 Attributes

We can assign values to attributes by using setAttributeValue-tag.

setAttributeValue
This tag passes a value for an index attribute to the request-object (instance of *com.empolis.orenge.api.Request*). This tag is usually used to pass the actual query text of the query.

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
attr	required	—	Name of the index-attribute	String
source	optional	—	Name of the request-parameter or context-attribute that represents the actual value for the parameter	String
target	optional	'queryObj'	Inforamtion if the attribute has to be set for the request's query-object or service-object.	'queryObj' or 'serviceObj'

Example
Samplecode in searchform

```
<textarea name="queryText">Red Balloon</textarea>
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setAttributeValue
request="${request}"
attr="Att_Text_In"
source="queryText"/>
```

This tag sets the value that is specified by the identifier *'queryText'* for the index attribute *'Att_Text_In'*. In this example the value will be resolved to *'Red Balloon'* (see section "Resolving Values in Set-Tags").
The resolved value should be a string, a collection of strings, or a string array.
If the identifier does not exist in any of the legal scopes the value for the given index attribute will not be set to request object.
If no *'source'*-attribute was given, the name of the attribute (in this case *'Att_Text_In'*) would be used as identifier for the actual value.

7.2.3 Filters

All of the attributes in a model can also be used to narrow or limit a search by setting filters. Filters are applied as 'hard' restrictions to the result. This means that results have to match the given filters exactly. The value of the filter has to correspond to a value of the attribute defined in the model.
There are three different kinds of filters. *Interval-filters* and *enumeration-filters* can be used for atomic attributes and *has-elements-filters* can be used for set attributes. For the first two kinds we distinguish for each between inclusion and exclusion filters. Inclusion means that the result of a query must include the corresponding attribute value. Exclusion means that the result of a query must not include the corresponding attribute value.
Taxonomy filters can be set. Thus, when specifying a filter for an attribute, the given value and all values in the taxonomy below are excluded or included.

setIntervalFilter

This tag passes filter data for a given index attribute to the request object (instance of *com.empolis.orenge.api.Request*). An interval filter specifies an interval by giving a minimum and a maximum value for a named index attribute. There are two possible filter types: *inclusion* or *exclusion*. For an inclusion filter the value of the index attribute has to lie between minimum and maximum. For an exclusion filter the value of the index attribute has to lie outside the given interval.

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
attr	required	—	Name of the index-attribute for which the filter should be set.	String
min	optional	—	Name of the request-parameter or context-attribute that represents the minimum value of the interval	String

max	optional	—	The name of the request-parameter or contect-attribute that represents the maximum value of the interval	String
type	optional	inclusion	Defines wether the given interval should be included or excluded in the results	'inclusion' or 'exclusion'

Example
Samplecode in searchform

```
<input type="hidden" name="minDate" value="2004-01-01"/>
<input type="hidden" name="maxDate" value="2005-01-01"/>
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setIntervalFilter request="${request}"
attr="Att_LastModified"
min="minDate"
max="maxDate"
type="inclusion" />
```

This tag sets the values that are specified by the identifiers *'minDate'* and *'maxDate'* as interval-filter for the index-attribute *'Att_LastModified'*. So the interval has a range from '2004-01-01' to '2005-01-01' (see section "Resolving Values in Set-Tags").
The resolved values should be strings.
If none of the identifiers do exist in any of the legal scopes the interval filter will not be passed to request object.
As type is set to *'inclusion'* we will only get results whose attribute *'Att_LastModified'* has a value that is between 2004-01-01 and 2005-01-01.

setEnumerationFilter
This tag passes an enumeration filter for a specified index-attribute to the request-object (instance of *com.empolis.orenge.api.Request*). Enumeration filters are used to define one or more (in-)valid values for an atomic attribute.
For the type *'inclusion'* the given values are taken as valid and for type *'exclusion'* the given values are taken as invalid.
If a taxonomy name is given, all values in the taxonomy below the given value(s) are used as filter-values.

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
attr	required	—	Name of the index-attribute for which the filter should be set.	String
source	required	—	Name of the request-parameter or context-attribute that represents the actual value of the filter	String

type	optional	inclusion	Defines wether the given value should be included or excluded in the results	'inclusion' or 'exclusion'
taxonomy	optional	—	Name of the taxonomy that contains filter-values	String'

Example
Samplecode in searchform

```
<select name="division" multiple="multiple">
<option value=""></option>
<option value="Development">Development</option>
<option value="Marketing">Marketing</option>
</select>
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setEnumerationFilter
request="${request}"
attr="Att_Division"
source="division"
type="exclusion"/>
```

In this example the selected value(s) from the select-field *'division'* are set as an enumeration filter to the attribute *'Att_Division'* (see section "Resolving Values in Set-Tags").
If type is set to *'exclusion'* only results will be returned that value of attribute *'Att_Division'* is different than the selected values.
If identifier *'devision'* cannot be resolved or no values are selected the filter will not be passed to the request object. The resolved value should be a string, a collection of strings, or a string array.

setHasElementsFilter
This tag passes a has-elements-filter for a specified index-attribute to the request object (instance of *com.empolis.orenge.api.Request*). Has-elements-filters are used to define one or more valid values for a set attribute. Filter type can be set to *'all'*, *'any'*, *'only,'* or *'none'* with the following meanings:
all — result needs to have all filter values
any — result needs to have at least one of the filter values
only — the filter values are the only allowed values for the result
none — only other values than the filter values are allowed for the result
If a taxonomyname is given, all values in the taxonomy below the given value(s) are used as filter values.

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
attr	required	—	Name of the index-attribute for which the filter should be set.	String

source	required	—	Name of the request-parameter or context-attribute that represents the actual value of the filter	String
type	optional	all		'all', 'any', 'only', 'none'
taxonomy	optional	—	Name of the taxonomy that contains filter-values	String'

Example
Samplecode in searchform

```
<input type="checkbox" name="authors" value="Kent Beck"/>
<input type="checkbox" name="authors" value="Erich Gamma"/>
<input type="checkbox" name="authors" value="Marty Hall"/>
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setHasElementsFilter
request="${request}"
attr="Att_Author"
source="authors"
type="any"/>
```

In this example the value(s) from the checked checkboxes with *name='authors'* are set as an has-elements-filter to the attribute *'Att_Authors'* (see section "Resolving Values in Set-Tags").
If type is set to *'any'* only results will be returned that values of attribute *'Att_Author'* include at least one of the filter values.
If identifier *'authors'* cannot be resolved or no values are selected the filter will not be passed to the request object.
The resolved value should be a string, a collection of strings, or a string array.

7.2.4 Dynamic Weighting

The dynamic weighting of every single attribute can be specified with different parameters. This way, you can indicate which attributes are more important and which are less. Models also can be adapted for specific tasks in this manner — for example, you might weight individual attributes differently for different customer groups. The given value must be greater than or equal to 0. This digit then is multiplied by the weight of the attribute specified in the valuation model (OVML). Consequently, a '1.0' does not alter the defined weighting while a '2.0' doubles the weighting and '0.5' halves it. When the value is set to '0', the corresponding attribute is not included in the global similarity calculation but still be used for filters.

setImportance
This tag passes the given weighting to the respective attribute in the request object (instance of *com.empolis.orenge.api.Request*).

