LucidWorks™

Business Rules Module

1.0 Documentation

Created: 10-Sep-2013

Table of Contents

Business Rules Module

The Business Rules module allows integrating the Drools business rules engine with Solr. Business rules provide a number of features, and the Drools engine allows applying complex rules without coding all of the logic into your search application.

For example, with business rules you can:

- Dynamically modify searches based on terms entered, user profile information or other factors.
- Dynamically choose and modify facets.
- Modify documents while being indexed.

This documentation covers:

- How to install the Business Rules module and use it with your existing Solr implementation
- How to configure business rules in your `solrconfig.xml` file
- How to write rules and how to tweak them for your needs
- Details of included example rules

The free download includes a 60-day license for use of the module. After the 60-day trial period, the software will stop working. If you would like to purchase this module, please contact sales@lucidworks.com for pricing and further details.

Installing the Business Rules Module

System Requirements

Solr 4.0 or higher must be installed before using the Business Rules module, and any Solr system requirements also apply to Business Rules. The installation process requires making changes to your existing Solr instance(s), so there is not an option to locate the module on another server.

If possible, it's recommended to have at least 4Gb of memory on the system that will run Solr and business rules together.

The module embeds the core Drools 5.5 engine with your Solr application, and any required dependencies are included with the `.jar` files. However, it may be worth taking a look at the Drools System Requirements (note that Drools Guvnor is not included at this time).

In addition, you should be already familiar with Solr requestHandlers, searchComponents, and updateProcessors generally, as those will be used to run the rules. If you are not yet familiar with those concepts, please also review the Solr Reference Guide section on Configuring solrconfig.xml.

Installing the Business Rules package

Step 1: Download and Unpack

There is a single package available for all Solr 4.x releases.

After downloading the installation package, move the `.zip` file to any preferred location. The `.jar` files will be added to Solr, and the module will run with Solr, so the package can be located anywhere while you copy files to your Solr installation.

Unzip the package, and you will find the following directories and files:

```
conf/
examples/
lib/
README.md
rules/
```

Step 2: Copy .jar files to Solr

In the `lib` directory of the downloaded package, there are a number of `.jar` files. These should all be copied to the `lib` directory of your Solr instance(s).

In a default Solr installation, this directory is found at `example/solr/lib`. Note that a brand new installation of Solr does not contain a `lib` directory, and a running Solr also will not contain this directory unless you have specifically created it in the past for other plugins. This is not the same as the `lib` directory found at `example/lib`, but is a level lower in the actual Solr instance (the `example` directory contains several separate Solr instances). If there is no `lib` directory, create it, and add the `.jar` files.

Solr supports an alternate approach for adding `.jar` files, which is to use a `lib` directive in `solrconfig.xml`. That **does not work** with the Business Rules module due to the way the Drools import function works. The `.jar` files must be added to the main Solr instance's `lib` directory.

Step 3: Add Rules to Solr

Rules are created by defining a rule in the Drools language, and saving the instructions to a `.drl` file. These rules files are then defined in Solr's `solrconfig.xml` file. In theory, the rules can be stored anywhere, but to avoid possible permission or other path errors, the simplest approach is to add a `rules` subdirectory the the `conf` directory of each collection that will use rules.

Copy the `rules` directory of the downloaded package to the `conf` directory of your Solr collection. If you have multiple collections, and they will each use rules, you will want to make copy the `rules` directory to the `conf` dir of each directory.

Step 4: Add Configuration Settings to Solr

The Business Rules module allows applying defined rules to queries. Since most of those settings are in `solrconfig.xml`, there are a number of settings to modify. The settings that need to be applied are in the `conf` directory of the downloaded package, in a file called `drools-snippet-xml.txt`. The simplest approach to get started is to copy the settings to the beginning of your `solrconfig.xml` file and save it.

The section Business Rules Configuration describes in more detail the configuration changes that are being made, and how to integrate those changes with your existing configuration.

