Entity Recognition Module

1.0 Documentation

Created: 27-Aug-2013

Table of Contents

Entity Recognition Module

The Entity Recognition module provides entity recognition and entity extraction from text documents.

It can extract these entities by default, with minimal configuration:

- Person
- Location
- Organization
- Time
- Date
- Currency
- Percent

This documentation covers:

- How to install the Entity Recognition module and use it with your existing Solr implementation
- How to configure entity extraction for your schema
- Details on the extractors used with the module and how to tweak them for your needs
- Information on indexing documents and extracting entities

The module includes many options for configuration, including named lists of entities, the fields entities are extracted from, and how the entities are stored in the index. These configuration options can be modified for each Solr collection, or applied globally.

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.

Entity Recognition Module Installation

Installing the Entity Recognition module is straightforward. This section also describes how to start and stop the module, as well as a few approaches to integration with LucidWorks Search if desired.

System Requirements

The Entity Recognition module has the following requirements:

- Solr version 4.0 or higher. The module is not required to be located on the same server as Solr, but wherever it is installed, it must be able to communicate with Solr to send documents that have been analyzed.
- Wherever the module is located, the server should have at least 4Gb of memory, more if possible.

Installation

Installation of the Entity Recognition Module is completed in a few steps.

1. Download the package `lwx-named-entity-module.tar.gz` package from the LucidWorks Marketplace and move it to the server and filesystem location of your choice. You could use a command like:
   `tar xzvf lwx-named-entity-module.tar.gz -C </path/>`

   This will extract the packages files to the location specific with `</path>`. Note also that the package name may include a version number reference so the package you have may not match exactly the package name shown above.
2. Once unpacked, navigate to the path defined. You should have a directory named `lwx-named-entity-module` (again, this may contain a version number reference in the directory name). In that directory, you should also have the following sub-directories:

```
conf
excludes
gazeteer
legal
lib
models
```

Once installed, see the Entity Recognition Quick Start section for for a simple example on how to use it. Or, jump straight into Entity Recognition Module Configuration.

Start and Stop

Before starting the Entity Recognition module, you should verify that the host and port definitions in `$NER_HOME/conf/lwx-indexing-module.properties` are correct.

To start the module, issue this command:

```
java -Xmx2g -cp .:lib/*:conf/:models/:gazetteer/:excludes/ com.lucid.lwx.SpringMain
camelContext.xml
```

Integration with LucidWorks Search

It is possible to integrate the Entity Module with LucidWorks Search, which will allow you to use the crawlers with LucidWorks Search as a source of content to be analyzed. Be warned, however, that this approach has not yet been fully tested, and may not work with all LucidWorks Search configurations. A future release of the module will include a more complete integration.

The integration works by sending the output of each connector through the Entity Recognition module before indexing. There are three parts to achieving this.

Step 1: Move `.jar` files into place. There are five `.jar` files in the `lib` directory of the module that should be moved to LucidWorks Search, specifically to `$LWS_HOME/app/webapps/connectors/lib`. These files are:

- avro-1.7.3.jar
- avro-ipc-1.7.3.jar
- avro-update-controller-0.1.jar
- lwx-shared-0.1.jar
- netty-3.6.5.Final.jar

If LucidWorks Search is running when you move these files, the system should be restarted.

Step 2: Create data sources with special output options. Connectors in LucidWorks Search send their output to Solr by default, but there are two options to change the output.

- Output Type (or, `output_type` with the API): This should be set to `com.lucid.lwx.AvroUpdateController`.
- Output Arguments (or, `output_args` with the API): This should be set to the Avro host and TCP port defined in `lwx-indexing-module.properties`, in the format `avro.host:avro.tcp.port`. So, using the default `avro.host` of "localhost" and the default `avro.tcp.port` of "8998", you would enter "localhost:8998".

Step 3: Ensure `lwx-named-entity-module.xml` is updated with correct field names. Check and makes sure that the `targetField` entries in `lwx-named-entity-module.xml` match defined fields or dynamic field rules.

When large, entity-rich files are being processed, the consumption of incoming documents should be throttled. This is done by adding two properties to the incoming endpoint in `conf/camlelContext.xml`, specifically the size of the queue when new documents should start blocked, and enabling `blockWhenFull` parameter (by setting it to true). The default incoming endpoint does not include these parameters; to add them, you would make the line look like this:

```
<endpoint id="incoming" uri="seda:incoming?size=1000&amp;blockWhenFull=true"/>
```

Entity Recognition Quick Start

If you have not already installed the Entity Recognition module, please see the section Entity Recognition Module Installation.

