DSpace Discovery

This page is outdated. You are probably looking for information of Discovery in your particular version of DSpace:

Discovery in DSpace 3.x

Discovery in DSpace 1.8.x

Discovery in DSpace 1.7.x

- Proposal For Inclusion into DSpace 1.7.0
- Introduction Video
- Documentation
- Design Premise for Discovery
- RoadMap
- Issue Management
- Subversion Access
- Examples in Production
- Discovery in DSpace 1.8.0
  - New Discovery backend (unconfirmed)
  - New discovery configuration (unconfirmed)
    - The general discovery settings (discovery.cfg)
    - The User Interface settings (spring-dspace-addon-discovery-configuration-services.xml)
  - The Structure of spring-dspace-addon-discovery-configuration-services.xml
    - Mapping a discovery configuration to the home page or a specified community/collection
    - Creating a new discovery configuration bean
      - The structure of the discovery configuration bean
- Other Resources

DSpace Discovery is a Maintained Addon for DSpace XMLUI that replaces the default Search and Browse behavior with Apache Solr.

Proposal For Inclusion into DSpace 1.7.0

Recent work on porting DSpace to an Asyncronous build process prooved too large a task to be completed in DSpace 1.7 with all the other significant changes, with this in mind. It is proposed that a version of Discovery be delivered within the DSpace 1.7.0 codebase and initially maintained there for the 1.7.x development path.

This proposal includes the following features:

- Discovery will be compiled/installed into DSpace 1.7.0 XMLUI but left disabled by default
- The Appropriate Solr Search Core will be available and prepared to be started
- All dependencies for turning on Discovery will be available without recompilation by doing the following steps:
- Solr will be upgraded to the latest 1.4.1 release
- Documentation will be included which outlines exactly how to turn on Discovery and run it

Introduction Video

http://www.youtube.com/v/abRSXTUEwws

Documentation

*Discovery Configuration in DSpace 1.7.0
*Discovery Install in pre 1.7.0 HowTo

Design Premise for Discovery

The Design premise behind Discovery is to keep as much the implementation of Search and Browse independent of DSpace as possible. The basis for this is to twofold. (a) to reduce cost in maintaining any customized code and (b) to repurpose third party solutions wherever possible (a.k.a. standing on shoulder of giants). So, the basic tenants are:

1. Keep as much of the customization and configuration in Solr as possible.
2. Keep it as generic as possible.
3. Keep it as simple as possible.
4. In cases where configuration is outside Solr, provide pluggability to replace functionality easily at end user deployment.
5. Align Search/Browse capabilities with Solr capabilities, not other way around. This means, possibly abandon certain strategies for navigating via Browse if it proves these do not fit well with Solr.

RoadMap

Discovery is currently an addon for DSpace that still requires significant addition of configuration files to support. Planned releases will initially coincide with DSpace Scheduled Releases. Eventually, once completely stabilized, Discovery may be included into DSpace releases as a replacement for DSpace Search and Browse out of the box.

Issue Management

Issues for Discovery

http://jira.duraspace.org/browse/DSCR

key	summary	assignee	reporter	status

Can't show details. Ask your admin to add this Jira URL to the allowlist.

View these issues in Jira

Subversion Access

http://scm.dspace.org/svn/repo/modules/dspace-discovery

Examples in Production

- Dryad Data Repository: http://www.datadryad.org/search?query=&rpp=10&group_by=none&sort_by=score&order=DESC&submit=Go

Discovery in DSpace 1.8.0

New Discovery backend (unconfirmed)

The goal of this task was twofold.

1. Ensure that discovery doesn't have to use Solr as a backend, in other words ensure that user can plug in their own backend implementation.
2. Rewrite some of the code to make it more logical (example: the sidebar filters for a community page where rendered because the CommunityRecentSubmissionsTransformer extended the AbstractsFiltersTransformer => logically these are 2 completely different things).

