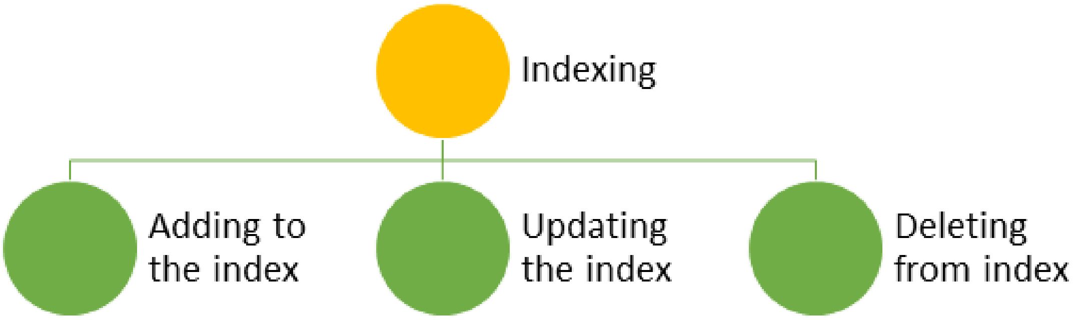
Lab 5. Data Indexing and Operations

In the last lab, we jumped into various text analysis methodologies, such as seeing the use of analyzers, filters, and tokenizers, to have an efficient text analysis.

In this lab, we will see ways to add data to Solr indexes.

Basics of Solr indexing

In order to make content available for searching, we need to index it first---as simple as that! The process of indexing essentially involves any one of the three activities as shown in this diagram:



Let's drill down and look at the indexing process, which has the following main actions:

- Adding content to the Solr Index
- Updating the index
- Deleting from the index

Now, there are two basic questions that might arise in your mind:

- From where does Solr accept data to be indexed? Or what are different sources from where data can be indexed?
- How do we index data from the sources that we have identified?

Common sources that the Solr index can get data from are:

- Database tables
- CSV files
- XML files
- Microsoft Word or PDF

The answers to "*How does the Solr index get data from the aforementioned sources?*" are as follows:

- Using client APIs
- Uploading XML files using HTTP requests to the Solr server
- Using the Apache Tika-based Solr Cell framework to ingest proprietary data formats, such as Word or PDF files

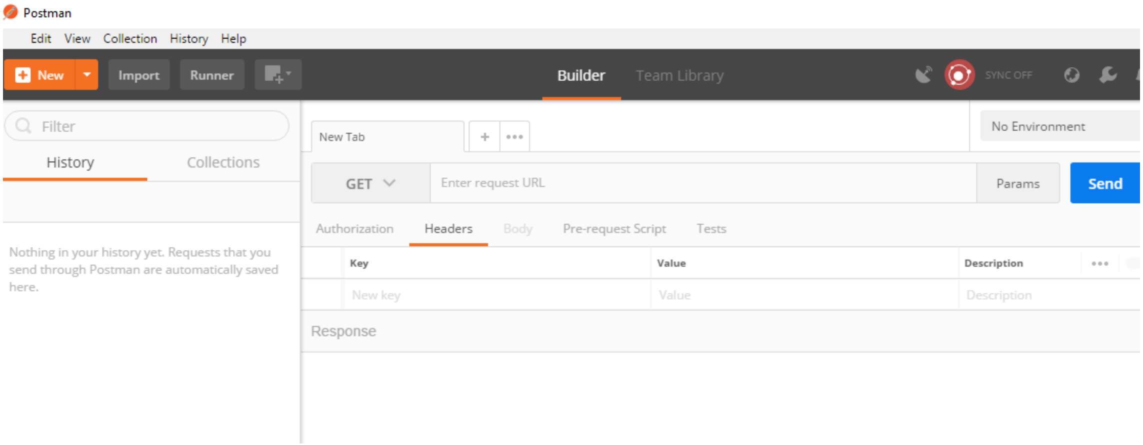
Installing Postman

For all HTTP-based service calls, we will be using Postman to invoke such services.

Note: Postman has been already installed in the lab environment.

Postman can be downloaded from https://www.getpostman.com/. The site provides installation instructions for each of the major operating systems. In this lab, we will do our exercise based on Windows, so we'll proceed to install the Windows executable.

Once you've downloaded and installed Postman, you should see a screen like this:



Nothing in your history yet. Requests that you send through Postman are automatically saved here.

Don't worry! We will get into the details as to how to use Postman later in the lab. Alternatively, you can use `curl` to do the same, but I prefer Postman due to its easy usability.

Exploring the post tool

In order to index different types of content to the Solr server, Solr provides a command-line tool.

