lucid
IMAGINATION

Thinking Lucene Think Lucid.

Indexing Text and HTML Files with Solr, the Lucene Search Server

A Lucid Imagination
Technical Tutorial

By Avi Rappoport
Search Tools Consulting

Abstract

Apache Solr is the popular, blazing fast open source enterprise search platform; it uses Lucene as its core search engine. Solr's major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, and complex queries. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites. Lucid Imagination's LucidWorks Certified Distribution for Solr provides a fully open distribution of Apache Solr, with key complements including a full Reference Guide, an installer, and additional functions and utilities. All the core code and many new features are available, for free, at the Lucid Imagination web site (www.lucidimagination.com/downloads).

In the past, examples available for learning Solr were for strictly-formatted XML and database records. This new tutorial provides clear, step-by-step instructions for a more common use case: how to index local text files, local HTML files, and remote HTML files. It is intended for those who have already worked through the Solr Tutorial or equivalent.

Familiarity with HTML and a terminal command line are all that is required; no formal experience with Java or other programming languages is needed. System Requirements for this tutorial are those of the Startup Tutorial: UNIX, Cygwin (Unix on Windows), Mac OS X; Java 1.5, disk space, permission to run applications, access to content.

Contents

Introduction

Apache Solr is the popular, blazing fast open source enterprise search platform; it uses Lucene as its core search engine. Solr's major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, and complex queries. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites[1].

Today, the newly released version of Solr 1.4 includes a new module called Solr Cell that can access many file formats including plain text, HTML, zip, OpenDocument, and Microsoft Office formats (both old and new). Solr Cell is invokes the Apache Tika extraction toolkit, another part of the Apache Lucene family, integrated in Solr). This tutorial provides a simple introduction to this powerful file access functionality.

In this tutorial, we'll walk you through the steps required for indexing readily accessible sources with simple command-line tools for Solr, using content you are likely to have access to: your own files, local discs, intranets, file servers, and web sites.

Part 1: Installing This Tutorial

As it turns out, the existing examples for in the default installation of the Solr Tutorial are for indexing specific formats of XML and JDBC-interface databases. While those formats can be easier for search engines to parse, many people learning Solr do not have access to such content. This new tutorial provides clear, step-by-step instructions for a more common use case: how to index local text files, local HTML files, and remote HTML files. It is intended for those who have already worked through the Solr Tutorial or equivalent.

This tutorial will add more example entries, using Abraham Lincoln's *Gettysburg Address* and the United Nations' *Universal Declaration of Human Rights* as text files, and as HTML files, and walk you through getting these document types indexed and searchable.

First, follow the instructions in the Solr Tutorial (from http://lucidimagination.com/Downloads/LucidWorks-for-Solr or http://lucene.apache.org/solr/tutorial.html) from installation to *Querying Data* (or

[1] Lucene, is the Apache search library at the core of Solr, presents the interfaces through a collection of directly callable Java libraries, offering fine-grained control of machine functions and independence from higher-level protocols, and requiring development of a full java application. Most users building Lucene-based search applications will find they can do so more quickly if they work with Solr, as it adds many of the capabilities needed to turn a core search function into a full-fledged search application.

beyond). When you are done, the Solr index file will have about 22 example entries, most of them about technology gadgets.

Next, use a browser or ftp program to access the tutorial directory on the Lucid Imagination web site (http://www.lucidimagination.com/download/thtutorial/). You should find the following files:

```
lu-example-1.txt    lu-example-4.txt    lu-example-8.html   thtutorial.zip
lu-example-1.xml    lu-example-5.txt    post-txt.sh
lu-example-2.txt    lu-example-6.html   remote
lu-example-3.txt    lu-example-7.html   schema.xml
```

For your convenience, all of the files above are included in thtutorial.zip (http://www.lucidimagination.com/download/thtutorial/thtutorial.zip). Move the zip file to the Solr *example* directory, (which is probably in *usr/apache-solr-1.4.0* or */LucidWorks)*, and unzip it: this will create an *example-text-html* directory

Working Directory: example-text-html

This tutorial assumes that the working directory is `[Solr home]/examples/examples-text-html`: you can check your location by using the Unix command line utility *pwd*.