This task focuses completely on these 2 points. Please bear in mind that almost nothing in the UI has been changed. While programming this task I also had a minor feature request which I implemented, namely the following:

The autocomplete filters can be configured not to split words on a space, this is particularly useful for authors.

The rewrite of the back end is explained below.

1. A new discovery sub-module has been created named "dspace-discovery-solr" this module is the only module that contains any Solr dependencies and contains the old "SolrServiceImpl" class and the spring-dspace-addon-discovery-services.xml Spring file. This module is used by default and ensures that a discovery out of the box will still work. None of the other discovery modules have dependencies on this module so it can be replaced very easily.
2. The "dspace-discovery-provider" module has lost the all the Solr dependencies as explained and the "SearchService" interface has undergone some changes, the search methods don't require a solrQuery anymore and do not return a solrResult. Instead 2 new objects have been created
    - DiscoverQuery: will hold all query information
    - DiscoverResult: will contain the resulting dspace objects and/or the facets
    - By doing things this way any programmer can implement the search service and write their own backend.
3. All the user interface classes have been rewritten to support these new objects.
4. One logical change was also made namely the following:
   The abstractFilters class has been transformed into the "SidebarFacetsTransformer". The facets will still appear on the same places, but unlike earlier where the only reason the community had sidebar facets was that the "CommunityRecentSubmissions" class extended the filters class the SidebarFacetsTransformer is now called on each community page. I did this because it just makes more sense that way.
5. I also added an extra method to indexing implementation of Solr named "indexItemFieldCustom", should it ever be required to adjust some simple indexing task the entire "SolrServiceImpl" does not need to be overwritten. One simply extends the class and implement that method (a boolean is returned which can prevent further indexing of the metadata field).

There is one thing that remains unfinished and that is the related items, I'm still thinking on the best way to implement that with the DiscoveryQuery /DiscoveryResult objects, if anybody has some suggestion I'm always willing to listen.

New discovery configuration (unconfirmed)

The configuration for discovery is located in 2 separate files.

- The **discovery.cfg** file located in thedspace.dir/config/modules directory, this file contains general discovery settings (the location of the solr server, which fields not to index, ...)
- The **spring-dspace-addon-discovery-configuration-services.xml** file located in dspace.dir/config/spring directory. This is a spring file that contains all the configuration for the user interface (Sidebar facet configuration, sort options, search filters, ...)

When changes are made to one of these files the tomcat needs to be restarted & a complete re index of the repository is required. To do this use the command line and navigate to the dspace directory and run the command below.

```
./bin/dspace update-discovery-index -f
```

The general discovery settings (discovery.cfg)

The discovery.cfg file is located in the dspace.dir/config/modules directory, it contains the following properties:

Property:	search.server
Example Value:	http://localhost:8080/solr/search
Informational Note:	Discovery relies on a SOLR index. This parameter determines the location of the SOLR index.
Property:	search.default.sort.order
Example Value:	search.default.sort.order=DESC
Informational Note:	The default sort order when searching in discovery, it can either be DESC or ASC.
Property:	index.ignore
Example Value:	dc.description.provenance,dc.language
Informational Note:	A comma separated list containing the metadata fields which are not to be indexed.

The User Interface settings (spring-dspace-addon-discovery-configuration-services.xml)

The file is located in the dspace.dir/config/spring/discovery directory.