To run this tool in Unix, use the following command:

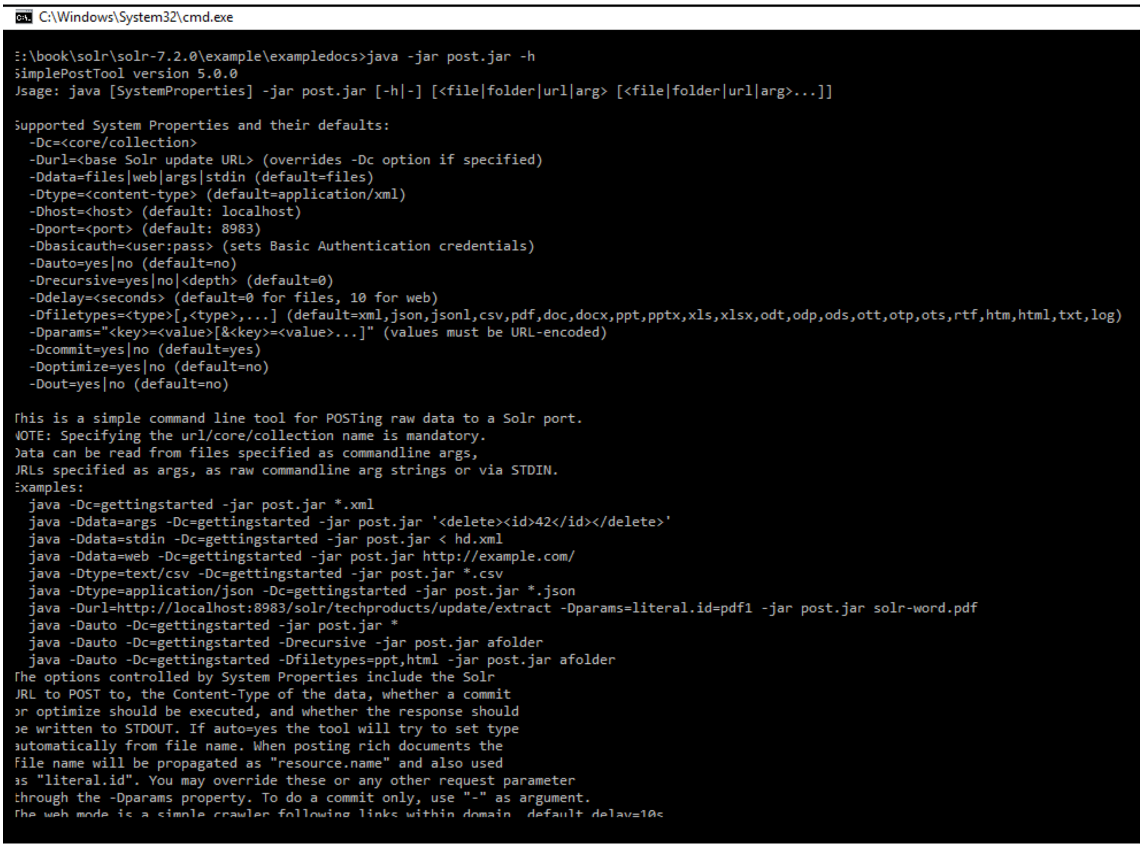
```
post -c gettingstarted /opt/solr/example/exampledocs/books.json
```

For Windows, it gets a bit tricky as `bin/post` is available only as a Unix shell script.

On Windows, we need to use `SimplePostTool`, which is a standalone Java program and can be packaged in `post.jar` located at `/opt/solr/example/exampledocs`. Navigate to `/opt/solr/example/exampledocs` and issue this command:

```
java -jar post.jar -h
```

We will see the following output:



```
E:\book\solr\solr-7.2.0\example\exampledocs>java -jar post.jar -h
SimplePostTool version 5.0.0
Usage: java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folder|url|arg>...]]

Supported System Properties and their defaults:
  -Dc=<core/collection>
  -Durl=<base Solr update URL> (overrides -Dc option if specified)
  -Ddata=files|web|args|stdin (default=files)
  -Dtype=<content-type> (default=application/xml)
  -Dhost=<host> (default: localhost)
  -Dport=<port> (default: 8983)
  -Dbasicauth=<user:pass> (sets Basic Authentication credentials)
  -Dauto=yes|no (default=no)
  -Drecursive=yes|no|<depth> (default=0)
  -Ddelay=<seconds> (default=0 for files, 10 for web)
  -Dfiletypes=<type>[,<type>,...] (default=xml,json,jsonl,csv,pdf,doc,docx,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log)
  -Dparams="<key>=<value>[&<key>=<value>...]" (values must be URL-encoded)
  -Dcommit=yes|no (default=yes)
  -Doptimize=yes|no (default=no)
  -Dout=yes|no (default=no)

This is a simple command line tool for POSTing raw data to a Solr port.
NOTE: Specifying the url/core/collection name is mandatory.
Data can be read from files specified as commandline args,
URLs specified as args, as raw commandline arg strings or via STDIN.
Examples:
  java -Dc=gettingstarted -jar post.jar *.xml
  java -Ddata=args -Dc=gettingstarted -jar post.jar '<delete><id>42</id></delete>'
  java -Ddata=stdin -Dc=gettingstarted -jar post.jar < hd.xml
  java -Ddata=web -Dc=gettingstarted -jar post.jar http://example.com/
  java -Dtype=text/csv -Dc=gettingstarted -jar post.jar *.csv
  java -Dtype=application/json -Dc=gettingstarted -jar post.jar *.json
  java -Durl=http://localhost:8983/solr/techproducts/update/extract -Dparams=literal.id=pdf1 -jar post.jar solr-word.pdf
  java -Dauto -Dc=gettingstarted -jar post.jar *
  java -Dauto -Dc=gettingstarted -Drecursive -jar post.jar afolder
  java -Dauto -Dc=gettingstarted -Dfiletypes=ppt,html -jar post.jar afolder
The options controlled by System Properties include the Solr
URL to POST to, the Content-Type of the data, whether a commit
or optimize should be executed, and whether the response should
be written to STDOUT. If auto=yes the tool will try to set type
automatically from file name. When posting rich documents the
file name will be propagated as "resource.name" and also used
as "literal.id". You may override these or any other request parameter
through the -Dparams property. To do a commit only, use "-" as argument.
The web mode is a simple crawler following links within domain, default delay=10s
```

As you can see, we get the full documentation of the post tool.