Step 5: Restart Solr

Once each step is complete, start Solr. The standard way to start Solr is to use `start.jar`, but you may have another approach for your environment.

There is nothing within the downloaded Business Rules package itself that needs to be started.

Business Rules Configuration

In order to properly use Business Rules with Solr, several modifications to your local `solrconfig.xml` should be made. The configuration changes that need to be made are found in the `conf` directory of your downloaded Business Rules module package, in a file called `drools-snippet-xml.txt`.

The configuration has several parts:

- a requestHandler named "/update-with-rules"
- a requestHandler named "/update-extract-with-rules"
- a requestHandler named "/search-with-rules"
- a requestHandler named "/rulesMgr"

- a searchComponent named "landingPage"
- a searchComponent named "firstRulesComp"
- a searchComponent named "lastRulesComp"

- an updateRequestProcessorChain named "update-with-rules-chain"

- a document transformer named "rules"

These configurations allow you to apply defined rules during indexing and at specific points while Solr is constructing the query response.

Configurations in this section:

The rest of this section will describe each one, and discuss how to integrate it with an existing Solr system. If you are not yet familiar with requestHandlers, searchComponents and similar configurations in a `solrconfig.xml` file, you may want to review the Solr Reference Guide section RequestHandlers and SearchComponents in SolrConfig.

RequestHandlers

/update-with-rules

This is an updateRequestHandler for indexing documents. Note that it calls the updateRequestProcessorChain, defined later. This allows using rules to alter documents while they are being indexed, using Solr's standard updateRequestHandler class.

```
<requestHandler name="/update-with-rules" class="solr.UpdateRequestHandler">
   <lst name="defaults">
      <str name="update.chain">update-with-rules-chain</str>
   </lst>
</requestHandler>
```

The "/update-with-rules" requestHandler works in a similar way to the default "/update" requestHandler and takes the same parameters when used. As with the default "/update" requestHandler, in Solr 4.x versions, you can use this one handler to send documents to Solr as CSV, JSON, and XML files.

/update-extract-with-rules

This is another updateRequestHandler for indexing documents with rules, and it also calls the updateRequestProcessorChain. However, this requestHandler is based on Solr's ExtractingRequestHandler, which allows you to use Tika to extract content from complex files such as Word documents, PDF files, and binary files.

```
<requestHandler name="/update-extract-with-rules"
    startup="lazy"
    class="solr.extraction.ExtractingRequestHandler" >
   <lst name="defaults">
      <str name="update.chain">update-with-rules-chain</str>
      <str name="lowernames">true</str>
      <str name="uprefix">ignored_</str>
      <!-- capture link hrefs but ignore div attributes -->
      <str name="captureAttr">true</str>
      <str name="fmap.a">links</str>
      <str name="fmap.div">ignored_</str>
   </lst>
</requestHandler>
```

Because this requestHandler is based on the ExtractingRequestHandler, it allows the same parameters.

/search-with-rules

This is a requestHandler which provides an example rules-based search. Note in the configuration below that we have defined two arrays, "first-components" and "last-components" and named specific searchComponents.

```
<requestHandler name="/search-with-rules" class="solr.SearchHandler">
   <lst name="defaults">
      <str name="echoParams">explicit</str>
      <int name="rows">10</int>
      <str name="df">text</str>
   </lst>
   <arr name="first-components">
      <str>landingPage</str>
      <str>firstRulesComp</str>
   </arr>
   <arr name="last-components">
      <str>lastRulesComp</str>
   </arr>
</requestHandler>
```

If you want to integrate rules with an existing requestHandler, you can add the named searchComponents to the handler, in the same way shown in this example.