Setting the schema

Before starting, it's important to update the example *schema* file to work properly with text and HTML files. The schema needs one extra field defined, so all words in the plain text files, and HTML body words go into the default field for searching.

Make a backup by renaming the conf directory file from *schema.xml* to *schema-bak.xml*

```
% mv ../../lucidworks/solr/conf/schema.xml ../../lucidworks/solr/conf/schema-
bak.xml
```

Then either copy the `text-html` version of the schema or edit the version that's there to include the body text field.

- **Either***:* copy the new one from the `example-text-html` directory into the conf directory:

  ```
  % cp schema.xml ../../lucidworks/solr/conf/schema.xml
  ```

  or (for apache installs)

  ```
  % cp schema.xml ../solr/conf/schema.xml
  ```

- **Or:** edit the schema to add this field:
  - Open the original `schema.xml` in your favorite text editor

- Go to line 469 (LucidWorks) or 450 (apache). This should be the Solr Cell section with other HTML tags.

```
...
<field name="last_modified" type="date" indexed="true" stored="true"/>
<field name="links" type="string" indexed="true" stored="true"
multiValued="true"/>
```

- and add the code to create the body field:

```
<field name="body" type="text" indexed="true" stored="true" multiValued="true"/>
```

- Go to line 558 (LucidWorks) or 540 (Apache), and look for the *copyfield* section.

```
...
<copyField source="includes" dest="text"/>
<copyField source="manu" dest="manu_exact"/>
```

- Go to the end of the section, after field *manu* and add the line to copy the body field content into the text field (default search field).

```
<copyField source="body" dest="text"/>
```

- Save and close the `schema.xml` file.

Restarting Solr

Solr will not use the new schema until you restart the search engine. If you haven't done this before, follow these steps:

- Switch to the terminal window in which the Solr engine has been started
- Press ^c (control-c) to end this session: it should show you that `Shutdown` hook is executing.
- (Apache) Type the command `java -jar start.jar` to start it again. This only works from the `example` directory, not from the `example-text-html` directory.
- (LucidWorks) Start Solr by running the start script, or clicking on the system tray icon

Part 2: Solr Indexing with *cURL*

Plain text seems as though it should be the simplest, but there are a few steps to go through. This tutorial will walk through the steps, using the Unix shell **cURL** command.

Using the cURL command to index Solr XML

The first step is communicating with Solr. The Solr Startup Tutorial shows how to use the Java tool to index all the .xml files. This tutorial uses the **cURL** utility available in Unix, within the command-line (terminal) shell.

- Telling Solr to index is like sending a POST request from an HTML form, with appropriate path name (by default `/update`) and parameters. cURL uses this process, on the command-line. This example uses the test file `lu-example-1.xml.`

To start, be in the `solr/example/example-text-html` directory.

Then, instruct Solr to `update` (index) an XML file using cURL , and then finish the index update with a `commit` command

```
 cURL 'http://localhost:8983/solr/update/' -H 'Content-type:text/xml' --data-
binary "@lu-example-1.xml"

 cURL 'http://localhost:8983/solr/update/' -H "Content-Type: text/xml" --data-
binary '<commit/>'
```

Successful calls have a `response status` of `0`.

```
<xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader">
<int name="status">0</int><int name="QTime">000</int></lst>
</response>
```

Troubleshooting errors with cURL Solr updates

If you have a cURL error, it's usually mis-matched double quotes or single quotes. If you see one of the following, go back and try again.

```
 cURL: (26) failed creating formpost data
 cURL: (3) <url> malformed
Warning: Illegally formatted input field!
 cURL: option -F: is badly used here
```