To get started with entity recognition, let's follow an example. We'll just extract entities from one simple document, shown below.

First, we need to modify a configuration file to tell the module where to find Solr. Navigate to `$NER_HOME/conf` and open the file `lwx-indexing-module.properties`. In the section marked `#SOLR`, find the following two lines:

```
# Solr
solr.host = localhost
solr.port = 8888
```

Change the `solr.host` value to the hostname of where Solr is running (if you are using LucidWorks Search, this would be the same host where LucidWorks Search is running). Then change `solr.port` to equal the port used to connect to Solr. By default in LucidWorks Search this is '8888'; in a default Solr installation this is '8983'. If you have modified the port, enter the correct value. Save the file when finished.

There are other parameters in `lwx-indexing-module.properties`, we'll learn more about them in the section MODULES:Named Entity Recognition Configuration.

Next, start the module with this command:

```
java -Xmx2g -cp .:lib/*:conf/:models/:gazetteer/:excludes/ com.lucid.lwx.SpringMain
camelContext.xml
```

Once started, we can submit a simple document. We can do this with an API request, formatting the document in JSON. For now, submit this document from the server where the Entity Recognition module is running; in Entity Recognition Module Configuration you'll learn how to change the properties to allow submitting documents from another server.

```
curl -H "Content-Type: application/json" -d '{"id": "document-1","metadata": \
{"collection":"collection1","needsCommit":"true"},"fields":{"title":"LucidWorks Entity
Recognition document","body_t": \
"This is a simple document to get started with LucidWorks Entity Recognition for
Solr."}}' \
 http://localhost:8000/documents
```

Once this document loads, you should find when you query the index that two fields have been added to the document: "organization_ss" and "time_ss". These are the entities that have been extracted from the document. Here is an example of the output of Solr's Query screen:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "*:*",
      "_": "1375730287991",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": "document-1",
        "title": [
          "LucidWorks Entity Recognition document"
        ],
        "body_t": "This is a simple document for getting started with LucidWorks Entity
Recognition for Solr.",
        "organization_ss": [
          "LucidWorks",
          "Search ."
        ],
        "time_ss": [
          "This"
        ],
        "_version_": 1442557263964274700,
        "timestamp": "2013-08-05T19:10:12.587Z"
      }
    ]
  }
}
```

For more detailed information about using this module for customized entity extraction, see the section Entity Recognition Module Configuration.

Entity Recognition Module Configuration

There are several configuration options for the Entity Recognition module. We can split these options between two types: module configuration, defining the ports and hostnames of the systems the module will interact with, and extraction configuration, where we define the various models and rules the extraction process should use.

Each of the configuration files is described below. However, the two main files that you will need to work with most often are `lwx-indexing-module.properties` and `lwx-named-entity-module.xml`.

Module Configuration

Indexing Properties

The primary configuration file for module configuration is `lwx-indexing-module.properties`.

This is where hosts and ports are defined for several components. They are described below in the order they appear in the properties file.

Avro IPC

The Entity Recognition module uses Apache Avro for data serialization tasks. Specifically, it uses the inter-process calls (IPC) to facilitate communication between external connections and Solr.

The available settings are:

- **avro.host**: The host address that Avro will listen on.
- **avro.tcp.port**: The port that Avro Netty will listen on.
- **avro.http.port**: The HTTP port that Avro will listen on.

JSON Endpoint

The JSON endpoint allows sending documents directly to the module for entity extraction. Documents are then sent to Solr for indexing. There are two settings:

- **json.host**: The host that the JSON endpoint will listen on.
- **json.port**: The port that the JSON endpoint will listen on

Pipeline

The pipeline is used with Camel routes. There is one available configuration parameter, `pipeline.threads`, which defines how many threads the routes can use during processing.

Solr

The Entity Recognition module uses Solr for document indexing. There are four available settings:

- **solr.host**: The host that Solr is listening on. If you are using LucidWorks Search, this would be the same host that was defined for the LWE-Core component.
- **solr.port**: The port that Solr is listening on. Again, if you are using LucidWorks Search, this is the same port that was defined for the LWE-Core component.
- **solr.batch.size**: The number of documents to accumulate before sending the documents to Solr. The default is 10.
- **solr.batch.timeout**: The number of milliseconds to wait for documents before sending documents to Solr. If the `solr.batch.size` is not reached, the module will wait this amount of time before sending the smaller batch to Solr for indexing.

Camel Context

The file `camelContext.xml` provides Apache Camel with the route configurations that will be loaded when the system is started. This has been pre-defined with the needed contexts, and it's not necessary to modify this file when just starting to use the system.

However, when large, entity-rich files are being processed, the consumption of incoming documents should be throttled to avoid running out of memory. This is done by adding two properties to the incoming endpoint in `conf/camlelContext.xml`, specifically the size of the queue when new documents should start blocked, and enabling `blockWhenFull` parameter (by setting it to true). The default incoming endpoint does not include these parameters; to add them, you would make the line look like this:

```
<endpoint id="incoming" uri="seda:incoming?size=1000&amp;blockWhenFull=true"/>
```

Other Configurations

The file `lwx-named-entity.properties` is not yet used for any purpose, but is a placeholder for future options.

Entries in `lwx-config.xml` should not be modified, as they are the main definitions for the module.

Extraction Configuration

There are two files that impact how entities are extracted (or, in some cases, recognized) and indexed.

Entity Recognition and Extraction

The file `lwx-named-entity-module.xml` is the primary file that will define the extractors. There are several sections to this file.

Extractors

This is in two sections of the file, "Extractor Factories" and "Extractors". These sections define three extractors included with the Entity Recognition module, called OpenNLP, Lookup and RegEx. These extractors use trained models, rules, and regular expressions to extract the entity types defined from documents.

You are able to define different extractor rules for different entity types. This is particularly useful with the Lookup Extractor, where item lists are used to define known entities:

```
<entry key="location">
          <list>
            <value>classpath:gazetteer/city*.lst</value>
            <value>classpath:gazetteer/country*.lst</value>
            <value>classpath:gazetteer/loc*.lst</value>
            <value>classpath:gazetteer/province*.lst</value>
            <value>classpath:gazetteer/region*.lst</value>
          </list>
        </entry>
```

Because the extractors have a number of options, please see the section Extractors for more information on how to configure the three types of extractors.

Entity Profiles

The profiles allow you to define extractors for specific types of entities. For example, you could use OpenNLP and RegEx extractors for one type of entity, and only the Lookup extractor for another type. Here is an example of defining the organization entity type:

```
<bean id="orgProfile" class="com.lucid.lwx.modules.ner2.EntityProfile"
scope="singleton">
    <property name="entityType" value="organization"/>
    <property name="extractors">
      <list>
        <ref bean="opennlpExtractor"/>
        <ref bean="lookupExtractor"/>
      </list>
    </property>
  </bean>
```

For best results, consider using multiple extractors for each entity profile as often as possible. This will provide the maximum entity recognition and extraction.

Field Mapping

It's possible to specify how incoming fields will be analyzed for entities, and to what field the extracted entities should be saved. The mapping is done per entity profile, defined earlier. A source field can be analyzed for more than one target field, and also for more than one entity type.

The mapping is done by defining a `sourceField`, which is field in the source document, a `targetField`, which is the target field in Solr. The `targetField` must have been previously defined in Solr as either a field or a dynamic field rule. Finally, a profile to use for that mapping is defined. If we take a look at this example:

```
<bean class="com.lucid.lwx.modules.ner2.FieldConfiguration">
          <property name="sourceField" value="title" />
          <property name="targetField" value="person_ss" />
          <property name="collection" value="collection1" />
          <property name="profile" ref="personProfile" />
        </bean>
```

The `sourceField` is defined as "title" (the title of the document) and the `targetField` is defined as "person_ss". In Solr's `schema.xml` file, we have a dynamic field rule "*_ss". The field type and analysis will be applied to the incoming content before indexing, same as any other incoming content. If the `targetField` does not exist, or if the field defined doesn't match a dynamic field rule, Solr will not be able to index the entities.

The `collection` value allows you to apply a configuration to a specific collection instead of to all collections.

Finally, the `profile` references an entity profile to use.

Indexing

The file `lwx-indexing-module.xml` defines Camel routes and processors for document acquisition and Solr indexing. It's not necessary to change any of the configurations defined there.

Related Topics

- Extractors
- Solr Fields and Dynamic Fields from the Apache Solr Reference Guide
- LucidWorks Search documentation on Fields and Dynamic Fields

Extractors

The Named Entity Recognition module includes three types of extractors, described below. These extractors can work together, or they can be defined separately.

- OpenNLP Extractor
- Lookup Extractor
- RegEx Extractor

OpenNLP Extractor

The OpenNLP Extractor uses a pre-trained to extract entities from documents. The model was trained with Apache OpenNLP tools. It can currently recognize the following entity types: person, location, organization, money, time, date, and percentage.

Note that the pre-trained model was created with English language news articles. If you are working with content in another language, or need models better suited for other styles of content, you may want to consider additional models available through OpenNLP (available through SourceForge, or training your own model. For more information about training your own models, see the OpenNLP documentation at Name Finder Training. Once a new model has been trained, it should be located in `$NER_HOME/models`, and defined in `lwx-named-entity-module.xml`.

Lookup Extractor

The Lookup Extractor uses predefined lists of items that will be extracted from documents. You can find the pre-defined lists in `$NER_HOME/gazetteer`. You can edit them, or add new lists as needed.

Once defined, the lists can be combined to make an aggregated list for a specific entity type. The aggregated list is configured in the `lwx-named-entity-module.xml` file, in the section "Extractor factories". First, the bean needs to be defined, with the class, followed by the entity maps. This is already in the `lwx-named-entity-module.xml` file:

```
<bean id="lookupExtractorFactory"
class="com.lucid.lwx.modules.ner2.impl.LookupExtractor$Factory" scope="singleton">
    <property name="extendedChars" value="€£"/>
    <property name="entityTypes">
      <map>
...
      </map>
    </property>
  </bean>
```

Between the `<map>` tags, the entity types are defined. For example, here is an aggregated list for a location entity type.

```
<entry key="location">
          <list>
            <value>classpath:gazetteer/city*.lst</value>
            <value>classpath:gazetteer/country*.lst</value>
            <value>classpath:gazetteer/loc*.lst</value>
            <value>classpath:gazetteer/province*.lst</value>
            <value>classpath:gazetteer/region*.lst</value>
          </list>
        </entry>
```

Other types are also defined by default. You can add to an existing type, or add your own type. You could make a list of rivers, for example, and add the list to the location entity type. Or you could define a new type "rivers" and only use your list as the value.

RegEx Extractor

The RegEx Extractor allows you to use a regular expression to match data in a document. This can be done with a few pre-defined regular expression libraries, or by defining your own. Below there are three pre-defined types to extract dates, currencies, and percentages.

```
<bean id="regexExtractor" class="com.lucid.lwx.modules.ner2.impl.RegexEntityExtractor">
    <property name="entityTypes">
      <map>
        <entry key="date"
value="#{T(com.lucid.lwx.modules.ner2.impl.RegexEntityExtractor).DATE_TIME_ISO8601}" />
        <entry key="money"
value="#{T(com.lucid.lwx.modules.ner2.impl.RegexEntityExtractor).US_CURRENCY}" />
        <entry key="percentage"
value="#{T(com.lucid.lwx.modules.ner2.impl.RegexEntityExtractor).PERCENTAGE}" />
      </map>
    </property>
  </bean>
```

To define your own, you could add a new entry like this, where the `<constructor-arg>` is the pattern to match:

```
<entry key="a_words">
           <bean class="java.util.regex.Pattern" factory-method="compile">
              <constructor-arg value="\b[Aa]\w*\b"/>
           </bean>
        </entry>
```

Indexing Documents

The Entity Recognition module is able to accept documents for analysis, and then pass the documents to Solr for addition to the index. With Solr, one usually sends the documents to Solr via an update processor. However, when you want to extract entities from those documents and apply them to the documents before indexing, you first send the documents through the Entity Recognition module.

There are two ways to send documents through the module: with the HTTP endpoint, or with the Apache Avro IPC endpoint.

HTTP Endpoint

This is possible due to an HTTP endpoint, which is defined in `lwx-indexing-module.properties` (as described in Entity Recognition Module Configuration) and a document schema used throughout the system.

Here is an example of a minimal representation of a document:

```
{"id": "document-1",
  "metadata": {
    "collection": "collection1",
    "needsCommit": "true"},
  "fields": {
    "foo": "bar",
    "x": 1}
}
```

The schema allows the following elements to be defined for each document:

Element	**Description**
id	The document ID. If the document ID is not unique to the index, Solr will overwrite an existing document with the incoming document
metadata	Defines elements that describe the document itself.
collection	The Solr collection the document will be a part of. This element is contained within the `metadata` element.
needsCommit	Whether Solr should execute a commit, which will save the document to the index and make it searchable. This is recommended to be set to **true**.
fields	A JSON map of field names and their attributes

⚠ With the HTTP endpoint, only one document can be sent per call, and it must be formatted in JSON.

Avro IPC Endpoint

The Entity Recognition module can use Apache Avro for data serialization tasks. Specifically, it uses the inter-process calls (IPC) and Netty to speed document transfers during indexing. As of this release, it is primarily used with LucidWorks Search connectors (see also the section Integration with LucidWorks Search, but it could be used with a bit of custom Java code to send documents to Solr.

As an example, here is a simple class that connects to the Avro endpoint and send a simple document.

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.util.HashMap;
import java.util.Map;

import org.apache.avro.ipc.specific.SpecificRequestor;
import org.apache.avro.ipc.NettyTransceiver;

import com.lucid.lwx.Constants;
import com.lucid.lwx.Document;
import com.lucid.lwx.rpc.SendDocument;

public class AvroExample {
  public static void main(String[] args) throws Exception {
    // Connect to the remote Avro receiver
    String host = "localhost";
    int port = 8998;
    NettyTransceiver client = new NettyTransceiver(new InetSocketAddress(host, port));

    // Create the local IPC proxy object
    SendDocument proxy  = (SendDocument)
SpecificRequestor.getClient(SendDocument.class, client);

    // Build a Document
    Document.Builder builder = Document.newBuilder();
    builder.setId("doc-1");
    Map<String, Object> fields = new HashMap<String, Object>();
    fields.put("title", "LucidWorks Entity Recognition document");
    fields.put("body_t", "This is a simple document for getting started with LucidWorks
Entity Recognition for Solr.");
    builder.setFields(fields);
    Map<String, Object> metadata = new HashMap<String, Object>();
    metadata.put(Constants.COLLECTION, "collection1");
    metadata.put(Constants.NEEDS_COMMIT, true);
    builder.setMetadata(metadata);
    Document doc = builder.build();

    // Send the Document
    proxy.send(doc);

    // Clean up
    client.close();
  }
}
```

Note that this class is named 'AvroExample'; if you modify the name, remember to also modify the compile and run examples below.

If you'd like to use this class as a base for your own custom code, keep in mind the libraries packaged with the Entity Recognition module are required to compile the code and also to run the client. To compile the code, you need these five dependencies, all found in the `lib` directory for the module:

- avro-1.7.3.jar
- avro-ipc-1.7.3.jar
- avro-update-controller-0.1.jar
- lwx-shared-0.1.jar
- netty-3.6.5.Final.jar

For example, you could compile the custom class with this command (which assumes your custom class is named 'AvroExample.java'):

```
javac -cp
.:lib/avro-1.7.3.jar:lib/avro-ipc-1.7.3.jar:lib/avro-update-controller-0.1.jar:lib/lwx-sha
AvroExample.java
```

To run the code, you need the same .jars used to compile, plus a few more:

- avro-1.7.3.jar
- avro-ipc-1.7.3.jar
- avro-update-controller-0.1.jar
- lwx-shared-0.1.jar
- netty-3.6.5.Final.jar
- slf4j-api-1.6.6.jar
- slf4j-log4j12-1.6.6.jar
- log4j-1.2.17.jar
- jackson-core-asl-1.9.12.jar
- jackson-mapper-asl-1.9.12.jar

For example, you could run the custom class with this command (again assuming your custom class is named 'AvroExample') :

```
java -cp
.:lib/avro-1.7.3.jar:lib/avro-ipc-1.7.3.jar:lib/avro-update-controller-0.1.jar:lib/lwx-sha
AvroExample
```

Alternately, you could also simply refer to the `lib` directory of the Entity Recognition module without having to name the specific .jars, like this (to compile):

```
javac -cp .:lib/* AvroExample.java
```