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object

attr	required	—	Name of the index-attribute for which the weighting should be set.	String
source	required	—	Name of the request-parameter or context-attribute that represents the actual value for the weighting	String

Example
Samplecode in searchform

```
<input type="hidden" name="Category_Importance" value="2.0" />
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setImportance
request="${request}"
attr="Category"
source="Category_Importance"/>
```

This tag sets the value that is specified by the identifier *'Category_Importance '* for the index-attribute *'Category'*. In this example the value will be resolved to *'2.0'* (see section “Resolving in Set-Tags”).
The resolved value should be a string that can be parsed into a double-value.
If the identifier does not exist in any of the legal scopes the property will not be set to request object.

7.2.5 Similarity Measures

Like with weighting, it is also possible to specify which similarity value is to be used for the respective attribute. Please consider that the given measure must have been specified in the valuation model.

setUseMeasure
This tag passes the given measure to the respective attribute in the request-object (instance of *com.empolis.orenge.api.Request*).

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
attr	required	—	Name of the index-attribute for which the measure should be set.	String
source	required	—	Name of the request-parameter or context-attribute that represents the actual value of the measure	String

Example
Samplecode in searchform

```
<input type="hidden" name="Milk_Measure" value="milk" />
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setUseMeasure
request="${request}"
attr="Milk"
source="Milk_Measure"/>
```

This tag sets the value that is specified by the identifier *'Milk_Measure'* for the index attribute *'Milk'*. In this example the value will be resolved to *'milk'* (see section "Resolving Values in Set-Tags").
The resolved value should be a string.
If the identifier does not exist in any of the legal scopes the property will not be set to request object.

7.2.6 Wildcards

With the help of SIP:TextMiner so-called wildcards can be recognized in a free text search query.

setExpandWildcards
This tag activates or deactivates the expansion of wildcards for the given attribute in the request object (instance of *com.empolis.orenge.api.Request*).

Attribute	Use	Default	Description	Value(s)
request	required	—	The request-object that represents the query.	Request-Object
attr	required	—	Name of the index-attribute for which the measure should be set.	String
source	required	—	Name of the request-parameter or context-attribute that represents the actual value of the measure	String

Example
Samplecode in searchform

```
<input type="hidden" name="expandWildcards" value="true" />
```

Samplecode in processing page

```
<ekm:createRequest requestManager ="${rm}" var="request" />
[…]
<ekm:setExpandWildcards
request="${request}"
attr="Att_Text_In"
source="expandWildcards"/>
```

This tag activates the expansion of wildcards with the index-attribute *'Att_Text_In'*, because the value of the specified identifier *'expandWildcards'* will be resolved to *'true'* (see section "Resolving Values in Set-Tags").
The resolved value should always be 'true' or 'false'.
If the identifier does not exist in any of the legal scopes the property will not be set to request-object.

7.3 Processing Results

7.3.1 Get-tags

getQueryObject
Having executed a search request (see executeRequest) we can get the query object of a given pipelet from the response object (instance of *com.empolis.orenge.api.Response*) by using the *'getQueryObject'* tag. The received result is available from the page-scoped variable with the name given in tag attribute 'var'. The result is returned as an instance of *com.empolis.orenge.api.AggregateAdapter*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received aggregate	String
response	required	—	The response-object that contains the result.	Response-Object
pipeletID	required	—	Name of the pipelet of which we want to get the queryObject	String

Example
Samplecode in processing page

```
<executeRequest var="response" requestManager="${rm}"
request="${request}" />
[…]
<ekm:getQueryObject
var="queryObject"
response="${response}"
pipeletID="multiRetrieverPipelet"/>
```

In this example the query object of the pipelet *'multiRetrieverPipelet'* is extracted from response object *'response'* and written to the page-scoped attribute *'queryObject'*.

getInput
Having executed a search request (see executeRequest) we can get the input of a given pipelet from the response object (instance of *com.empolis.orenge.api.Response*) by using the *'getInput'* tag. The received input is available from the page-scoped variable with the name given in tag attribute 'var'. The result is returned as a list of *AggregateAdapters* (*com.empolis.orenge.api.AggregateAdapter*) so that we can easily iterate over the single list items.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received aggregate list	String
response	required	—	The response-object that contains the result.	Response-Object
pipeletID	required	—	Name of the pipelet of which we want to get the input	String

Example
Samplecode in processing page

```
<executeRequest var="response" requestManager="${rm}"
request="${request}" />
<ekm:getInput
var="retrievalInput"
response="${response}"
pipeletID="multiRetrieverPipelet"/>
```

In this example the pipelet input of the pipelet *'multiRetrieverPipelet'* is extracted from response object *'response'* and written to the page-scoped attribute *'retrievalInput'*.

getResult

Having executed a search request (see executeRequest) we can get a result of a given pipelet from the response object (instance of *com.empolis.orenge.api.Response*) by using the *'getResult'* tag. The received result is available from the page-scoped variable with the name given in the tag attribute *'var'*. The result is returned as a list of *AggregateAdapters* (*com.empolis.orenge.api.AggregateAdapter*) so that we can easily iterate over the single list items.

Note

This tag should only be used for service results that return a list of aggregates.
There are other services (e. g. *TermFinder*, *Dialog* …) that return special, individual result structures. Those are handled in special tags (e. g for TermFinder service see getSimilarTerms)

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
response	required	—	The response-object that contains the result.	Response-Object
pipeletID	required	—	Name of the pipelet of which we want to get the result	String

Example

Samplecode in processing page

```
<executeRequest var="response" requestManager="${rm}"
request="${request}" />
<ekm:getResult
var="adaptationResult"
response="${response}"
pipeletID="adaptationPipelet"/>
```

In this example the result of the pipelet *'adaptationPipelet'* is extracted from the response object *'response'* and written to the page-scoped attribute *'adaptationResult'*.

getDialogResult

The dialog result can be received from the response object (instance of *com.empolis.orenge.api.Response*) as result of dialog service. The result is returned as a list of *QuestionAdapters* (list of *com.empolis.orenge.api.QuestionAdapter*) containing their associated answers and is available from the page-scoped variable with the name given in the tag attribute *'var'*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
response	required	—	The response-object that contains the result.	Response-Object
pipeletID	required	—	Name of the pipelet of which we want to get the result	String

Example

```
<executeRequest var="response" requestManager="${rm}"
request="${request}" />
[…]
<ekm:getDialogResult
var="dialogResult"
response="${response}"
pipeletID="dialogPipelet"/>
<c:forEach var="question" items="${dialogResult}" end="2">
<p>${question}</p>
</c:forEach>
```

In this example the result of pipelet 'dialogPipelet' is extracted from the response object 'response' and written to the page-scoped attribute 'dialogResult', whereas the result is a list of questions. After that the first three questions are printed to the page by iteration over the result list. The following HTML-output may be generated:

```
<p>Which category matches your dish?</p>
<p>What kind of dish do you prefer?</p>
<p>What equipment would you like to use?</p>
```

getDialogAnswersForQuestion

This tag extracts the associated answers to a given question (instance of *com.empolis.orenge.api.QuestionAdapter*) and writes them as a list of dialog answers to the page-scoped variable with the name given in the tag-attribute *'var'*.
Depending on the question type the returned result can be an *AtomicEnumeration* (for enumeration and taxonomy questions) or an *AtomicInterval* (for interval questions)

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
question	required	—	Dialog question	QuestionAdapter