/rulesMgr

The rulesMgr handles references to rules engine instances. Each of the engines are defined and used by the searchComponents.

```
<requestHandler class="com.lucid.rules.RulesEngineManagerHandler" name="/rulesMgr">
    <!-- Engines can be shared, but they don't have to be.  A SearchComponent or other
consumer can
         specify the engine they want by name.
     -->
   <lst name="engines">
      <lst name="engine">
         <str name="name">first</str>
         <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
         <lst name="rules">
            <str name="file">rules/defaultFirst.drl</str>
         </lst>
      </lst>
      <lst name="engine">
        <str name="name">landing</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultLanding.drl</str>
        </lst>
      </lst>
      <!-- Engine is using rules that are designed to be called after all other
components -->
      <lst name="engine">
        <str name="name">last</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultLast.drl</str>
        </lst>
      </lst>
      <lst name="engine">
        <str name="name">docs</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultDocs.drl</str>
        </lst>
      </lst>
    </lst>
  </requestHandler>
```

SearchComponents

landingPage

The landingPage searchComponent is generally used to define a specific result for conditions that match the rule. For example, you could redirect users to a specific page of the website in response to a query, or you could highlight specific documents for a query in combination with other factors such as time of day, or user attributes.

```
<searchComponent class="com.lucid.rules.LandingPageComponent" name="landingPage">
   <str name="requestHandler">/rulesMgr</str>
   <str name="engine">landing</str>
    <!-- The handle can be used to turn on or off explicit rules components in the
         case when you have multiple rules at different stages of the component
ordering
     -->
   <str name="handle">landing</str>
</searchComponent>
```

firstRulesComp

The firstRulesComp is a searchComponent which is meant to be placed within the "first-components" capability of Solr. This allows applying a rule before other searchComponents have been applied. An example of this might be to limit search results with parameters not entered by the user (which may be conditional depending on the user, or other factors). Then other searchComponents, such as faceting or highlighting, can be applied to the reduced result set.

```
<searchComponent class="com.lucid.rules.RulesComponent" name="firstRulesComp">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">first</str>
    <!-- The handle can be used to turn on or off explicit rules components in the
         case when you have multiple rules at different stages of the component
ordering-->
    <str name="handle">first</str>
  </searchComponent>
```

lastRulesComp

The lastRulesComp is a searchComponent which is meant to be placed within the "last-components" capability of Solr. This allows applying a rule after other searchComponents have been applied.

```
<searchComponent class="com.lucid.rules.RulesComponent" name="lastRulesComp">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">last</str>
    <str name="handle">last</str>
  </searchComponent>
```

UpdateRequestProcessorChain

The "update-with-rules-chain" is a custom updateRequestProcessorChain that defines the `RulesUpdateProcessorFactory`, as well as two standard updateRequestProcessor classes. This allows you to make transformations to documents while they are being indexed. Note that the example "/update-with-rules" and "/update-extract-with-rules" requestHandlers both call this chain definition.

```
<updateRequestProcessorChain name="update-with-rules-chain" default="true">
    <processor class="com.lucid.rules.RulesUpdateProcessorFactory">
        <str name="requestHandler">/rulesMgr</str>
        <!-- we re-use the engine, but we could have an independent one-->
        <str name="engine">docs</str>
        <!-- Each one should have it's own handle, as you can have multiple in the
chain -->
        <str name="handle">docProc</str>
    </processor>
    <!-- <processor class="com.lucid.update.DistributedUpdateProcessorFactory"> -->
    <!-- example configuration... "shards should be in the *same* order for
        every server in a cluster. Only "self" should change to represent what server
        *this* is. <str name="self">localhost:8983/solr</str> <arr name="shards">
        <str>localhost:8983/solr</str> <str>localhost:7574/solr</str> </arr> -->
    <!-- </processor> -->
    <processor class="solr.LogUpdateProcessorFactory">
        <int name="maxNumToLog">10</int>
    </processor>
    <!-- <processor class="com.lucid.update.FieldMappingUpdateProcessorFactory" /> -->
    <!-- last two are standard -->
    <processor class="solr.DistributedUpdateProcessorFactory" />
    <processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

This example includes a couple of other classes that are not used (they are commented out of the definition). One of those is the `DistributedUpdateProcessorFactory`, which allows updates in SolrCloud mode.

If you already have a chain defined, you could add only the section that defines the rules processor to your existing chain.

Transformer

The document transformer allows applying rules that alter documents during query time. It is invoked as part of Solr's response and can inject or modify fields before they are returned.