Common errors numbers from the Solr server itself include 400 and 500. This means that the POST was properly formatted but included parameters that Solr could not identify. When that happens, go back to a previous command that did work, and start building the new one up from there. These errors should not damage your Solr search engine or index.

```
<title>Error 400 </title>
</head>
<body><h2>HTTP ERROR: 400</h2><pre>Unexpected character 's' (code 115) in
prolog; expected '&lt;'
 at [row,col {unknown-source}]: [1,1]</pre>
```

or

```
<title>Error 500 </title>
</head>
<body><h2>HTTP ERROR: 500</h2>
<pre>org.apache.lucene.store.NoSuchDirectoryException: directory
'/Applications/LucidWorks/example/solr/data/index' does not exist
```

If you can't make this work, you may want to follow the instructions with the Solr Startup Tutorial to create a new Solr directory and confirm using the Java indexing instructions for the `exampledocs` XML files before continuing.

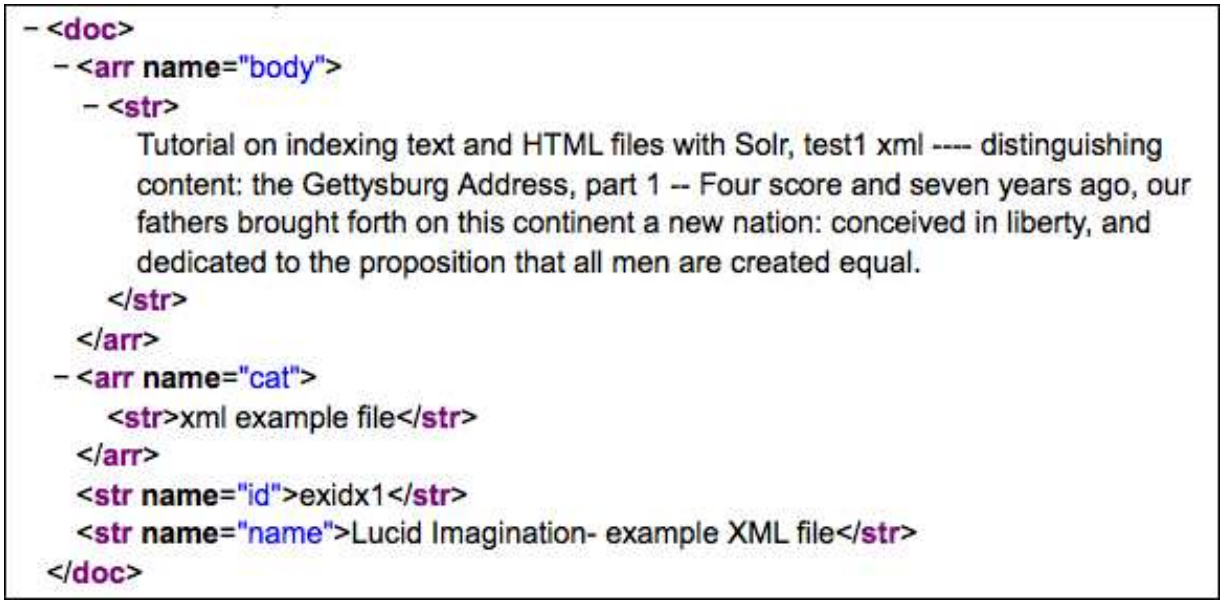
Viewing the first text file in Solr

Once you have successfully sent the XML file to Solr's update processor, go to your browser, as in the Getting Started tutorial, and search your Solr index for *"gettysburg"* http://localhost:8983/solr/select?q=gettysburg.

The result should be an XML document, which will report one item matching the new test file (rather than the earlier example electronic devices files). The number of matches is on about the eighth line, and looks like this:

```
<result name="response" numFound="1" start="0">
```

After that, the Solr raw interface will show the contents of the indexed file:



Notes

You must use a browser than can render XML, such as Firefox or Internet Explorer or Opera (but not Safari).

The field label `arr` indicates a multiValued field.

Part 3: Using Solr to Index Plain Text Files

Integrated with Solr version 1.4, Solr Cell (also known as the ExtractingRequestHandler) provides access to a wide range of file formats using the integrated Apache Tika toolkit, including untagged plain text files. The test file for this tutorial is `lu-example-2.txt`. It has no tags or metadata within it, just words and line breaks.