Example

```
<ekm:getDialogResult
var="dialogResult"
response="${response}"
pipeletID="dialogPipelet"/>
<c:forEach var="question" items="${dialogResult}" end="2">
<ekm:getDialogAnswersForQuestion
var="answers"
question="${question}"/>
      </c:forEach>
```

In this example the answers for the given question are written to the page-scoped attribute 'answers'.
To see how to parse and output these answers have a look at *checkDialogAnswerType*, *printDialogEnumerationAnswer,* and *printDialogIntervalAnswer* tags.

getAnalysisMisspelledWords
From this tag we receive a list of misspelled words (instances of *com.empolis.orenge.api.MisspelledWordAdapter*) that is written to the page-scoped variable with the name given in the tag attribute *'var'*. The spellchecker results can be extracted out of the result of textminer service.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
result	required	—	result-object from textminer service	Textminerresult (list of AggregateAdapter)
attr	optional	—	Name of the attribute to receive misspelled words from. If not set the misspelled words from all attributes are returned.	String

Example

```
<ekm:getResult var="analysisResult" response="${response}"
pipeletID="textminerPipelet"/>
<ekm:getAnalysisMisspelledWords
var="misspelledWords"
result="${analysisResult}" />
```

In this example all misspelled words from result object *'analysisResult'* are extracted and written to the page-scoped attribute *'misspelledWords'*. In a next step we could iterate over the received list of misspelled words. For an example see *printAnalysisMisspelledWord*, *printAnalysisSuggestion*.

getAnalysisSuggestions
For each misspelled word from the textminer result (see *getAnalysisMisspelledWords*) we can get a list of suggestions for the correct spelling of the respective word. This list is written to the page-scoped variable with the name given in tag-attribute *'var'*. Each list item is an instance of *com.empolis.orenge.api.CandidateAdapter.*

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
element	required	—	Misspelled word	MisspelledWord Object

Example

```
<ekm:getResult var="analysisResult" response="${response}"
pipeletID="textminerPipelet"/>
<ekm:getAnalysisMisspelledWords
var="misspelledWords"
result="${analysisResult}" />
<c:forEach var="word" items="${misspelledWords}">
<ekm:getAnalysisSuggestions var="suggestions" element="${word}"/>
</c:forEach>
```

In this example all misspelled words from the result object *'analysisResult'* are extracted and written to the page-scoped attribute *'misspelledWords'*. Then we use a forEach-loop to iterate over the received list of misspelled words.In any iteration we extract the list of suggestions for the respective word and write it to the page-scoped variable *'suggestions'*. For another example see *printAnalysisMisspelledWord*, *printAnalysisSuggestion*.

getAttributeValue

With the tag *'getAttributeValue'* we can request a value of a single attribute in an aggregate and store it in a page-scoped variable with the name given in tag-attribute *'var'*. After that we can access the variable at multiple locations in the JSP.

This tag is often used for attributes containing a set of values, so that we can print out the single items by iterating over the recently created variable.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received attribute value	String
aggregate	required	—	Actual aggregate (result item)	Aggregate Adapter
attr	required	—	Name of attribute	String
lang	optional	Default language from confiuration	Desired language as a two-letter string	String

Example

Samplecode in result page

```
<ekm:getResult var="adaptationResult" response="${response}"
pipeletID="adaptationPipelet"/>
<c:forEach var="case" items="${adaptationResult}">
<ekm:getAttributeValue
var="authors"
aggregate="${case}"
attr="Att_Author" />
<c:forEach var="author" items="${authors}">
${author}, 
</c:forEach>
</c:forEach>
```

In the example we first receive the result of *adaptationPipelet* which is represented by a list of *AggregateAdapters* stored in variable *'adaptationResult'*. In the following forEach-loop we iterate over this list, whereas each list item is stored in variable *'case'*. Assuming *'Att_Authors'* is a set attribute we use the *getAttributeValue* tag to write the value of this attribute into variable *'authors'*. Now we iterate over this variable and print out each item of the set by using a second JSTL forEach-tag.

getPropertyValue

With the tag *'getPropertyValue'* we can request a value of a property from a single attribute in an aggregate and store it in a page-scoped variable with the name given in tag attribute *'var'*. Depending on the property the result can be a single string or a list of strings.

This tag is often used, for properties containing a set of values, so that we can print out the single items by iterating over the recently created variable.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received attribute value	String
aggregate	required	—	Actual aggregate (result item)	AggregateAdapter
attr	required	—	Name of attribute	String
property	required	—	Specifies the property to be printed. Property values may be the importance measure or filters of a query values	'importance', 'measure', 'enumerationInc', 'enumerationExc', 'enumerationIncTax', 'enumerationExcTax', 'hasElementsAll', 'hasElementAny', 'hasElementsOnly', 'hasElementsNone', 'haselementsAllTax', 'hasElementsAnyTax', 'hasElementsOnlyTax', 'hasElementsNoneTax', 'intervalMin', 'intervalMax', 'intervalType'

Example

```
<ekm:getQueryObject var="queryObject" response="${response}"
pipeletID="multiRetrieverPipelet"/>
<c:if test="${queryObject != null}">
<ekm:getPropertyValue
var="hasElementsProp"
aggregate="${queryObject}"
property="hasElementsAny"
attr="Category" />
<c:forEach var="item" items="${hasElementsProp}">
${item}, 
</c:forEach>
</c:if>
```

In this example we request the values of the hasElements filter (type 'any') from the aggregate *'queryObject'* (instance of *com.empolis.orenge.api.AggregateAdapter*) and attribute *'Category'*. If a hasElements filter always contains a list of values, we receive a list which is written to the page-scoped variable *'hasElementsProp'*.
After that we iterate over this variable and print out each item of the list by using the JSTL forEach-tag.

getSimilarTerms (only SIP.5.x)

The *'getSimilarTerms'* tag requests the result of a TermFinderService and writes the received list of terms to the page-scoped variable with the name given in the tag

attribute *'var'*. With the tag attribute pipeletID we specify the service from which we request the result. This service necessarily has to return a TermServiceResult.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
response	required	—	Response-object that contains the result	Response-Object
pipeletID	required	—	Name of the pipelet of which we want to get the result	String

Example
Samplecode in result page

```
<executeRequest var="response" requestManager="${rm}"
request="${request}" />
[…]
<ekm:getSimilarTerms
var="simTerms"
response="${response}"
pipeletID="TermFinderPipelet" />
<c:forEach var="term" items="${simTerms}">
<a href="javascript:addTerm('${term}')">${term}</a>
</c:forEach>
```

In this example we request the result from *TermFinderPipelet* and write it to the variable *'simTerms'*. The result is a list of terms. In the next step we use the JSTL forEach-tag to iterate over this list and print the single terms twice into a HTML a-tag. In this example the mentioned JavaScript-function is used to add the term to the searchform and submit the form.

getTotalNumberOfHits
This tag writes the total number of hits to the page-scoped variable with the name given in tag-attribute *'var'*. The total number of hits can only be received from the result of retrieval service. So the given result has to be an instance of *com.empolis.orenge.api.RetrievedCaseList* which is automatically returned as retrieval service results.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
result	required	—	Retrieval result	RetrievedCase List

Example
Samplecode in result page

```
<ekm:getResult var="retrievalResult" resultBean="${rb}"
pipeletID="multiRetrieverPipelet"/>
<p>
<ekm:getTotalNumberOfHits var="countHits'
result="${retrievalResult}" />
Number of Results: ${countHits}
</p>
```