Issue the following command to run the post tool in Windows:

```
java -Dc=gettingstarted -jar /opt/solr/example/exampledocs/post.jar
example/films/films.json
```

This will index content from `films.json` to the server at `localhost:8983`.

In order to index all the documents with the extension XML, issue the following command from the `SOLR_HOME` directory:

```
java -jar /opt/solr/example/exampledocs/post.jar -Dc gettingstarted *.xml
```

Let's say you want to delete a document with ID `23` from the `gettingstarted` collection/core; you can issue the following command:

```
java -jar /opt/solr/example/exampledocs/post.jar -Dc gettingstarted -Dd '<delete>
<id>23</id></delete>'
```

Similarly, we can index `.json` and `.csv` files as shown here:

```
java -jar /opt/solr/example/exampledocs/post.jar -Dc gettingstarted *.json
java -jar /opt/solr/example/exampledocs/post.jar -Dc gettingstarted *.csv
```

As you can see, there is not much difference in indexing CSV, XML, and JSON documents.

Now Let's learn how to index rich documents. Let's say we want to index a Word document; we will issue the following command:

```
java -jar /opt/solr/example/exampledocs/post.jar -Dc gettingstarted sample.doc
```

If we want to specify a bunch of documents of type `.pdf` and `.doc` in a folder named `samplefolder`, then we issue the following command:

```
java -jar /opt/solr/example/exampledocs/post.jar -Dc gettingstarted -Dfiletypes
doc,pdf samplefolder/
```

Now that we have learned how to use the post tool for indexing, Let's see another technique to do the same, known as **index handlers**. Understanding index handlers

---

Solr provides a native way to index structured documents such as XML, JSON, and CSV using index handlers.

The default request handler (which is configured by default) is as follows:

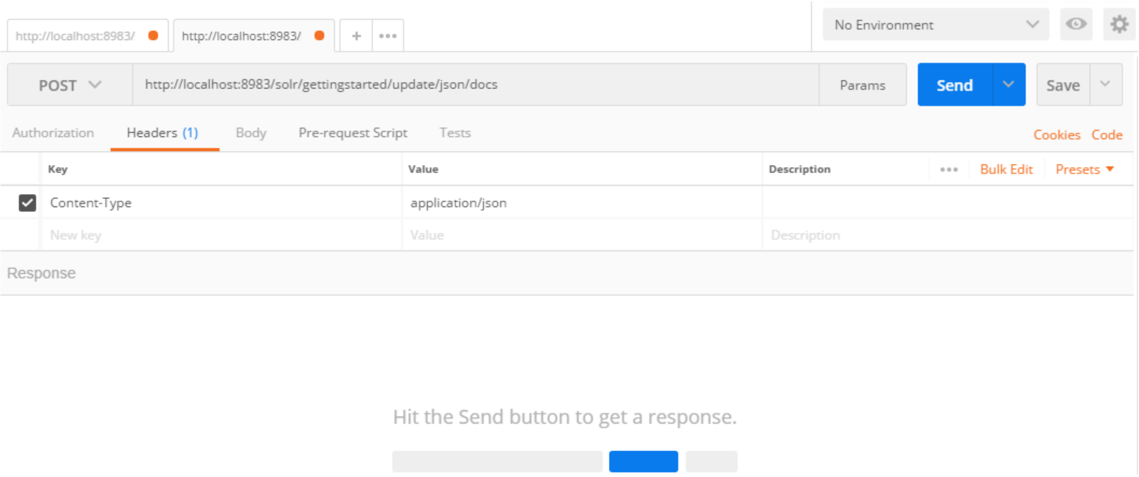
```
<requestHandler name="/update" class="solr.UpdateRequestHandler" />
```

Or you can mention them separately in `solrconfig.xml`:

```
<requestHandler name="/update/json" class="solr.UpdateRequestHandler">
     <lst name="invariants">
         <str name="stream.contentType">application/json</str>
     </lst>
</requestHandler>
<requestHandler name="/update/csv" class="solr.UpdateRequestHandler">
     <lst name="invariants">
         <str name="stream.contentType">application/csv</str>
     </lst>
</requestHandler>
```

Working with an index handler with the XML format

Now Let's try to add some content to our index using the XML format. Open the Postman tool and add the URL `http://localhost:8983/solr/gettingstarted/update`, as shown in the following screenshot:



POST http://localhost:8983/solr/gettingstarted/update/json/docs

Key	Value	Description
☑ Content-Type	application/json	
New key	Value	Description

Hit the Send button to get a response.

Note that we have selected the method type to `POST`. You also need to add a request header, `Content-Type`, to `text/xml`. Most of this can be done in Postman as we just select values from the dropdown, as shown previously.