Note

The Apache Tika project reports that extracting the words from plain text files is surprisingly complex, because there is so little information on the language and alphabet used. The text could be in Roman (Western European), Indic, Chinese, or any other character set. Knowing this is important for indexing, in particular for defining the rules of word breaks, which is *Tokenization.*

Invoking Solr Cell

To trigger the Solr Cell text file processing (as opposed to the Solr XML processing), add `extract` in the URL path in the POST command: `/solr/update/extract`.

This example includes three new things: the `extract` path term, a **document ID** (because this file doesn't have an ID tag), and an inline `commit` parameter, to send the update to the index.

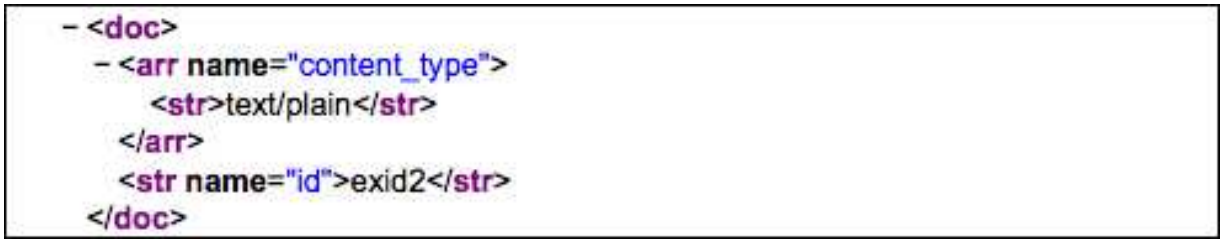
```
 cURL 'http://localhost:8983/solr/update/extract?literal.id=exid2&commit=true' -
F "myfile=@lu-example-2.txt"
```

The `response` status of `0` signals success. Your cURL command has added the contents of *lu-example-2.txt* to the index.

When running the query http://localhost:8983/solr/select?q=gettysburg in the index, both documents are matched.

```
<result name="response" numFound="2" start="0">
```

Unlike the indexed XML document, with this text document, there are only two fields (`content-type` and `id`) that are visible in the search result. The text content, even the word *"Gettysburg,"* all seems to be missing.



How can Solr match words in a file using text that doesn't seem to be there? It's because Solr's default `schema.xml` is set to *index for searching*, but not *store for viewing*. In other words, Solr isn't preset to store for your viewing the parts of the documents with no HTML tags or other labels. For plain text files, that's everything, so the next section is about changing that behavior.

Parameters for more fields

Solr Cell provides ways to control the indexing without having to change source code. Parameters in the POST message set the option to save information about the documents in appropriate fields, and then to grab the text itself and save it in a field. The metadata can be extracted without the file contents or with the contents.

Solr Cell external metadata

When Solr Cell reads a document for indexing, it has some information about the file, such as the name and size. This is metadata (information about information), and can be very valuable for search and results pages. Although these fields are not in the *schema.xml* file, Solr is very flexible, and can put them in *dynamic* fields that can be searched and displayed.

The operative parameter is `uprefix=attr_`; when added to the POST command, it will save the file name, file size (in bytes), content type (usually text/plain), and sometimes the content encoding and language.

```
 cURL
'http://localhost:8983/solr/update/extract?literal.id=exid2&uprefix=attr_&commit
=true' -F "myfile=@lu-example-2.txt"
```

Here is an example of the same file, indexed with the `uprefix=attr_` parameter:

Mapping document content

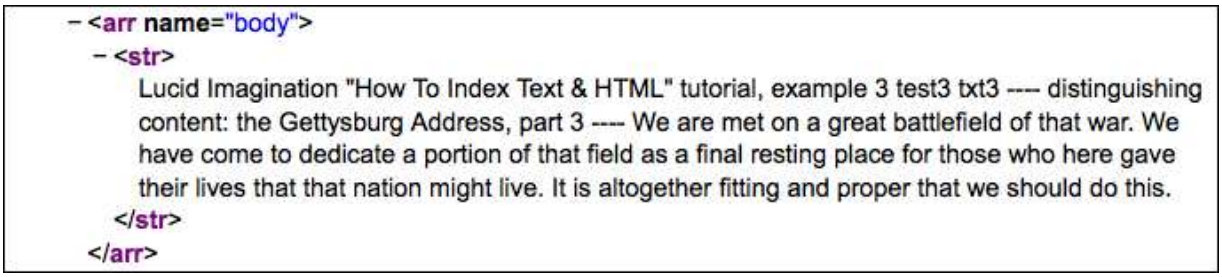
Once the metadata is extracted, Solr Cell can be configured to grab the text at well. The `fmap.content=body` parameter stores the file content in the `body` field, where it can be searched and displayed.

> ***Note***
>
> Using the `fmap` parameter without `uprefix` will not work. To see the body text, the `schema.xml` must have a `body` field, as described in the Install section above.

Here's an example of an index command with both attribute and content mapping:

```
 cURL
'http://localhost:8983/solr/update/extract?literal.id=exid3&uprefix=attr_&fmap.c
ontent=body&commit=true' -F "txtfile=@lu-example-3.txt"
```

Searching the Solr index <http://localhost:8983/solr/select?q=gettysburg> will now display the all three example files. For `lu-example-3.txt`, it shows the body text in the `body` field and metadata in various fields.



Part 4: Indexing All Text Files in a Directory

The Solr Startup Tutorial `exampledoc` directory contains a *post.sh* file, which is a shell script that uses cURL to send files to the default Solr installation for indexing. This version uses the cURL commands above to send .txt (as opposed to .xml) files to Solr for indexing. The file `post-text.sh` should be in the `../example/example-text-html/` directory with the test files.

Shell script for indexing all text files

- Set the permissions: `chmod +x post-text.sh`
- Invoke the script: `./post-text.sh`

You should see the `<response>` with status `0` and the other lines after each item: if you do not, check each line for exact punctuation and try again.

When you go back to search on Solr, http://localhost:8983/solr/select?q=gettysburg, you will find five text documents and one XML document.

Different doc IDs: adds aan additional document

Note that the results include two different copies of the first example, both containing "*Four score and seven years ago*", because the script loop sent all text files with the generated **`exid`** number, while the XML example contains an id starting with **`exidx`**.

Identical doc IDs - replaces a document

The second example text file had some text that was indexed but not stored as a text block when we first indexed it. Now it has content in the `body` field, because the script loop sent it with the same ID (and the new parameters), so Solr updated the copy that was already in the index, using the Doc ID as the definitive identifier.

For more information on IDs, see the *LucidWorks Certified Distribution Reference Guide* on Unique Key.

More robust methods of indexing files

Sending indexing and other commands to Solr via cURL is an easy way to try new things and share ideas, but cURL is not built to be a production-grade facility. And because Solr's HTTP API is so straightforward, there are many ways to call Solr programmatically. There are libraries for Solr in nearly every language, including Java, Ruby, PHP, JSON, C#, and Perl, among others. Many content management sytems (CMS), publishing and social media systems have Solr modules, such as Ruby on Rails, Django, Plone, TYPO3, and Drupal; it is also used in cloud computing environments such as Amazon Web Services and Google

Code. For more information, check the Solr wiki and the LucidWorks Solr client API Lineup in the *LucidWorks Certified Distribution Reference Guide*.

Part 5: Indexing HTML Files

This tutorial uses the same cURL commands and shell scripts for HTML as for text. Solr Cell and Tika already extract many HTML tags such as `title` and `date modified`.