In this example the total number of results is extracted from the result of *'multiRetrieverPipelet'* and written to the page-scoped variable *'countHits'*. The following HTML output may be generated:

```
<p>
Number of Results: 5308
</p>
```

getOMMLModel

This tag reads the actual OMML-model and writes it to the page-scoped variable with the name given in the tag attribute *'var'*. The returned model is an instance of *com.empolis.orenge.api.ModelAdapter*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
requestManager	required	—	RequestManagerBean that handles the requests	RequestManager Bean

Example

```
<jsp:useBean
id="rm"
scope="request"
class="com.empolis.orenge.api.RequestManager">
[…]
</jsp:useBean>
<ekm:getOMMLModel var="model" requestManager="${rm}"/>
```

getOMMLModelProperties

This tag reads the value of the requested property from the model and writes it as a list of strings to the page-scoped variable with the name given in the tag attribute *'var'*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
model	required	—	OMML model as ModelAdapter	ModelAdapter
property	required	—	Specifies the model property that is returned	‘Name’, ‘Explanation’, ‘comment’, ‘annotation’, ‘Creation’, ‘Locale’, ‘Memo’
name	optional	—	This attribute has nly to be specified if the property is a Memo. In this case Memo’s name is given. If no name is given all Memos are returned.	String (Memo’s name)

timestamp	optional	—	This parameter has only to be specified if the property is a Creation. In this case the Creation's timestamp may be given. If no timestamp is given all Creations are returned.	String (Vreation's timestamp
lang	optional		Desired language as a two-letter string. If no language attribute is given, the property's value is returned in all possible languages.	String

Example

```
<ekm:getOMMLModel var="model" requestManager="${rm}" />
[…]
<ekm:getOMMLModelProperties
var="properties"
model="${model}"
property="Memo"
name="MyModel" />
<c:forEach var="memo" items="${properties}">
MEMO: ${memo} <br/>
</c:forEach>
```

Example

```
<ekm:getOMMLModel var="model" requestManager="${rm}" />
[…]
<ekm:getOMMLModelProperties
var="properties"
model="${model}"
property="Creation"
timestamp="2005-01-03 12:00:00.0" />
<c:forEach var="creation" items="${properties}">
Created on 2005-01-03 12:00:00.0: ${creation} <br/>
</c:forEach>
```

getOMMLClassProperties

This tag reads the value of the requested property of the given class from the model and writes it as a list of strings to the page-scoped variable with the name given in the tag attribute *'var'*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
model	required	—	OMML model as ModelAdapter	ModelAdapter
className	required	—	Specifies the class to which the property is attached	String
property	required	—	Specifies the model property that is returned.	'Name', 'explanation', 'comment', 'Annotation', 'Creation', Question', 'Memo'

name	optional	—	This attribute has only to be specified if the property is a Memo. In thiscase Memo's name may be given. If no name is given all Memoms are returned.	String (Memo's name)
timestamp	optional	—	This parameter has only to be specified if the property is a Creation. In this case Creation's timestamp may be given. If no timestamp is given all Creations are returned.	String (Creation's timestamp)
lang	optional	—	Desired language as a two-letter string. If no language attribute is given, the property's value is returned in all possible languages.	String

Example

```
<ekm:getOMMLModel var="model" requestManager="${rm}" />
[…]
<ekm:getOMMLClassProperties
var="question"
model="${model}"
property="Question"
className="Recipe.V1"
lang="de"/>
<c:forEach var="q" items="${question}">
Question: ${q} <br/>
</c:forEach>
```

getOMMLValueProperties

This tag reads the value of the requested property of the given class and the value from the model and writes it as a list of strings to the page-scoped variable with the name given in the tag attribute *'var'*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
model	required	—	OMML model as ModelAdapter	ModelAdapter
className	required	—	Specifies the class to which the property is attached	String
value	required	—	Specifies the value to which the property is attached	String
property	required	—	Specifies the model propertythat is returned	'Name', 'Explanation', 'Comment', Annotation'
lang	optional	—	Desired Language as a two-letter string. If no language attribute is given property's value is returend in all possible languages.	String

Example

```
<ekm:getOMMLModel var="model" requestManager="${rm}" />
[…]
<ekm:getOMMLValueProperties
var="name"
model="${model}"
property="Name"
className="Liquids.V1"
value="Essig"
lang="en" />
<c:forEach var="n" items="${name}">
Value in English: ${n} <br/>
</c:forEach>
```

getOMMLAttributeProperties

This tag reads the value of the requested property of the given attribute from the model and writes it as a list of strings to the page-scoped variable with the name given in the tag attribute *'var'*.

Attribute	Use	Default	Description	Value(s)
var	required	—	Name for the indentifier of the received result	String
model	required	—	OMML model as ModelAdapter	ModelAdapter
agg	required	—	Specifies the name of the aggregate containing the attribute	String (aggregate name)
attr	required	—	Specifies the attribute to which the property is attached.	String (attribute name)
property	required	—	Specifies the model propertythat is returned	'Name', 'Explanation', 'Comment', Annotation', 'Question', 'Memo'
name	optional	—	This attribute has only to be specified if the property is a Memo. In this case Memo's name may be given. If no name is given all Memos are returned.	String (Memo's name)
lang	optional	—	Desired language as a two-letter string. If no language attribute is given, the property's value is returned in all possible languages.	String

Example

```
<ekm:getOMMLModel var="model" requestManager="${rm}" />
[…]
<ekm:getOMMLAttributeProperties
var="question"
model="${model}"
property="Question"
agg="Recipe.V1"
attr="Category"
lang="de"/>
<c:forEach var="q" items="${questionAttr}">
      Attribute Question: ${q} <br/>
</c:forEach>
```

7.3.2 Print-Tags

printAttributeValue
Related to the *'getAttributeValue'* tag the *'printAttributeValue'* tag returns the value of a single attribute from a given aggregate (instance of *com.empolis.orenge.api.AggregateAdapter*). But here the value is immediately written to the HTML-output instead of writing it into a variable. As an additional attribute we can set the output format to either *'text'* or *'url'*. Format *'text'* is default and returns the value as it is. Format *'url'* returns the value url-encoded.
If we set the optional attribute *lang* we have to assure that the requested language is defined in the model.

Attribute	Use	Default	Description	Value(s)
aggregate	required	—	Actual aggregate	AggregateAdapter
attr	required	—	Name of attribute	String
format	optional	text	Output format (original text or url-encoded text)	String (‘text’ or ‘url’)
lang	optional	Default language from configuration	Desired language as a two-letter string	String

Example
Samplecode in result page

```
<ekm:printAttributeValue aggregate="${case}" attr="Att_Title"
format="text"/>
```

This example requests the value of the attribute *'Att_Title'* from the given aggregate that is represented by the variable *'case'* and prints it to the HTML-output.

Abbreviated Form
As an alternative for using the *printAttributeValue* tag above we can use the following abbreviated syntax to get the same output:

```
${case["Att_Title"].value}
```

As we can see an *AggregateAdapter* is a map with keys corresponding to the attributes of the aggregate. The value of the map-item returns the actual value of the attribute.