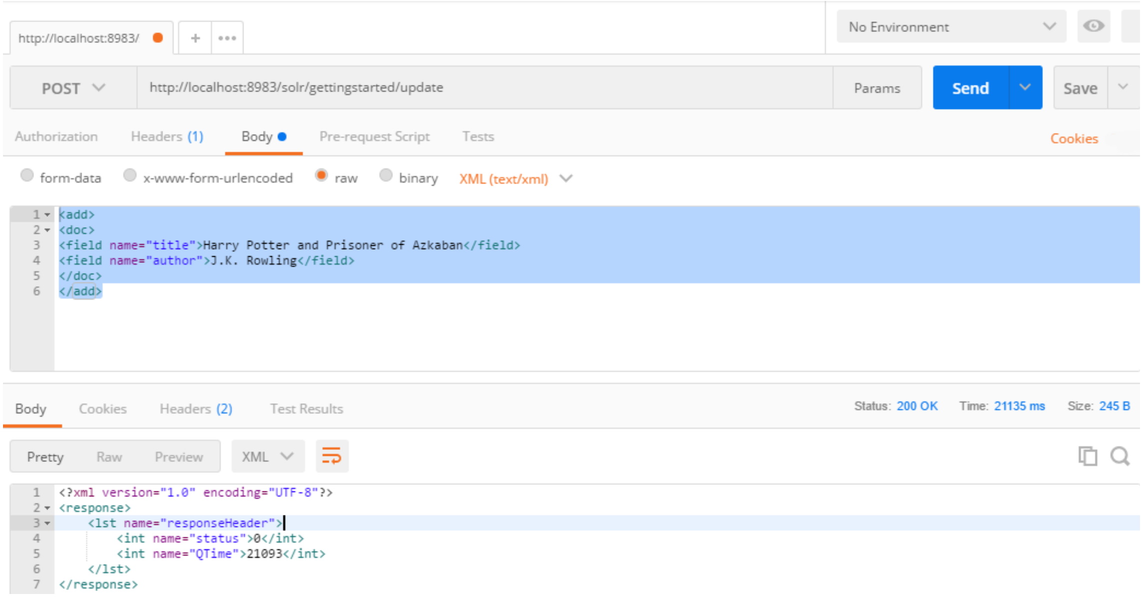
Now switch to the `Body` tab and add the following content:

```
<add>
    <doc>
        <field name="title">Harry Potter and Prisoner of
          Azkaban</field>
        <field name="author">J.K. Rowling</field>
    </doc>
</add>
```

For adding any documents to index, the XML schema should have the following elements:

- `<add>`: Specifies that the operation we are going to perform will add one or more documents
- `<doc>`: This has fields that make up the whole document
- `<field>`: Specifies each field to be added for the document

Once you are done with the aforementioned steps, click on the `Send` button in Postman. If everything goes right, you should see the following response:



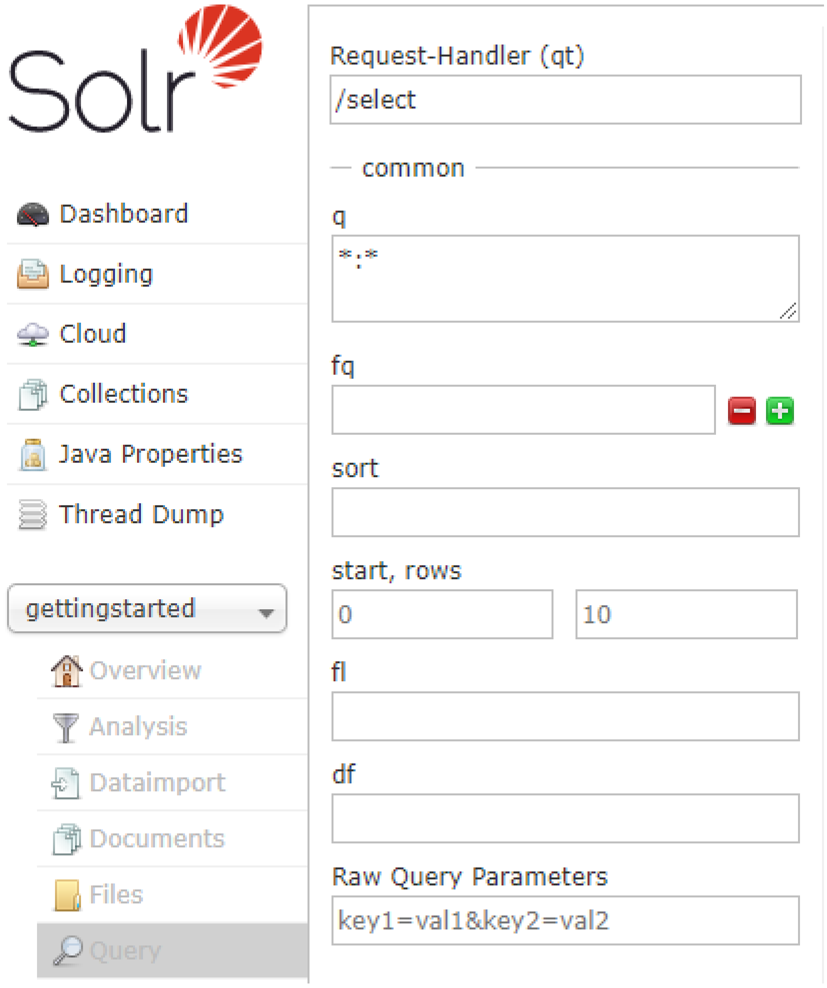
http://localhost:8983/solr/gettingstarted/update

```
<add>
<doc>
<field name="title">Harry Potter and Prisoner of Azkaban</field>
<field name="author">J.K. Rowling</field>
</doc>
</add>
```