The Structure of spring-dspace-addon-discovery-configuration-services.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-2.5.xsd"
    default-autowire-candidates="*Service,*DAO,javax.sql.DataSource">
    <context:annotation-config /> <!-- allows us to use spring annotations in beans -->


<!--Bean that is used for mapping communities/collections to certain discovery configurations-->
<bean id="org.dspace.discovery.configuration.DiscoveryConfigurationService" class="org.dspace.discovery.
configuration.DiscoveryConfigurationService">
        <property>
            <map>
                <!--The map containing all the settings,
                    the key is used to refer to the page (the "site" or a community/collection handle)
                    the value-ref is a reference to an identifier of the DiscoveryConfiguration format
                    -->
                <!--The default entry, DO NOT REMOVE the system requires this-->
               <entry key="default" value-ref="defaultConfiguration" />
                         ......
            </map>
        </property>
    </bean>


<bean id="defaultConfiguration" class="org.dspace.discovery.configuration.DiscoveryConfiguration" scope="
prototype">
        <!--Which sidebar facets are to be displayed-->
        <property name="sidebarFacets">
            <list>
            </list>
        </property>
        <!--The search filters which can be used on the discovery search page-->
        <property name="searchFilters">
            <list>
            ....
            </list>
        </property>
        <!--The sort filters for the discovery search-->
        <property name="searchSortFields">
            <list>
            ....
            </list>
        </property>
        <!--Any default filter queries, these filter queries will be used for all queries done by discovery for
this configuration-->
        <!--<property name="defaultFilterQueries">-->
            <!--<list>-->
            ....
            <!--</list>-->
        <!--</property>-->
        <!--The configuration for the recent submissions-->
        <property name="recentSubmissionConfiguration">
            <bean class="org.dspace.discovery.configuration.DiscoveryRecentSubmissionsConfiguration">
             ...
            </bean>
        </property>
    </bean>
```

Because this file is in XML format, you should be familiar with XML before editing this file. By default, this file contains the "defaultConfiguration" for discovery which contains the following settings:

- Sidebar facets configured:
    - **sidebarFacetAuthor:** contains the metadata fields dc.contributor.author & dc.creator with a facet limit of 10 and sorted by count
    - **sidebarFacetSubject:** contains all subject metadata fields (dc.subject.*) with a facet limit of 10 and sorted by count
    - **sidebarFacetDateIssued:** contains the dc.date.issued metadata field has a type "date" and is sorted by value

- The configured search filters:
    - **searchFilterTitle:** contains the dc.title metadata field and has a tokenized autocomplete
    - **searchFilterAuthor:** contains the dc.contributor.author & dc.creator metadata fields and has a non tokenized autocomplete configured
    - **searchFilterSubject:** contains the dc.subject.* metadata fields and has a non tokenized autocomplete configured
    - **searchFilterIssued:** contains the dc.date.issued metadata field with the type "date" and has a tokenized autocomplete

- The configured sort fields:
    - **sortTitle**: contains the dc.title metadata field
    - **sortDateIssued:** contains the dc.date.issued metadata field, this sort has the type date configured.

- The default filter queries are disabled by default but there is an example in the default configuration in comments which allows discovery to only return items (as opposed to also communities/collections).

- The recent submissions configuration is sorted by dc.date. accessioned which is a date and a maximum number of 5 recent submissions are displayed.

Many of the properties contain lists which use references to point to the configuration elements. This way a certain configuration type can be used in multiple discovery configurations so there is no need to duplicate these.**Adding a new discovery configuration**

Mapping a discovery configuration to the home page or a specified community/collection

```
<bean id="org.dspace.discovery.configuration.DiscoveryConfigurationService" class="org.dspace.discovery.
configuration.DiscoveryConfigurationService">
        <property name="map">
            <map>
                <!--The map containing all the settings,
                    the key is used to refer to the page (the "site" or a community/collection handle)
                    the value-ref is a reference to an identifier of the DiscoveryConfiguration format
                    -->
                <!--The default entry, DO NOT REMOVE the system requires this-->
               <entry key="default" value-ref="defaultConfiguration" />