We also can print out attribute's label:

```
${case["Att_Title"].label}
```

Syntax for the value of the attribute

```
aggregate["attributename"].value
```

Syntax for the label of the attribute

```
aggregate["attributename"].label
```

As a restriction towards the *printAttributeValue* tag we cannot set parameters *format* and *lang* in the abbreviated form. For those only the default-values can be assumed.

printAttributeMarkup

This tag prints the mark-up of a given attribute in an aggregate (instance of *com.empolis.orenge.api.AggregateAdapter*) to the screen. We can choose between text or table mark-up. Text mark-up marks recognized attributes in text; table mark-up visualizes attributes, e. g. by specifying the similarity value.
If no mark-up is available for the given attribute an empty string is printed to the page.

Attribute	Use	Default	Description	Value(s)
aggregate	required	—	Actual aggregate	AggregateAdapter
attr	required	—	Name of attribute	String
type	optional	'text'	Choose TextMarkup or TableMarkup	String ('text' or 'table')

Example

Samplecode in result page

```
<ekm:getResult var="markerResult" response="${response}"
pipeletID="simMarkerPipelet"/>
<c:if test="${fn:length(markerResult) > 0}">
<ekm:printAttributeMarkup aggregate="${markerResult[0]}"
attr="Filename"/>
</c:if>
```

printConcept

This tag prints the concept name to a given *AtomicAdapter* (*com.empolis.orenge.api.AtomicAdapter*) to the screen. As an example an *AtomicAdapter* could be the value of an atomic attribute or a single item from a set attribute's value.

Attribute	Use	Default	Description	Value(s)
value	required	—	Actual value	AtomicAdapter

Example

```
<ekm:getAttributeValue var="Vegetables" aggregate="${queryObject}"
attr="Vegetables"/>
<c:forEach var="item" items="${Vegetables}">
<input type="hidden" name="Vegetables" value="<ekm:printConcept
value="${item}" />">
</c:forEach>
```

printPropertyValue
Related to the *'getPropertyValue'* tag the *'printPropertyValue'* tag returns the value of a property from a single attribute in an aggregate (instance of *com.empolis.orenge.api.AggregateAdapter*). But here the value is immediately written to the HTML output instead of writing it into a variable. As an additional attribute we can set the output format to either *'text'* or *'url'*. Format *'text'* is default and returns the value as it is. Format *'url'* returns the value url-encoded.

Attribute	Use	Default	Description	Value(s)
aggregate	required	—	Actual aggregate (result item)	AggregateAdapter
attr	required	—	Name of the attribute	String
property	required	—	Specifies the property to be printed. Property values may be the importance, measure, or filters of a query value.	'importance', 'measure', enumerationInc',enumerationExc', enumerationTax', 'hasElementsAll', 'hasElementAny', 'hasElementsOnly', 'hasElementsNone', 'haselementsAllTax', 'hasElementsAnyTax', 'hasElementsOnlyTax', 'hasElementsNoneTax', 'intervalMin', 'intervalMax', 'intervalType'
format	optional	text	Output format (original text or url-encoded text)	String ('text' or 'url')

Example

```
<ekm:getQueryObject var="queryObject" response="${response}"
pipeletID="multiRetrieverPipelet"/>
<c:if test="${queryObject != null}">
<ekm:printPropertyValue
aggregate="${queryObject}"
attr="Category"
property="hasElementsAny" />
</c:if>
```

In this example we request the values of the hasElements filter (type 'any') from the aggregate *'queryObject'* and the attribute *'Category'*. If a hasElements filter always contains a list of values, the list items are printed to the HTML output as a comma separated list.

printAnalysisMisspelledWord
Depending on the given value this tag either prints the actual word from the spellchecker result or the name of the respective source attribute to the screen.

Attribute	Use	Default	Description	Value(s)
word	required	—	Specifies the object that represents the misspelled word	MisspelledWordAdapter
value	optional	'label'	Choose to print either the actual word or the name of the respective source attribute	'label' or 'sourceAttribute'

Example

```
<ekm:getResult var="analysisResult" response="${response}"
pipeletID="textminerPipelet"/>
<ekm:getAnalysisMisspelledWords var="misspelledWords"
result="${analysisResult}" />
<c:if test="${fn:length(misspelledWords) > 0}">
<h3>Unknown words:</h3>
<c:forEach var="word" items="${misspelledWords}">
<ekm:printAnalysisMisspelledWord word="${word}" />
(found in attribute: <ekm:printAnalysisMisspelledWord word="${word}"
value="sourceAttribute"/>)<br/>
</c:forEach>
</c:if>
</c:if>
```

In this example all misspelled words from result-object *'analysisResult'* are extracted and written to the page-scoped attribute *'misspelledWords'*. After that we iterate over the received list of misspelled words and print them to the screen. The following HTML output may be generated:

```
<h3>Unknown words:</h3>
cheese
(found in attribute: PlainQuery)<br/>
potateo
(found in attribute: PlainQuery)<br/>
```

Abbreviated Form

As an alternative for using the *printAnalysisMisspelledWord* tag above we can use the following abbreviated syntax to get the same output:
For the print out of the actual word

```
${word.label}
```

For the print out of the name of the respective source attribute

```
${word.sourceAttribute}
```

Example

```
[…]
<c:forEach var="word" items="${misspelledWords}">
${word.label}
(found in attribute: ${word.sourceAttribute})<br/>
</c:forEach>
[…]
```

printAnalysisSuggestion

If we have received suggestions (instance of *com.empolis.orenge.api.CandidateAdapter*) for a misspelled word from the

spellchecker result we can print the single suggestions to an HTML output. Depending on the given value we can display different information on the current suggestion.

Attribute	Use	Default	Description	Value(s)
suggestion	required	—	Specifies one suggestion as alternative to the misspelled word	CandidateAdapter
value	optional	'key'	Choose what inforamtion to print about the suggestion	'key' → suggestion', 'attr' → name of destination attribute, 'concept' → concept name, 'sim' → similarity between misspelled and suggested word
lang	optional	Default language	Desired language as a two-letter string	String

Example

```
<ekm:getResult var="analysisResult" response="${response}"
pipeletID="textminerPipelet"/>
<ekm:getAnalysisMisspelledWords var="misspelledWords"
result="${analysisResult}" />
<c:if test="${fn:length(misspelledWords) > 0}">
<h3>Unknown words:</h3>
<c:forEach var="word" items="${misspelledWords}">
<p><b><ekm:printAnalysisMisspelledWord word="${word}" /></b>:
<ekm:getAnalysisSuggestions var="suggestions" element="${word}"/>
<c:choose>
<c:when test="${fn:length(suggestions) > 0}">
<select name="${word.sourceAttribute}"/>">
<option value="">please choose:</option>
<c:forEach var="sugg" items="${suggestions}">
<option value="<ekm:printAnalysisSuggestion suggestion="${sugg}"
value="key"/>">
<ekm:printAnalysisSuggestion suggestion="${sugg}" value="key" />
<ekm:printAnalysisSuggestion suggestion="${sugg}" value="sim" />
</option>
</c:forEach>
</select>
</c:when>
<c:otherwise>
Please check the spelling of this word.
</c:otherwise>
</c:choose>
</p>
</c:forEach>
</c:if>
</c:if>
```

In this example all misspelled words from result-object *'analysisResult'* are extracted and written to the page-scoped attribute *'misspelledWords'*. After that we iterate over

the received list of misspelled words and print them to the screen together with the respective suggestions. The following HTML output may be generated:

```
<h3>Unknown words:</h3>
<p><b>chese</b>:
<select name="PlainQuery">
<option value="">please choose:</option>
<option value="cheese">cheese - 92%</option>
<option value="chee">chee - 88%</option>
<option value="cheeses">cheeses - 86%</option>
</select>
</p>
<p><b>potateo</b>:
<select name="PlainQuery">
<option value="">please choose:</option>
<option value="potato">potato - 92%</option>
<option value="potatoe">potatoe - 86%</option>
</select>
</p>
<p><b>better</b>:
Please check the spelling of this word.
</p>
```

printDialogQuestionAttribute
This tag prints the attribute of a question to an HTML output. For an example see *printDialogEnumerationAnswer* and *printDialogIntervalAnswer*.