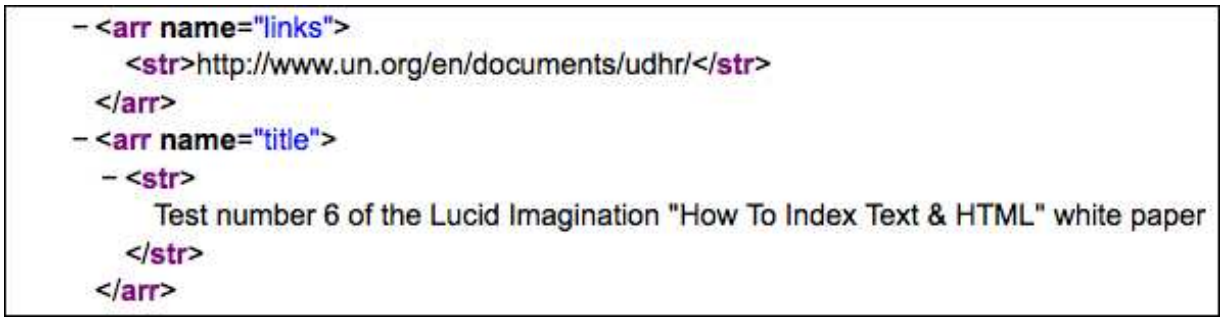
> **Note**
> All the work described above on text files also applies to HTML files, so if you've skipped to here, please go back and read the first sections.

cURLSimplest HTML indexing

The first example will index an HTML file with a quote from the Universal Declaration of Human Rights:

```
cURL 'http://localhost:8983/solr/update/extract?literal.id=exid6&commit=true' -F
"myfile=@lu-example-6.html"
```

Doing a query for "*universal*", http://localhost:8983/solr/select?q=universal , shows us that Solr Cell created the metadata fields `title`, and `links`, because they are standard HTML constructs.



Again, by default, the body text is indexed but not stored; and, changing that is just as easy as changing it with the text files.

Storing more metadata from HTML

As in the text file section of this tutorial, this example uses the `uprefix` parameter `attr_` to mark those fields that Solr Cell automatically creates but which are not in the `schema.xml`. This is not a standard, but it's a convention that's widely used.

```
cURL
'http://localhost:8983/solr/update/extract?literal.id=exid7&uprefix=attr_&commit
=true' -F "myfile=@lu-example-7.html"
```

Searching for "universal" now finds both HTML documents. While `exid6` has very little stored data, `exid7` has the internal metadata of the document, including the title, author, and comments.