               <!--Use site to override the default configuration for the home page & default discovery page-->
               <!--<entry key="site" value-ref="defaultConfiguration1" />-->
               <!--<entry key="123456789/7621" value-ref="defaultConfiguration2"/>-->
            </map>
        </property>
    </bean>
```

When adding a new discovery configuration an additional entry in the map of the bean with id org.dspace.discovery.configuration.DiscoveryConfigurationService is required. This map can contain as many entries as there are communties or collections.

The map contains one entry already the default one, it is not recommended to remove this one. Each entry requires 2 attributes. The first one is the **key** the key can contain the following values:

- **Default:** the default if no specific match is found
- **site:** the discovery configuration for the home page and also for dspace.url/discover
- **a handle:** the handle for a community or collection

The second attribute is the **value-ref** this value must refer to an existing configuration bean which contains the configuration for the facets, filters, ...

Creating a new discovery configuration bean

The structure of the discovery configuration bean

```
<bean id="defaultConfiguration" class="org.dspace.discovery.configuration.DiscoveryConfiguration" scope="
prototype">
        <!--Which sidebar facets are to be displayed-->
        <property name="sidebarFacets">
        ...
        </property>
        <!--The search filters which can be used on the discovery search page-->
        <property name="searchFilters">
        ...
        </property>
        <!--The sort filters for the discovery search-->
        <property name="searchSortFields">
            ...
        </property>
        <!--Any default filter queries, these filter queries will be used for all queries done by discovery for
this configuration-->
        <!--<property name="defaultFilterQueries">-->
            ...
        <!--</property>-->
        <!--The configuration for the recent submissions-->
        <property name="recentSubmissionConfiguration">
            <bean class="org.dspace.discovery.configuration.DiscoveryRecentSubmissionsConfiguration">
                ...
            </bean>
        </property>
    </bean>
```

Creating a new discovery bean

Start by creating a new bean with an identifier specified in the mapping section from the previous point and ensure that it has the following attributes:

- **class:** org.dspace.discovery.configuration.DiscoveryConfiguration
- **scope:**prototype

```
<bean id="{identifier}" class="org.dspace.discovery.configuration.DiscoveryConfiguration" scope="
prototype">
</bean>
```

Configuring sidebar facets

Add a new element named **property** and the attribute **name="sidebarFacets"** and add a subelement list. This property is mandatory by the discovery configuration.

```
<bean id="{identifier}" class="org.dspace.discovery.configuration.DiscoveryConfiguration" scope="prototype">
        <property name="sidebarFacets">
                <list>
                </list>
        </property>
</bean>
```

In this list the user can add sidebar configuration beans, if the list is left empty no sidebar facets will be displayed. Each subelement of the list is a **ref** which has one attribute named "**bean**" the value of this bean is a reference to an identifier which will contain all the configuration of the sidebar facet.

Below is an example of how the list can be configured.

```
<property name="sidebarFacets">
    <list>
        <ref bean="sidebarFacetAuthor" />
        <ref bean="sidebarFacetSubject" />
        <ref bean="sidebarFacetDateIssued" />
    </list>
</property>
```

Each of these properties refers to another bean which must be configured in the file.

The structure of a sidebar facet bean looks like this:

```
<bean id="{sidebar.facet.identifier}" class="org.dspace.discovery.configuration.SidebarFacetConfiguration">
    <property name="indexFieldName" value="{index.field.name}"/>
    <property name="metadataFields">
        <list>
            <value>{metadata.field}</value>
            <value>{metadata.field}</value>
        </list>
    </property>
    <property name="facetLimit" value="{facet.limit}"/>
    <property name="sortOrder" value="{ COUNT  or VALUE} "/>
    <property name="type" value="{text or value} "/>
</bean>
```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below.

- **indexFieldName** (Required)**:** A unique sidebarfacet field name, the metadata will be indexed under this field name
- **metadataFields** (Required): A list containing the metadata fields which are to be shown in the sidebar facets.
- **facetLimit** (optional)**:** The maximum number of sidebar facets to be shown, this property is optional, if none is specified 10 will be used. When a type of **date** is given, this property will not be used since dates are grouped years.
- **sortOrder** (optional)**:**The sort order for the sidebar facets, it can either be COUNT or VALUE. If none is given the COUNT value is used as a default.
    - **COUNT** Facets will be sorted by the amount of times they appear in the repository
    - **VALUE** Facets will be sorted alphanumeric
- **type**(optional): the type of the sidebar facet it can either be date or text, if none is defined text will be used.
    - **text:** The facets will be treated as is
    - **date**: With dates only the year is indexed and sidebar facets are grouped by these years and a drill down is used

Example of a sidebar facet configuration bean