Attribute	Use	Default	Description	Value(s)
question	required	—	Actual question	QuestionAdapter

checkDialogAnswerType
This tag is used to handle the questions returned by SIP:Dialog in different ways, depending on the corresponding answer types. A question needs to be given as an instance of *com.empolis.orenge.api.QuestionAdapter* as returned by *getDialogResult*-tag. For an example see *printDialogIntervalAnswer*.

Attribute	Use	Default	Description	Value(s)
question	required	—	Actual question	QuestioneAdapter
type	required	—	Defines the type of answer values for the spective question.	‘interval’, ‘enumeration’, ‘taxonomy’

printDialogEnumerationAnswer
This prints answers of the type enumeration. As *getDialogAnswersForQuestion*-tag returns a list of *AnswerValueAdapters* for enumeration answers, the actual answer has to be specified as instance of *com.empolis.orenge.api.AnswerValueAdapter*. The returned value differs depending on the value given in content attribute. 'value' returns the name of the concept and 'name' returns the language specific identifier. For an example see *printDialogIntervalAnswer*.

Attribute	Use	Default	Description	Value(s)
answer	required	—	Specifies the object that represents the answer	AnswerValueAdapter

content	optional	'value'	Defines what kind of information will be print out.	'value' — print concept name, 'name' — print language specific name

printDialogIntervalAnswer
This prints the answers of the type interval. As *getDialogAnswersForQuestion*-tag returns an *AtomicInterval* for interval answers, the actual answer has to be specified as instance of *com.empolis.orenge.api.AtomicInterval*. For interval answers we can print out either the minimum or maximum of the available interval. The actual output depends on the value of tag attribute minOrMax.

Attribute	Use	Default	Description	Value(s)
answer	required	—	Specifies the object that represents the answer	AtomivInterval
minOrMax	required	—	Defines whether min or max has to be printed	'min', 'max'

Example

```
<ekm:getDialogResult var="dialogResult" response="${response}"
pipeletID="dialogPipelet"/>
<c:if test="${fn:length(dialogResult) > 0}">
<c:forEach var="question" items="${dialogResult}" end="2">
<p>${question}</p>
<ekm:getDialogAnswersForQuestion var="answers"
question="${question}"/>
<ekm:checkDialogAnswerType question="${question}" type="interval">
<input name="<ekm:printDialogQuestionAttribute
question="${question}"/>"/>(between
<ekm:printDialogIntervalAnswer answer="${answers}" minOrMax="min" />
and
<ekm:printDialogIntervalAnswer answer="${answers}" minOrMax="max"
/>)
</ekm:checkDialogAnswerType>
<ekm:checkDialogAnswerType question="${question}"
type="enumeration">
<select name="<ekm:printDialogQuestionAttribute
question="${question}"/>" size="3" multiple>
<option value="">equal</option>
<c:forEach var="answer" items="${answers}">
<option value="<ekm:printDialogEnumerationAnswer answer="${answer}"
content="value"/>">
<ekm:printDialogEnumerationAnswer answer="${answer}"
content="name"/>
</option>
</c:forEach>
</select>
</ekm:checkDialogAnswerType>
</c:forEach>
</c:if>
```

In this example the result of the pipelet 'dialogPipelet' is extracted from response-object 'response' and written to the page-scoped attribute 'dialogResult', whereas the result is a list of questions. After that the first three questions are printed to the page by looping over the result list. At the same time the answers to the respective question

are printed to the screen. The answer format differs depending on the answer type. In the example we distinguish between enumeration answers and interval answers. The following HTML output may be generated:

```
<p>Which category matches your dish?</p>
<select name="Category" size="3" multiple>
<option value="">equal</option>
<option value="Kalorienarmes">lowcal</option>
<option value="Hauptgericht">main course</option>
<option value="Gefluegelgericht">poultry dish</option>
</select>
<p>What kind of spices and herbs do you like?</p>
<select name="SpicesAndHerbs" size="3" multiple>
<option value="">equal</option>
<option value="Zimt">cinnamon</option>
<option value="Dill">dill</option>
<option value="Muskat">nutmeg</option>
<option value="Zwiebel">onion</option>
<option value="Zucker">sugar</option>
</select>
<p>What equipment would you like to use?</p>
<select name="Tools" size="3" multiple>
<option value="">equal</option>
<option value="Schuessel">bowl</option>
<option value="Dose">can</option>
<option value="Kasserolle">casserole</option>
<option value="Ofen">oven</option>
<option value="Loeffel">spoon</option>
<option value="Essloeffel">tablespoon</option>
</select>
```

printSearchcontext

The search context of all hits is returned with the result of a *RetrieverService*. We can use the *'getResult'* tag to receive the result (which is a list of *com.empolis.orenge.api.RetrievedCase* objects) and while iterating over the result list we can print out the search context of each result item by calling the *'printSearchcontext'* tag.

A search context is a section of the current result document that represents a perfect match towards the query. The single query terms are highlighted within this search context. Each term has an individual relevance factor (percent value) for characterization of the document which is reflected by the highlighting.

The highlighting is realized by setting a HTML *span*-tag around the actual term.

This *span*-tag contains a *class*-attribute with the value *'highlightX'*, whereas 'X' is a variable value that depends on the relevance and the value of the given *round*-attribute.

Attribute	Use	Default	Description	Value(s)
case	required	—	Current retrieval case (result item)	Retrieved Case
round	optional	25	Round factor for the relevance value that is used to define css-classes for highlighting	String (Number)

Example

Samplecode in result page

```
<ekm:printSearchcontext case="${case}" round="25"/>
```

This example prints the searchcontext of the given case (result-item) to the HTML output.
Assuming that we searched for the term *'clinton'* the following HTMLoutput may be generated:

```
<span class="highlight100">Clinton</span> William J. <span
class="highlight100">Clinton</span> was the 42nd President of the
United States.
```

If the round attribute is set to '25' the following highlighting classes are possible:

- `highlight100`
- `highlight75`
- `highlight50`
- `highlight25`
- `highlight0`

Thus, we have to define five corresponding css-classes for our result-page.

printRetrievalSim
The similarity of each hit towards the query is returned with the result of a *RetrieverService,* which is a list of *com.empolis.orenge.api.RetrievedCase* objects.
We can use the *'getResult'* tag to receive the result and while iterating over the result list we can print out the similarity of each result item by calling the *'printRetrievalSim'* tag.
We can print out the similarity in three different formats:

- Actual value (double value between 0 and 1)
- Percent-value
- Using a given round-factor.