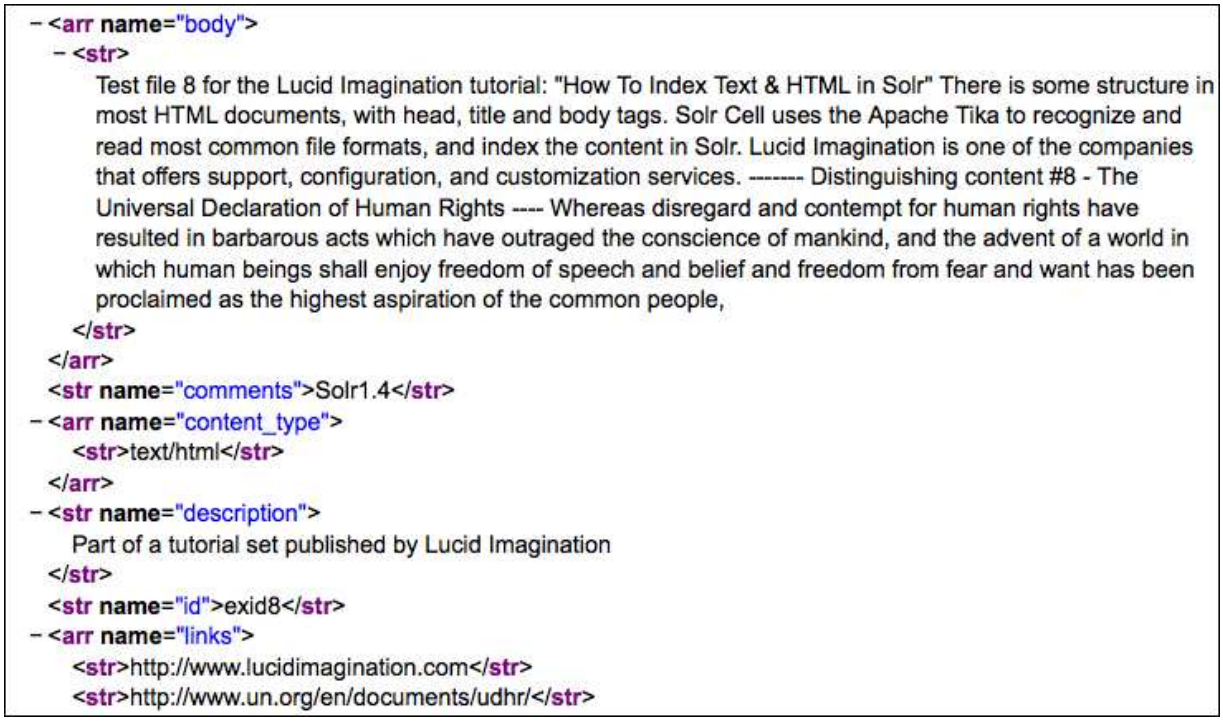
Note

Apache Tika uses several methods to identify file formats. These include extensions, like *.txt* or *.html*, MIME types such as *text/plain* or *application/pdf*, and known file format header patterns. It's always best to have your source files use these labels, rather than relying on Tika to guess.

Storing body text in a viewable field

As in the text file, indexing Example 8 uses the `fmap` parameter to set the text from within the `<body>` field of the HTML document to the `body` field which is in this example schema, so it will be both searchable and stored.

```
cURL -f
'http://localhost:8983/solr/update/extract?uprefix=attr_&fmap.content=body&commi
t=true&literal.id=exid8' -F "myfile=@lu-example-8.html"
```



Part 6: Using Solr indexing for Remote HTML Files

Using cURL to download and index remote files

The cURL utility is a fine way to download a file served by a Web server, which in this tutorial we'll call a **remote file**. With the `-O` flag (capital letter O, not the digit zero), cURL will save a copy of the file with the same name into the current working directory. If there's a file with that name already, it will be over-written, so be careful.

```
cURL -O http://www.lucidimagination.com/download/thtutorial/lu-example-9.html
```

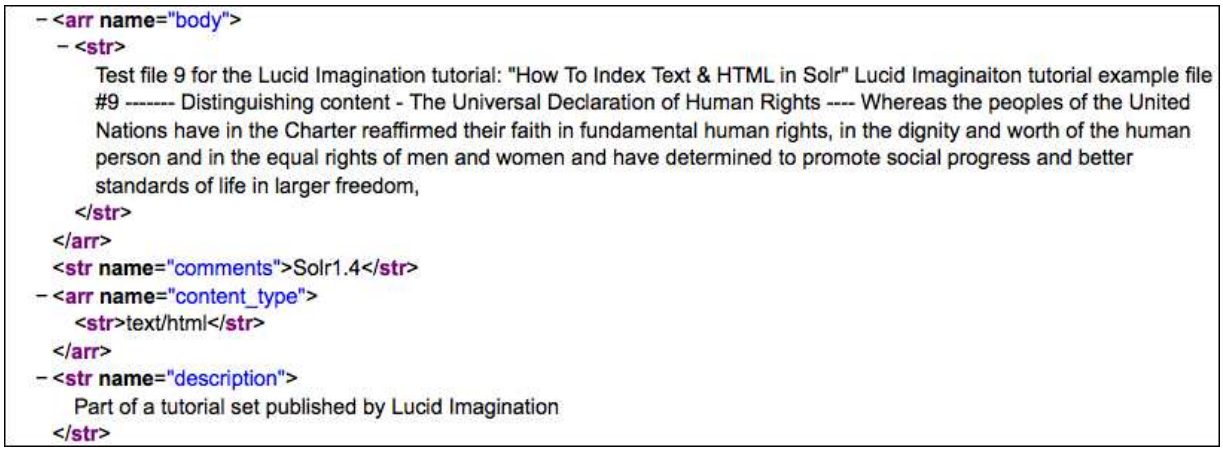
Note

If web access is unavailable, there's a copy of the `lu-example-9.html` file in the `remote` subdirectory in the zip file.