Status: 200 OK Time: 21135 ms Size: 245 B

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <lst name="responseHeader">
        <int name="status">0</int>
        <int name="QTime">21093</int>
    </lst>
</response>
```

This indicates that the document has been successfully added to the index. In order to verify the documents, you can go to the Solr admin console in the browser, select `gettingstarted`, and then navigate to the `Query` section, as follows:



Request-Handler (qt): /select

q: *:*

fq:

sort:

start, rows: 0, 10

fl:

df:

Raw Query Parameters: key1=val1&key2=val2

Once you are on the previous page, click on the `Execute Query` button available at the very bottom of the page. You should see something like this:

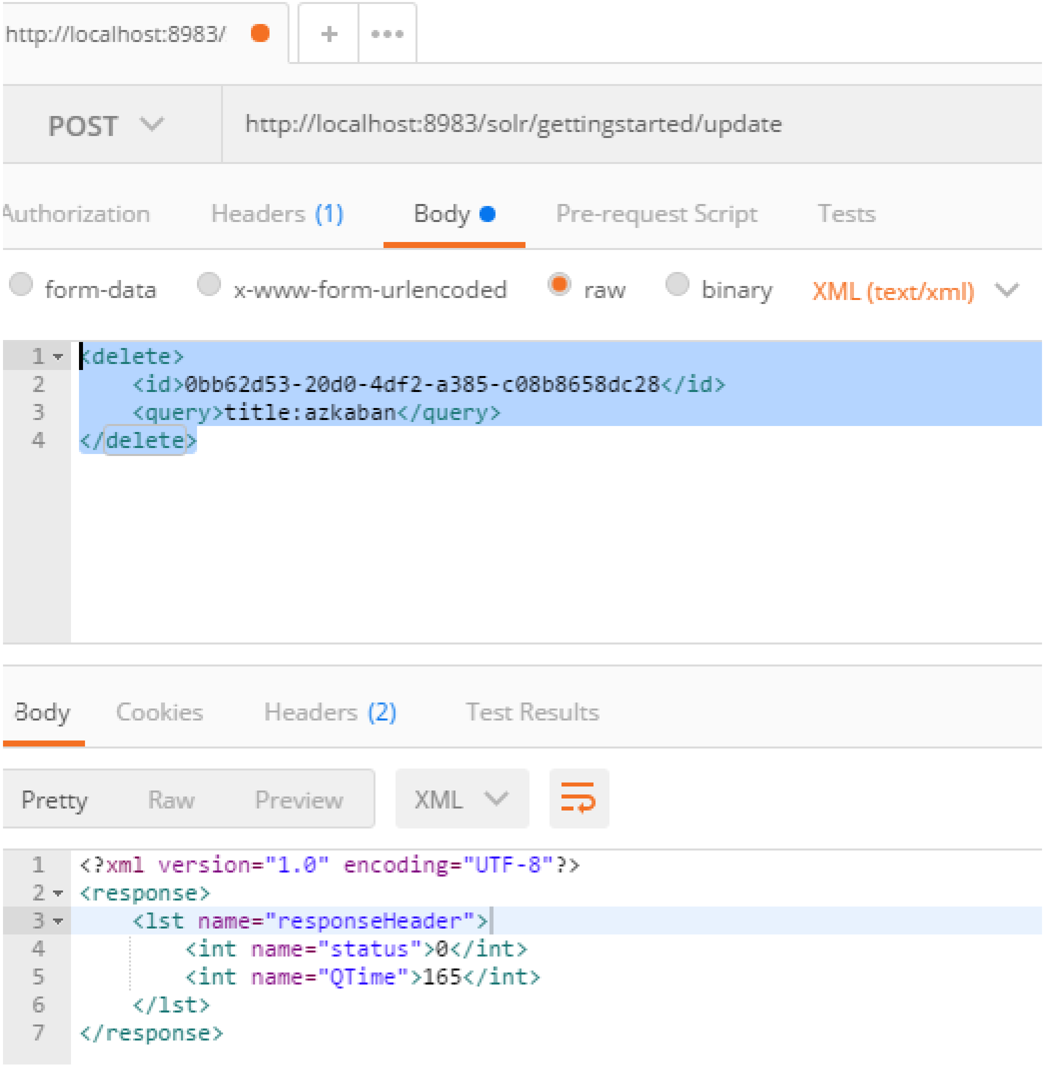
```
{
 "responseHeader":{
     "zkConnected":true,
     "status":0,
     "QTime":18,
     "params":{
         "q":"*:*",
         "_":"1514434725886"}},
 "response":{"numFound":3,"start":0,"maxScore":2.0,"docs":[
     {
         "title":["Harry Potter and Chamber of Secrets"],
         "author":["J.K. Rowling"],
         "id":"0bb62d53-20d0-4df2-a385-c08b8658dc28",
         "title_str":["Harry Potter and Chamber of Secrets"],
         "author_str":["J.K. Rowling"],
         "_version_":1587892939487444992},
     {
         "title":["Harry Potter and Prisoner of Azkaban"],
         "author":["J.K. Rowling"],
         "id":"401f8336-88d9-478c-93a2-2de070188a3d",
         "title_str":["Harry Potter and Prisoner of Azkaban"],
         "author_str":["J.K. Rowling"],
         "_version_":1587999318593241088},
     {
         "title":["Harry Potter and Philosophers Stone"],
         "author":["J.K. Rowling"],
         "id":"dcbfd369-2a4b-4e9d-9713-1af373849438",
         "title_str":["Harry Potter and Philosophers Stone"],
         "author_str":["J.K. Rowling"],
         "_version_":1587892903981613056}]
 }}
```

As you can see, the document with the title `Harry Potter and Prisoner of Azkaban` is added to the index. I had previously added some other documents, so that's why you are seeing two additional entries.

Now Let's try to delete some documents from the index. Open Postman and just replace the content with the following:

```
<delete>
     <id>0bb62d53-20d0-4df2-a385-c08b8658dc28</id>
     <query>title:azkaban</query>
</delete>
```

In Postman, it will look something like this once executed:



POST http://localhost:8983/solr/gettingstarted/update

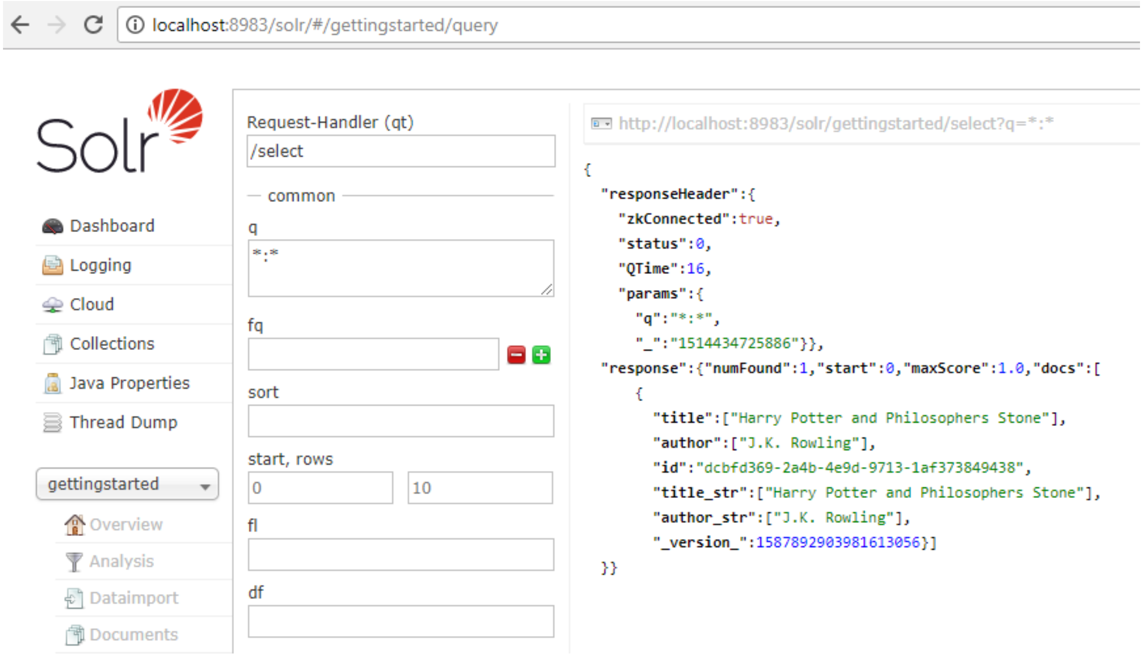
● raw XML (text/xml)