Attribute	Use	Default	Description	Value(s)
case	required	—	Current retrieval case (result item)	Retrieved Case
format	optional	percent	Round factor for the relevance value that is used to define css-classes for highlighting	String (‘percent’, ‘value’, ‘round25’)

Example
Samplecode in result page

```
<img src="sim<ekm:printRetrievalSim case="${case}"
format="round25"/>.jpg">
```

If the format is set to *'round25'* we receive one of the values 0, 25, 50, 75, or 100, which is equivalent to a rounded percentage.
The result might look like this:

```
<img src="sim75.jpg">
```

This result line only consists of pure HTML code and displays the picture *"sim75.jpg"* in the browser.

printTotalNumberOfHits

This tag prints the total number of hits to the HTML output. The total number of hits only can be received from the result of a *RetrieverService*. So the given result has to be an instance of *com.empolis.orenge.api.RetrievedCaseList* which is automatically returned as retrieval service results.

Attribute	Use	Default	Description	Value(s)
result	required	—	retrieval result	RetrievedCaseList

Example
Samplecode in result page

```
<ekm:getResult var="retrievalResult" response="${response}"
pipeletID="multiRetrieverPipelet"/>
<p>
Number of Results:
<ekm:printTotalNumberOfHits result="${retrievalResult}" />
</p>
```

In this example the total number of results is extracted from the result of *'multiRetrieverPipelet'*. The following HTML output may be generated:

```
<p>
Number of Results:
5308
</p>
```

7.4 Taglib Functions

containedInColl
Signature

```
boolean containedInColl(java.util.Collection coll, java.lang.Object
obj)
```

Returns true if obj is contained in coll, otherwise returns false (compares via 'equals').
Example

```
<input type=”checkbox” name=”feedback” value=”$(docID)”
<c:if test=”${ekm.containesInColl(sessionScope.FeedbakList,
docID)}”>
Checked
</c:if>
/>
```

Checks if the current docID is contained in the feedbacklist and if true it prints out 'checked'.

Possible output:

```
<input type=”checkbox” name=”feedback” value=”135” checked/>
```

or

```
<input type=”checkbox” name=”feedback” value=”135”/>
```

containedInArray
Signature

```
boolean containedInArray(java.lang.Object[] array, java.lang.Object
obj)
```

Returns true if obj is contained in array, otherwise it returns false (compares via 'equals').

copyList

Signature

```
java.util.List copyList(java.util.List list)
```

This returns a new list that contains the copies of all items of the original list.

Example

```
<c:set var=”feedbackList”
value=”${ekm:copyList(supplementedFeedbackList)}” scope=”session”/>
```

This creates a copy of supplementedFeedbackList and sets it as session parameter feedbackList.

urlEncode

Signature

```
java.lang.String urlEncode(java.lang.String string)
```

This translates a string into `application/x-www-form-urlencoded` format using a specific encoding scheme.

Example

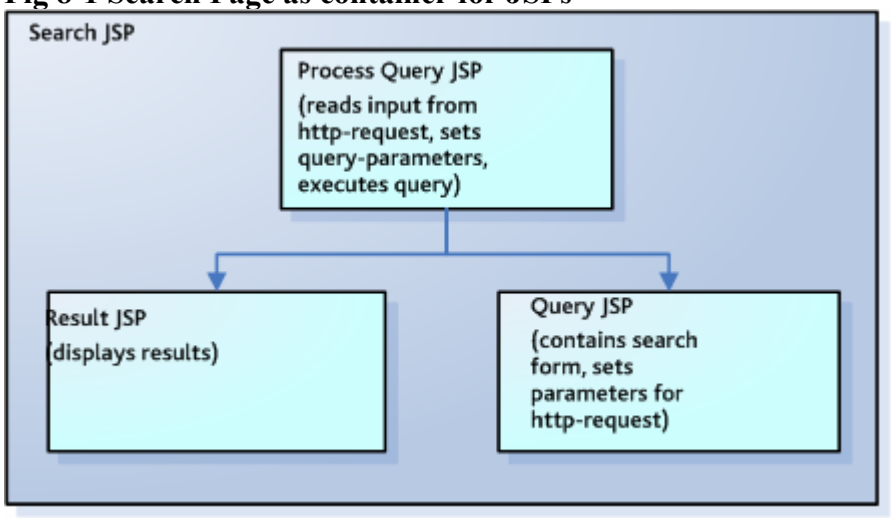
```
<a href="?linkPrefix=${ekm:urlEncode(linkPrefix)}">
```

8 Example

8.1 Search Page

The search page is a container for the three important JSPs (Query JSP, Process Query JSP, and Result JSP) which are included to the page by JSPs include directive. In the search page global parameters for the Web application and the HTML header and HTML footer can be set.

Fig 8-1 Search Page as container for JSPs

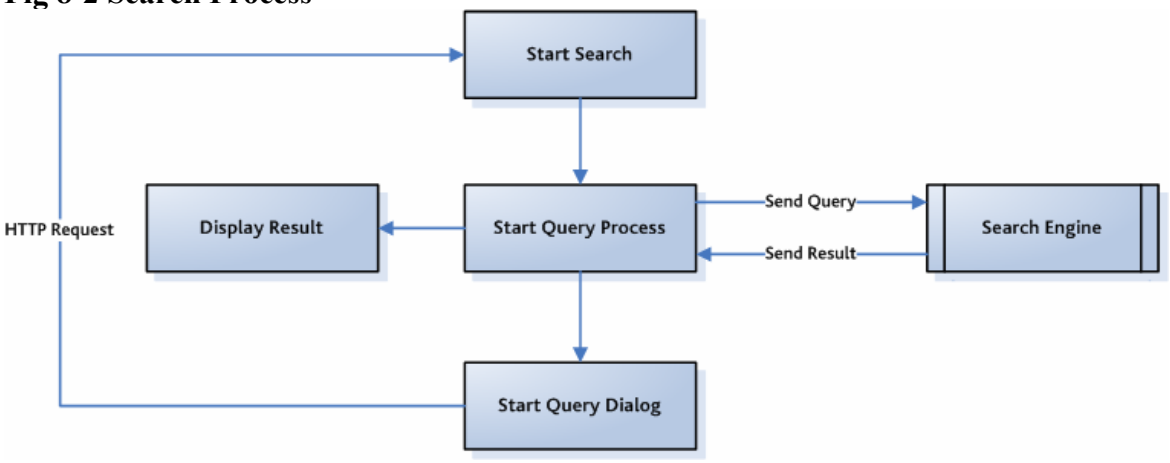


Examples

- The required taglibs are bound to the page.
- ResourceBundles for localization issues are referenced.
- At the beginning of search page we set a page-parameter *'debug'*. If set to *'true'* debug-messages are printed to the page, which is recommended during development process. We can switch off this debug-output easily by setting this parameter to *'false'*.
- With the session-parameter *'lang'* we can set the default-language for the application, which will be used for all kinds of output, if not explicitly set to another language.

The search process begins with the search itself which starts the query process. The query is been send to the search engine, which sends back the results. The next process is the display of the results and a follow-up of another query dialog to refine the search.

Fig 8-2 Search Process



8.2 Query JSP

The Query JSP demonstrates how values of parameters, attributes, and filters for the search request can be set.
All values are set within HTML form-elements. There are some fixed values, like *resultSetSize* or *cursorPosition* that can be set within hidden fields. Other values result from the user input and thus are set within input fields or text areas. List values can be created by using multi-select menus, grouped checkboxes, or just by giving the same name to a group of hidden-fields.