If you view the files in the local `examples-text-html` directory, there will be a `lu-example-9.html` file. The next step is to send it to Solr, which will use Solr Cell to index it.

```
cURL
"http://localhost:8983/solr/update/extract?literal.id=exid9&uprefix=attr_&fmap.c
ontent=body&commit=true" -F "exid9=@lu-example-9.html"
```

This will index and store all the text in the file, including the body, comments, and description.



File streaming for indexing remote documents

Solr also supports a *file streaming* protocol, sending the remote document URL to be indexed. For more information, see the ExtractingRequestHandler and ContentStream pages in the *LucidWorks Certified Distribution Reference Guide for Solr*, or the Solr wiki. Note that enabling remote streaming may create an access control security issue: for more information, see the Security page on the wiki.

Spidering tools

This tutorial doesn't cover the step of adding a spider (also known as a *crawler* or *robot*) to the indexing process. Spiders are programs that open web pages and follow links on the pages, to index a web site or an intranet server. This is how horizontal consumer web search providers such as Google, Ask, and Bing find so many pages.

Solr doesn't have an integrated spider, but works well with another Apache Lucene open source project, the Nutch crawler. There's a very helpful post on Lucid Imagination's site, Using Nutch with Solr, which explains further how this works.

Alternatives include Heritrix from the Internet Archive, JSpider, WebLech, Spider on Rails, and OpenWebSpider.

Conclusion and Additional Resources

Now that you've had the opportunity to try Solr on HTML content, the opportunities to build a search application with it are as diverse and broad as the content you need to search! Here are some resources you will find useful in building your own search applications.

Configuring the ExtractingRequestHandler in Chapter 6 of the *LucidWorks for Solr Certified Distribution Reference Guide*
http://www.lucidimagination.com/Downloads/LucidWorks-for-Solr/Reference-Guide

Solr Wiki: Extracting Request Handler (Solr Cell)
http://wiki.apache.org/solr/ExtractingRequestHandler

Tika http://lucene.apache.org/tika/

Content Extraction with Tika, by Sami Siren:
http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Content-Extraction-Tika

Optimizing Findability in Lucene and Solr, by Grant Ingersoll:
http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Optimizing-Findability-Lucene-and-Solr

About Lucid Imagination

Mission critical enterprise search applications in e-commerce, government, research, media, telecommunications, Web 2.0, and many more use Apache Lucene/Solr to ensure end users can find valuable, accurate information quickly and efficiently across the enterprise. Lucid Imagination complements the strengths of this technology with a foundation of commercial-grade software and services with unmatched expertise. Our software and services solutions help organizations optimize performance and achieve high-quality search results with their Lucene/Solr applications. Lucid Imagination customers include AT&T, Nike, Sears, Ford, Verizon, The Guardian, Elsevier, The Motley Fool, Cisco, Macy's and Zappos.

Lucid Imagination is here to help you meet the most demanding search application requirements. Our free LucidWorks Certified Distributions are based on these most popular open source search products, including free documentation. And with our industry-leading services, you can get the support, training, value added software, and high-level consulting and search expertise you need to create your enterprise-class search application with Lucene and Solr.

For more information on how Lucid Imagination can help search application developers, employees, customers, and partners find the information they need, please visit http://www.lucidimagination.com to access blog posts, articles, and reviews of dozens of successful implementations. Please e-mail specific questions to:

- Support and Service: support@lucidimagination.com
- Sales and Commercial: sales@lucidimagination.com
- Consulting: consulting@lucidimagination.com
- Or call: 1.650.353.4057

About the Author

Avi Rappoport really likes good search, and Solr is really good. She is the founder of Search Tools Consulting, which has given her the opportunity to work with site, portal, intranet and Enterprise search engines since 1998. She also speaks at search conferences, writes on search-related topics for InfoToday and other publishers, co-manages the LinkedIn Enterprise Search Engine Professionals group, and is the editor of SearchTools.com. Contact her at consult [at] searchtools . com or via the searchtools.com web site.