```
<delete>
    <id>0bb62d53-20d0-4df2-a385-c08b8658dc28</id>
    <query>title:azkaban</query>
</delete>
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <lst name="responseHeader">
        <int name="status">0</int>
        <int name="QTime">165</int>
    </lst>
</response>
```

What we have done is deleted a couple of records, one with a query where the title is `Azkaban` and one with the ID `0bb62d53-20d0-4df2-a385-c08b8658dc28`. In order to verify this, go to the admin console and click on the **`Execute Query`** button once again:



localhost:8983/solr/#/gettingstarted/query

Request-Handler (qt): /select

q: `*:*`

fq:

sort:

start, rows: 0, 10

fl:

df:

http://localhost:8983/solr/gettingstarted/select?q=*:*

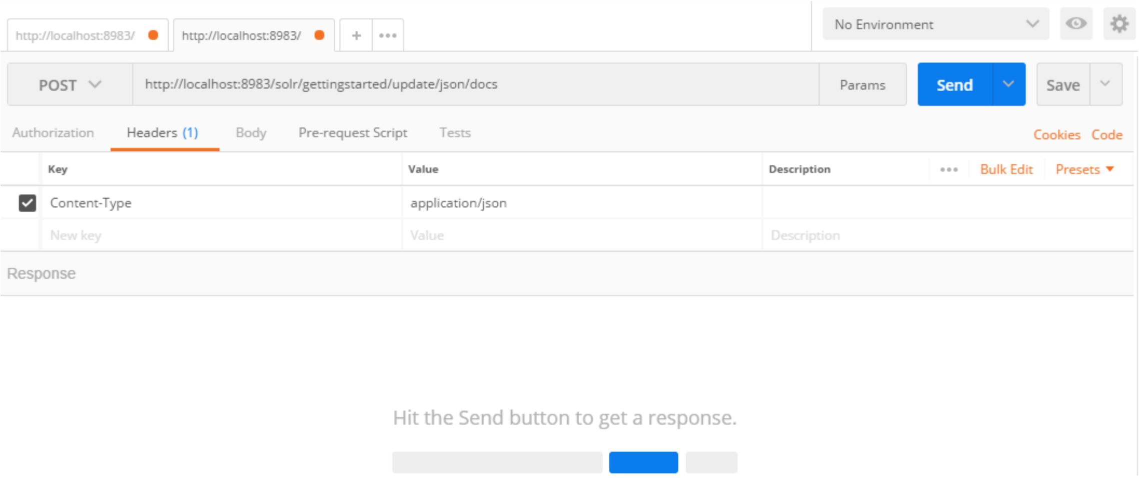
```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":16,
    "params":{
      "q":"*:*",
      "_":"1514434725886"}},
  "response":{"numFound":1,"start":0,"maxScore":1.0,"docs":[
      {
        "title":["Harry Potter and Philosophers Stone"],
        "author":["J.K. Rowling"],
        "id":"dcbfd369-2a4b-4e9d-9713-1af373849438",
        "title_str":["Harry Potter and Philosophers Stone"],
        "author_str":["J.K. Rowling"],
        "_version_":1587892903981613056}]
  }}
```

As you can see, there is only one entry now as the other two entries have been deleted.

Index handler with JSON

Solr also supports JSON-formatted documents to be indexed. Let's look at a simple example of indexing just one document. To add documents in JSON format on our `gettingstarted` collection, we need to use the following **URL**: `http://localhost:8983/solr/gettingstarted/update/json/docs`.

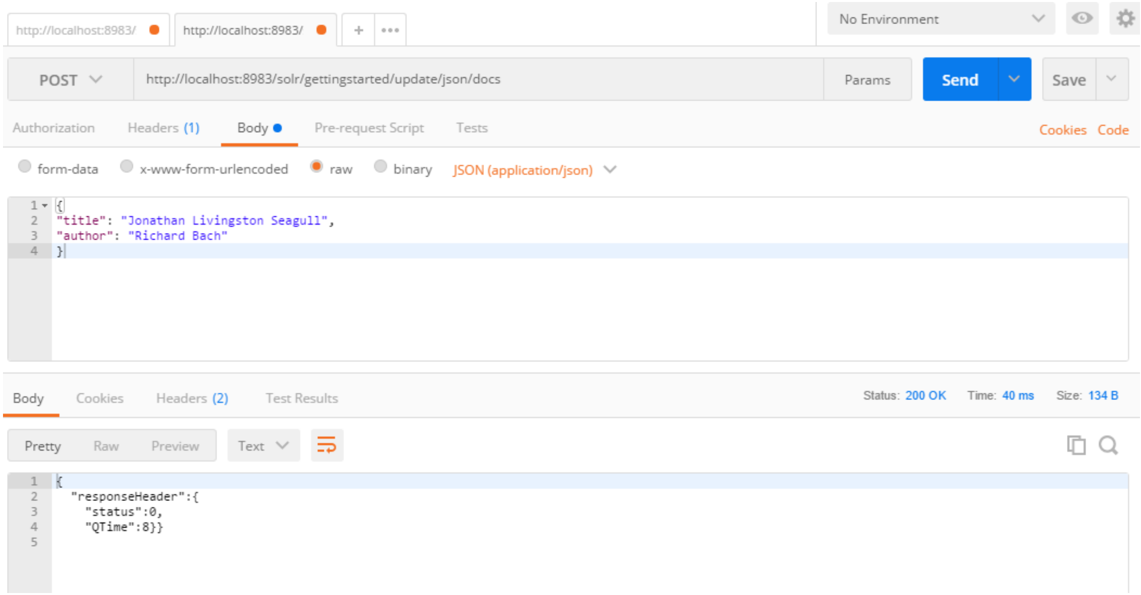
Open Postman and create a new request with this URL. See the following screenshot for clarity:



POST http://localhost:8983/solr/gettingstarted/update/json/docs

	Key	Value	Description
☑	Content-Type	application/json	
	New key	Value	Description

As you can see, we have set **`Content-Type`** to `application/json`. Now click on the **`Body`** tab and put the JSON content as follows:



POST http://localhost:8983/solr/gettingstarted/update/json/docs

Body: ● raw — JSON (application/json)