```
<bean id="sidebarFacetAuthor" class="org.dspace.discovery.configuration.SidebarFacetConfiguration">
        <property name="indexFieldName" value="author"/>
        <property name="metadataFields">
            <list>
                <value>dc.contributor.author</value>
                <value>dc.creator</value>
            </list>
        </property>
        <property name="facetLimit" value="10"/>
        <property name="sortOrder" value="COUNT"/>
    </bean>
```

Configuring search filters

Search filters can be used on the discovery search page to further filter the discovery results. These filters have an autocomplete option.

Start of by adding an element named **property** with the attribute **name="searchFilters"** afterworths create a sub element list. The searchFilters property is mandatory.

```
<property name="searchFilters">
    <list>
        ...
    </list>
</property>
```

Like the sidebar facets the list also contains sublements named **ref** and with the attribute **bean** referencing (in this case) a search filter configuration. If this list is empty no search filters will be displayed. Below is an example of the filters.

```
<list>
    <ref bean="searchFilterTitle"/>
    <ref bean="searchFilterAuthor"/>
    <ref bean="searchFilterSubject"/>
    <ref bean="searchFilterIssued"/>
</list>
```

Each of these properties refers to another bean which must be configured in the file.

The structure of a sidebar facet bean looks like this:

```
<bean id="{bean.identifier}" class="org.dspace.discovery.configuration.DiscoverySearchFilter">
    <property name="indexFieldName" value="{index.field.name}"/>
    <property name="metadataFields">
        <list>
            <value>{metadata.field.1}</value>
            <value>{metadata.field.2}</value>
        </list>
    </property>
    <property name="fullAutoComplete" value="{true or false}"/>
    <property name="type" value="{text or value} "/>
</bean>
```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below.

- **indexFieldName** (Required)**:** A unique search filter field name, the metadata will be indexed under this field name
- **metadataFields** (Required): A list containing the metadata fields which can be used in this filter
- **fullAutoComplete** (optional): If set to true the values indexed for autocomplete will not be tokenized, if set to false tokenization will occur
- **type**(optional): the type of the search filter it can either be date or text, if none is defined text will be used.
    - **text:** The metadata will be treated as is
    - **date**: With a type of date the dates will receive the following format: yyyy-MM-dd (2011-07-01)

Example of a search filter configuration bean

```
<bean id="searchFilterAuthor" class="org.dspace.discovery.configuration.DiscoverySearchFilter">
    <property name="indexFieldName" value="author"/>
    <property name="metadataFields">
        <list>
            <value>dc.contributor.author</value>
            <value>dc.creator</value>
        </list>
    </property>
    <property name="fullAutoComplete" value="true"/>
</bean>
```

Configuring sort options

Sort options are used in the discovery search page, by default there is always one sort option (relevance). The structure of the sort options looks like this:

```
<property name="searchSortFields">
    <list>
        ...
    </list>
</property>
```

Like the other properties the list also contains sublements named **ref** and with the attribute **bean** referencing (in this case) a sort option configuration. If this list is empty the only sort option available will be. Below is an example of the sort options.