```
<transformer name="rules" class="com.lucid.rules.RulesDocTransformerFactory">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">docs</str>
  </transformer>
```

Note that alterations to documents made with this transformer are not saved to the documents themselves. If you want to make changes that are saved with documents, use the UpdateRequestProcessorChain instead.

Rules with Index Replication

If you are using what is now considered "old-style" replication (i.e., you are not using SolrCloud), you should add the rules files to the `confFiles` list of configuration files that are copied to the slave servers with each update.

```
<!-- Optional -->
<!-- If using older v3 style master/slave replication, instead of 4x SolrCloud,
     add these files to your master confFiles list
     <str
name="confFiles">...,rules/defaultFirst.drl,rules/defaultLast.drl,rules/defaultLanding.drl
<requestHandler name="/replication" class="solr.ReplicationHandler" >
       <lst name="master">
         <str name="replicateAfter">commit</str>
         <str name="replicateAfter">startup</str>
         <str
name="confFiles">schema.xml,stopwords.txt,rules/defaultFirst.drl,rules/defaultLast.drl,rul
</lst>
       <lst name="slave">
         <str name="masterUrl">http://your-master-hostname:8983/solr</str>
         <str name="pollInterval">00:00:60</str>
       </lst>

  </requestHandler>
```

Writing Rules

The Business Rules module integrates Drools 5.5 with Solr. The Drools Rule Language Reference provides a much more thorough overview, but the below can serve as a brief introduction.

In Drools, rules are defined with Java-like declarations. While the software is meant to be easier for non-programmers to write rules, it is still a heavily technical syntax and assumes some technical proficiency.

To help you with writing rules, we have provided a `DroolsHelper.java` class which consists of helper functions to make the task easier. You can find this class in the `solr-business-rules.jar` file (the full name may include version numbers, but you should only have one `.jar` starting with `solr-business-rules`) found in the `lib` directory of the downloaded package, or in `example/solr/lib`, if you have moved the `.jars` as per the installation instructions.

Rules Files

A rules file has a file extension of `.drl`. For the Business Rules module, we have placed the rules in the `conf` directory of each Solr collection, in a sub-directory called `rules`. The example configurations assume this path; if they are located in another area of the filesystem, the examples will need to be updated.

Before starting the rule declarations, the package is defined, as are any imports and globals. The import statements are similar to import statements in Java, where you specify the fully qualified paths and type names for objects that will be used with the rules. The global statements allow you to make application objects available to the rules, such as if there is data or services the rules use.

Rule Declarations

At it's simplest, a rule declaration looks like this:

```
rule "name"
   <a set of attributes>
   when
      <a set of qualifying conditions, in Drools called "Left Hand Side">
   then
      <a set of actions to perform, in Drools called "Right Hand Side">
end
```

rule and Attributes

The first step is to state you are going to define a rule, simply with `rule` and a name of the rule.

Next, you can define attributes for the rule, which influence the behavior of the rule. One of the most important of these is `no-loop`, which prevents an infinite loop if a rule modifies a fact that causes the rule to activate again. There are several other attributes, however, which may be important to your rule. See the Drools documentation on Rule Attributes for more information.

when Conditions

In Drools language, the conditions that must be met for a rule to fire are also called "Left Hand Side".

Conditions work on one or more patterns, which include the object and constraints. For example, a condition like `$rb: ResponseBuilder( $qStr : req.params.get("q") matches "(?i).*ipod.*" ))` will match queries sent to Solr containing the term "ipod". What's going on in this example?

First, we've declared that the variable `$rb` will match the object `ResponseBuilder`. The `ResponseBuilder` is a Solr class that builds the query responses. The rest of the condition states we want to look at what the value was for Solr's `q` parameter, and match queries that contain the term "ipod".

There are multiple variations on how to declare the conditions. You can use Java expressions, booleans, binding variables, maps, and many more. Refer to the Drools documentation on Left Hand Side (when) syntax for all of the options and details on how to use them.

then Actions

In Drools language, the actions of a rule are also called "Right Hand Side". These are the changes that should be made to the "facts" known to the rules engine. In search, this would be changes to documents, the order of results, or other impacts on the results of the user's query. Keep in mind that these actions should not be conditional (as in, "`when` this, maybe this"), but atomic, meaning all of the stated actions should be performed (as in, "`when` this, `then` this"). If you find you need further conditions, you may want to consider breaking your rule into smaller pieces to achieve this goal.

As with `when` conditions, there are multiple variations on how to use `then` actions. Of particular assistance here is the `DroolsHelper.class`, found in the `solr-business-rules-0.1-solr-4.4.0.jar`, where several methods have been pre-defined such as `addToResponse`, which allows adding a key-value pair to the response, and `modRequest`, which modifies the request to Solr.

Refer to the Drools documentation on Right Hand Side (then) for more details.

Related Topics

There are several rules provided as examples, which may help you get started with the rules language. See Example Rules for a walk-through of two examples, plus an overview of other included examples.

Example Rules

Several example rules are provided in the `examples` directory of the module download.

In this section we'll pick a couple of the rules and walk through them.

Sample Rule Files

The example rules are designed to be used with the example documents provided by Solr. Each file includes extensive comments that explain what they are doing and how to use them with the sample documents that are included with Solr.

In most cases, the recommendation is to add new rules to the files in the `rules` directory (which was added to the `conf` directory of each Solr core during installation; see also Step 3: Add Rules to Solr in the installation instructions).

While it's possible to define multiple rules files in `solrconfig.xml` (in the `/rulesMgr` requestHandler section, it is simpler to use a single rules file (when possible) for each rules engine. This keeps all your rules in one place, making them easier to manage. You can modify the name of the single file if you'd like, just be sure to update the `/rulesMgr` requestHandler appropriately.

The following rules are included as examples:

Filename	Rule Type	What It Does
`defaultDocs-create-title.drl`	Indexing rule	Adds title fields to incoming documents.
`defaultDocs-manufacturer-check.drl`	Indexing rule	Copies the document ID field to the `manu` field on documents where `manu` is blank.
`defaultDocs-price-check.drl`	Indexing rule	Checks the price of an incoming document and adds a label when it matches a specific criteria. This approach is designed for times when using text (i.e., JSON, XML) codecs for indexing.
`defaultDocs-price-check-long-form.drl`	Indexing rule	An alternate approach to price checks. This approach is designed for times when using binary (i.e., Javabin) codecs for indexing.
`defaultFirst-apple.drl`	Query rule	Adds a defined `manu` field to all searches for a specific term.

defaultFirst-facets-part1of2.drl	Query rule	First of two steps to modify a facet; injects a facet query and alters the facet limit.
defaultFirst-from-readme-file.drl	Query rule	Adds a term to the query.
defaultFirst-model-number.drl	Query rule	Defines a method to find model numbers in a query, and if found looks in the ID field for a match.
defaultFirst-show-phases.drl		Demonstrates the phases of filtering.
defaultLanding-belkin.drl	Landing rule	Returns a specific URL in response to a query, which can be used by the front-end to either redirect the user or display it a specific way.
defaultLast-facets-part2of2.drl	Query rule	Part two of the earlier rule to modify a facet; injects the facet to the response.

Detailed Examples

README Example

This example is included in the file `defaultFirst-from-readme-file.drl`. The goal of this rule is to add query terms to a search when the user enters a specific string.

First, here is the text of the rule (note, this isn't the whole file, just the part that defines a rule; be sure to look at the whole rule for important comments on how to run it).