8.3 Process Query JSP

The example Process Query JSP demonstrates how RequestManagerBean can be addressed and parameters, attributes, and filters can be assigned for the query. At the end the actual query is executed.
First we initialize RequestManagerBean and create a *Request* object by using the tag *createRequest*.
In the next steps we collect the request-information that is relevant for the query and assign them:

- Set general parameter values by giving the name of the parameter (*'param'*-attribute) , which we want to set and its value (*'source'*-attribute). As *'source'* we set the name of the request parameter that represents the according value.
- Set pipelet parameter values by giving the name of the pipelet (*'pipeletID'*), the name of the parameter (*'param'*), and the value (*'source'*).
- Set filters. There are different kinds of filters, e. g. the interval filter, for which we have to set a minimum and maximum value. Both are set as names of request parameters that represent the value. Other possible filters are 'HasElementsFilter' or 'EnumerationFilter'.
- Set attribute values by giving the name of the attribute and the according value. A typical attribute value is the actual query text.

Now we are ready to execute the search request by calling the tag *'executeRequest'* which will deliver the result as *Response* object. From this Response object we can extract the results of the different pipelets for example by using the *'getResult'* tag.

8.4 Result JSP

In the result JSP we see how to process and visualize the result.
We can get the results of single pipelets by calling *'getResult'*. According to the actual pipelet we get the result in shape of a list of aggregates, which means that we can iterate over the result content or count the number of items by calculating the length of the result. For iterating and calculating the length we use predefined functionality of JSTL (*c:forEach* and *fn:length*).
When iterating over the result we can visualize single result attributes by calling *'printAttributeValue'*-tag giving the current result item and the desired attribute name to the tag.
Some pipelets deliver special result types that do not contain a list of aggregates. For example, TermFinderPipelet delivers a number of terms that are similar to the terms in the search query.
For those pipelets we do not use the *'getResult'*-tag but there are special tags. For example to receive and visualize the results of *TermFinderPipelet* we use the tag *'getSimilarTerms'*. The result, which is a list of terms, will be written to a variable. The name of this variable can be set by the tag's *'var'*-attribute.
After that we can simply iterate over the list of terms and display them on the page.

9 Appendix — Overview

Create and execute request

```
<ekm:createRequest
            requestManager="
            var=""
            pipeline=""
            queryClass=""
            serviceClass=""
            view="" />
```

```
<ekm:executeRequest
            requestManager=""
            request=""
            var="" />
```

Set Tags

```
<ekm:setGeneralParameterValue
            request=""
            param=""
            source="" />
```

```
<ekm:setPipeletParameterValue
            request=""
            pipeletID=""
            param=""
            source="" />
```

```
<ekm:setAttributeValue
            request=""
            attr=""
            source=""
            target="queryObj","serviceObj" />
```

```
<ekm:setIntervalFilter
            request=""
            attr=""
            min=""
            max=""
            type="inclusion","exclusion" />
```

```
<ekm:setEnumerationFilter
            request=""
            attr=""
            source=""
            type="inclusion","exclusion"
```

```
            taxonomy="" />
<ekm:setHasElementsFilter
            request=""
            attr=""
            source=""
            type="all","any","only","none"
            taxonomy="" />
<ekm:setImportance
            request=""
            attr=""
            source="" />
<ekm:setUseMeasure
            request=""
            attr=""
            source="" />
<ekm:setExpandWildcards
            request=""
            attr=""
            source="" />
```

Get Tags

```
<ekm:getQueryObject
            var=""
            response=""
            pipeletID="" />
<ekm:getInput
            var=""
            response=""
            pipeletID="" />
<ekm:getResult
            var=""
            response=""
            pipeletID="" />
<ekm:getDialogResult
            var=""
            response=""
            pipeletID="" />
<ekm:getDialogAnswersForQuestions
            var=""
            question="" />
<ekm:getAnalysisMisspelledWords
```

```
            var=""
            result=""
            attr="" />
<ekm:getAnalysisSuggestions
            var=""
            element="" />
<ekm:getAttributeValue
            var=""
            aggregate=""
            attr=""
            lang="" />
<ekm:getAttributeValue
            var=""
            aggregate=""
            attr=""
            property="importance","measure","enumerat
ionInc",

            "enumerationExc","enumerationIncTax",

            "enumerationExcTax""hasElementsAll",
                        "hasElementsAny","hasElementsOnly",

            "hasElementsNone","hasElementsAllTax",

            "hasElementsAnyTax","hasElementsOnlyTax",
                        "hasElementsNoneTax",

            "intervalMin","intervalMax","intervalType"
/>
<ekm:getSimilarTerms
            var=""
            response=""
            pipeletID="" />
<ekm:getTotalNumberOfHits
            var=""
            result="" />
<ekm:getOMMLModel
            var=""
            requestManager="" />
<ekm:getOMMLModelProperties
            var=""
```

```
            model=""
            property="Name","Explanation","Comment",

            "Annotation","Creation","Locale","Memo"
            name=""
            timestamp=""
            lang="" />
```

```
<ekm:getOMMLClassProperties
            var=""
            model=""
            className=""
            property="Name","Explanation","Comment",

            "Annotation","Creation","Question","Memo"
            name=""
            timestamp=""
            lang="" />
```

```
<ekm:getOMMLValueProperties
            var=""
            model=""
            className=""
            value=""
            property="Name","Explanation","Comment","
Annotation"
            lang="" />
```

```
<ekm:getOMMLAttributeProperties
            var=""
            model=""
            agg=""
            attr=""
            property="Name","Explanation","Comment",
                      "Annotation","Question","Memo"
            name=""
            lang="" />
```

Print Tags

```
<ekm:printAttributeValue
            aggregate=""
            attr=""
            format="text","url"
            lang="" />
```

```
<ekm:printAttributeMarkup
```

```
            aggregate=""
            attr=""
            type="text","table" />
<ekm:printConcept
            value="" />
<ekm:printPropertyeValue
            aggregate=""
            attr=""
            property="importance","measure","enumerat
ionInc",

            "enumerationExc","enumerationIncTax",

            "enumerationExcTax""hasElementsAll",
                        "hasElementsAny","hasElementsOnly",

            "hasElementsNone","hasElementsAllTax",

            "hasElementsAnyTax","hasElementsOnlyTax",
                        "hasElementsNoneTax","intervalMin",
                        "intervalMax","intervalType"
            format="text","url" />
<ekm:printAnalysisMisspelledWord
            word=""
            value="label","sourceAttribute" />
<ekm:printAnalysisSuggestion
            suggestion=""
            value="key","attr","concept","sim"
            lang="" />
<ekm:printDialogQuestionAttribute
            question="" />
<ekm:checkDialogAnswerType
            question=""
            type="interval","enumeration","taxonomy"
>
</ekm:checkDialogAnswerType >
<ekm:printDialogEnumerationAnswer
            answer=""
            content="value","name","count" />
<ekm:printDialogIntervalAnswer
            answer=""
            minOrMax="min","max" />
```

```
<ekm:printSearchcontext
            case=""
            round="" />
```

```
<ekm:printRetrievalSim
            case=""
            format="percent","value","round25" />
```

```
<ekm:printTotalNumberOfHits
            result="" />
```

Index