```
<list>
    <ref bean="sortTitle"/>
    <ref bean="sortDateIssued"/>
</list>
```

Each of these properties refers to another bean which must be configured in the file. The structure of a sort option bean looks like this:

```
<bean id="{bean.identifier}" class="org.dspace.discovery.configuration.DiscoverySortConfiguration">
    <property name="metadataField" value="{metadata.field}"/>
    <property name="defaultSort" value="{true or false} "/>
    <property name="type" value="{text or date}"/>
</bean>
```

The id & class attributes are mandatory for this type of bean. The properties that it contains are discussed below.

- **metadataField** (Required): The metadata field indicating the sort values
- **defaultSort** (Optional): A boolean indicating which sort filter should be the default one.
- **type** (optional): the type of the search filter it can either be date or text, if none is defined text will be used.

Example of a sort option configuration bean.

```
<bean id="sortTitle" class="org.dspace.discovery.configuration.DiscoverySortConfiguration">
        <property name="metadataField" value="dc.title"/>
        <property name="defaultSort" value="true"/>
 </bean>
```

Default filter queries

The default queries are queries that are used on all queries linked to the configuration block they are in. So these queries are used to retrieve the results, the sidebar filters, ...

The filter queries element is an entirely optional property.

The layout of this property is displayed below.

```
<property name="defaultFilterQueries">
    <list>
        <value>query1</value>
        <value>query2</value>
    </list>
</property>
```

This property contains a simple list which in turn contains the queries. Some examples of queries:

- search.resourcetype:2
- dc.subject:test
- dc.contributor.author: "Van de Velde, Kevin"
- ...

Recent submissions configuration

The recent submissions configuration element contains all the configuration settings to display the list of recently submitted items on the home page or community/collection page. Because the recent submission configuration is in the discovery configuration block, it is possible to show 10 recently submitted items on the home page but 5 on the community/collection pages.

The layout of the recently submitted is displayed below:

```
<property name="recentSubmissionConfiguration">
    <bean class="org.dspace.discovery.configuration.DiscoveryRecentSubmissionsConfiguration">
        <property name="metadataSortField" value="{metadata.field}"/>
        <property name="type" value="{text or date} "/>
        <property name="max" value="{max}"/>
    </bean>
</property>
```

The property name & the bean class are mandatory. The property field names are discusses below.

- *metadataSortField (*mandatory): The metadata field to sort on to retrieve the recent submissions
- **max** (mandatory): The maximum number of results to be displayed as recent submissions
- **type** (optional): the type of the search filter it can either be date or text, if none is defined text will be used.

Below is an example configuration of the recent submissions.

```
<property name="recentSubmissionConfiguration">
    <bean class="org.dspace.discovery.configuration.DiscoveryRecentSubmissionsConfiguration">
        <property name="metadataSortField" value="dc.date.accessioned"/>
        <property name="type" value="date"/>
        <property name="max" value="5"/>
    </bean>
</property>
```

Deploying the custom discovery configuration

The DSpace web application only reads your custom configuration when it starts up, so it is important to remember:

```
You must always restart Tomcat (or whatever servlet container you are using) for changes made to the spring
file to take effect.
```

When the tomcat has restarted there is an option to check if the changes you made to the spring file are indeed valid. You can do this by running the command below in a command line interface.

```
./bin/dspace dsrun org.dspace.discovery.configuration.DiscoveryConfigurationService
```

This command will print the current configuration if it is indeed valid. After verifying that the configuration is correct a complete re index of the discovery index is required.

To do this use the command line and navigate to the dspace directory and run the command below.

```
./bin/dspace update-discovery-index -f
```

Other Resources

- http://lucene.apache.org/solr/
- AJAX Integration: http://github.com/evolvingweb/ajax-solr http://solrjs.solrstuff.org/
- Integration With DSpace REST project for unified Search and Browse in the REST webapplication as well.
- Access Control Request Handler for Solr Access control for Solr Service, Documents and Fields.
- General specifications
- FieldType JAVADOC
- Predefined FieldTypes in SolR
- http://wiki.github.com/evolvingweb/ajax-solr/reuters-tutorial-step-9 Integrate AutoSuggest, Tag Clouds, Google Maps and dynamic Paging into DSpace Search Results.