```
rule "electronics"
no-loop
when
    $rb: ResponseBuilder($qStr : req.params.get("q") == "text:electronics");
then
    addToResponse($rb, "origQuery", $qStr);
    addToResponse($rb, "modQuery", "text:electronics text:apache");
    modRequest($rb, "q", "text:electronics text:apache");
end
```

Let's step through this example in detail.

Line 1 states we are declaring a rule and gives it the name "electronics".

Line 2 says to only run the rule once to prevent an infinite loop. In this case, our `when` statement looks for the query term "electronics" on the field "text"; after the modifications from the rule, the query will still match the rule, which could make it fire again. Using `no-loop` prevents the rule firing over and over.

Line 3 starts the `when` conditions.

Line 4 uses Solr's ResponseBuilder to analyze the query, and match when the query (in the `q` parameters of the request sent to Solr) matches "text:electronics". Note this line is also setting a variable `$qStr`, and assigning it the query and parameters. This variable will be used again later.

Line 5 starts the `then` actions.

Line 6 defines a key/value pair for the ResponseBuilder of "origQuery" and the query string variable defined in line 4 (`$qStr`.

Line 7 defines another key/value pair for the ResponseBuilder of "modQuery", and the modified query string.

Line 8 modifies the request to the ResponseBuilder with a key/value pair, modifying the user's entry to include "text:apache" as well as what was initially entered.

Line 9 ends the rule.

To run this rule, once the rule has been added to `rules/defaultFirst.drl`, you can use the 'search-with-rules' requestHandler that was added to your `solrconfig.xml` file during installation. Then you can send a request to Solr that looks something like this:

```
http://localhost:8983/solr/collection1/search-with-rules?q=text:electronics&rules=true&rul
```

The request should be customized for your hostname and port, and this example also assumes you have indexed Solr's sample documents in the `example/exampledocs` directory.

Landing example

This example is included in the file `defaultLanding-belkin.drl`. The goal of this rule is to force Solr to return a document first in the list when a specific manufacturer ("Belkin") is entered by the user.

First, here is the text of the rule (note, this isn't the whole file, just the part that defines a rule; be sure to look at the whole rule for important comments on how to run it).

```
rule "Landing Page"
no-loop
when
    $rb: ResponseBuilder($qStr : req.params.get("q") == "manu:Belkin");
then
    addToResponse((NamedList)$rb.rsp.getValues().get("responseHeader"), "landingPage",
"http://www.Belkin.com");
end
```

This rule is quite simple, actually, but let's step through it line-by-line.

Line 1 states we are declaring a rule and gives it the name "Landing Page".

Line 2 says to only run the rule once to prevent an infinite loop. In this case, our `when` statement looks for the query term "Belkin" on the field "manu"; after the modifications from the rule, the query will still match the rule, which could make it fire again. Using `no-loop` prevents the rule firing over and over.

Line 3 starts the `when` conditions.

Line 4 uses Solr's ResponseBuilder to analyze the query, and match when the query (in the `q` parameters of the request sent to Solr) matches "manu:Belkin". Note this line is also setting a variable `$qStr`, and assigning it the query and parameters. This variable will be used again later.

Line 5 starts the `then` actions.

Line 6 defines a key/value pair to the NamedList. In this case, inserting "landing page" and the URL into the responseHeader.

Line 7 ends the rule.

Note that this rule by itself does not magically redirect the user to the Belkin website - it includes the information to the client, which then must decide what to do: redirect the user, make it the first result in the list, or some other transformation as needed.

To run this rule, once the rule has been added to `rules/defaultLanding.drl`, you can use the 'search-with-rules' requestHandler that was added to your `solrconfig.xml` file during installation. Then you can send a request to Solr that looks something like this:

```
http://localhost:8983/solr/collection1/search-with-rules?q=manu:Belkin&rules=true&landing=
```

The request should be customized for your hostname and port, and this example also assumes you have indexed Solr's sample documents in the `example/exampledocs` directory.