Customizing GSearch and Solr

The out-of-the box functions will allow you to support all of our current solution packs, and the MODS and DC metadata streams associated with them. Once you have GSearch installed and running there is very little you need to do. However, you may wish to customize Solr to index a new metadata schema (if you are creating a custom content model) or if you want to modify existing fields. To do this, you would modify the foxmlToSolr.xslt located in the GSearch webapps directory. If you followed the instructions for installing GSearch in Chapter 9 - Enabling Indexing & Searching with Solr, the file would be located here:

```
/usr/local/fedora/tomcat/webapps/fedoragsearch/WEB-INF/classes/config/index/gsearch_solr
```

For example, to add the Darwin Core to the index you can add the following lines to the xslt:

```
<xsl:for-each select="foxml:datastream/foxml:datastreamVersion\[last()\]/foxml:
xmlContent/dwc:SimpleDarwinRecordSet/dwc:SimpleDarwinRecord/*">
<xsl:if test="text() \[normalize-space(.) \]"><\!\--don't bother with empty space-\->

<field >
<xsl:attribute name="name">
<xsl:value-of select="concat('dwc.', substring-after(name(),':'))"/>
</xsl:attribute>
<xsl:value-of select="normalize-space(text())"/>
</field>

</xsl:if>
</xsl:for-each>
```

The xsl above will index most Darwin Core fields. Once GSearch is aware of the new schema, you can make Solr aware of it by modifying the schema.xml. Quite often, it makes sense to assign the same content to two fields:

- one analyzed (tokenized, lower cased etc.) for searching
- one unanalyzed (stored exactly as is) for displaying in facets, etc.

The xsl above will create many fields one of which would be dwc.language. In the Solr Schema we would add a declaration for this field:

```
<field name="dwc.language" type="text" indexed="true" stored="true" multiValued="true"/>
```

Here, we have given it a type = “text”, which in the default schema is analyzed.

```
<fieldType name="text" class="solr.TextField" positionIncrementGap="100"><analyzer
type="index"><tokenizer class="solr.WhitespaceTokenizerFactory"/><!-- in this example,
we will only use synonyms at query time
        <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt"
ignoreCase="true" expand="false"/>
        --><filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.
txt"/><filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0"/><filter
class="solr.LowerCaseFilterFactory"/><filter class="solr.EnglishPorterFilterFactory"
protected="protwords.txt"/><filter class="solr.RemoveDuplicatesTokenFilterFactory"/><
/analyzer><analyzer type="query"><tokenizer class="solr.WhitespaceTokenizerFactory"
/><filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
expand="true"/><filter class="solr.StopFilterFactory" ignoreCase="true" words="
stopwords.txt"/><filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0"/><filter
class="solr.LowerCaseFilterFactory"/><filter class="solr.EnglishPorterFilterFactory"
protected="protwords.txt"/><filter class="solr.RemoveDuplicatesTokenFilterFactory"/><
/analyzer></fieldType>>
```

The types are also defined in the schema.xml. If we want to use this in a filter, it makes sense to also store it unanalyzed under a different name. This requires two more entries in the schema.xml:

```
<field name="language" type="string" maxChars="300" indexed="true" stored="true"
multiValued="true"/>
<copyField source="dwc.language" dest="language"/>
```

Once we have created a field named language to store the unanalyzed data in, we'll use copyField to copy the dwc.language field into the language field which will happen during indexing before it is analyzed. Notice the type is now defined as a string. We can now use these fields in solr request handlers. Request handlers determine what fields to search and what to return, and you can assign certain fields more weight than others.

A request handler may look like this:

```
<requestHandler name="herbarium" class="solr.SearchHandler" default="true">

<\!-\- default values for query parameters -->

<lst name="defaults">

<str name="echoParams">explicit</str>

<str name="qf">dwc.type^2.0 dwc.language^2.0 dwc.rightsHolder^2.0 dwc.accessRights^2.0
dwc.rights^2.0 dwc.basisOfRecord^2.0 dwc.scientificName^2.0 dwc.vernacularName^2.0 dwc.
kingdom^2.0 dwc.phylum^2.0 dwc.class^2.0 dwc.order^2.0 dwc.family^2.0 dwc.genus^2.0dwc.
specificEpithet^2.0 dwc.continent^2.0 dwc.country^2.0 dwc.countryCode^2.0 dwc.
stateProvince^2.0 dwc.county^2.0 dwc.municipality^2.0 dwc.verbatimLocality^2.0 dwc.
decimalLatitude^2.0 dwc.decimalLongitude^2.0 dwc.occurrenceID^2.0 dwc.institutionCode^2.
0 dwc.collectionCode^2.0 dwc.catalogNumber^2.0 dwc.recordedBy^2.0 dwc.eventDate^2.0
PID^0.5</str>

<str name="fl">rightsHolder, accessRights, rights,basisOfRecord, scientificName,
vernacularName, kingdom, phylum, class, order, family, genus, specificEpithet,
continent, country,countryCode,stateProvince, county, municipality, verbatimLocality,
decimalLatitude, decimalLongitude, occurrenceID, institutionCode, collectionCode,
catalogNumber, recordedByeventDate, PID</str>

<str name="q.alt">*:*</str>
</lst>

<lst name="appends">
<str name="fq">PID:herbarium*</str>
</lst>

</requestHandler>
```

Some interesting things to take note of:

The request handler example shown above limits the results to objects that have the herbarium namespace.

In the qf , we are searching fields like dwc.type and dwc.language and they are all weighted the same. We can tweak the weights later if we wish to customize the results. Solr returns are the fields in <str name=”fl”> element. This gives us nice values to use when displaying the results and when listing facets.