```
{
"title": "Jonathan Livingston Seagull",
"author": "Richard Bach"
}
```

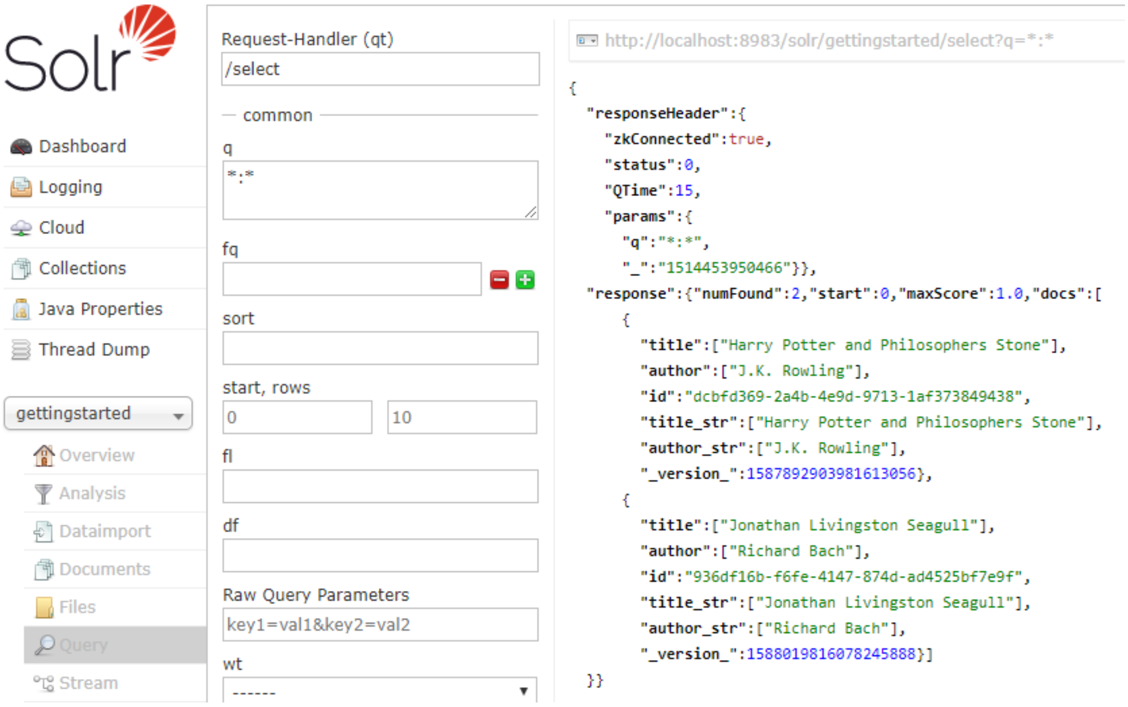
Status: 200 OK Time: 40 ms Size: 134 B

```
{
  "responseHeader":{
    "status":0,
    "QTime":8}}
```

Once you are done, execute the request and it will run to success with the following response:

```
{
 "responseHeader":{
 "status":0,
 "QTime":8}
}
```

The `status` value set to `0` means it is a success. If it is a non-zero value, then it means there is a failure. In order to validate this, go to the Solr admin console and execute the query as we did earlier:



Request-Handler (qt): /select

q: `*:*`

start, rows: 0, 10

Raw Query Parameters: key1=val1&key2=val2

wt: ------

http://localhost:8983/solr/gettingstarted/select?q=*:*

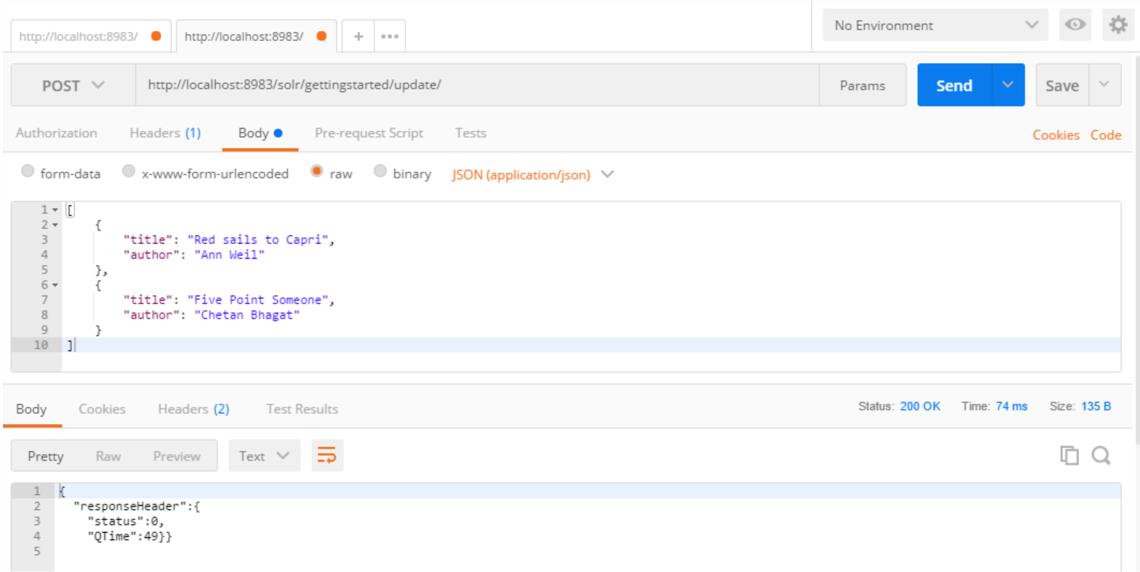
```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":15,
    "params":{
      "q":"*:*",
      "_":"1514453950466"}},
  "response":{"numFound":2,"start":0,"maxScore":1.0,"docs":[
      {
        "title":["Harry Potter and Philosophers Stone"],
        "author":["J.K. Rowling"],
        "id":"dcbfd369-2a4b-4e9d-9713-1af373849438",
        "title_str":["Harry Potter and Philosophers Stone"],
        "author_str":["J.K. Rowling"],
        "_version_":1587892903981613056},
      {
        "title":["Jonathan Livingston Seagull"],
        "author":["Richard Bach"],
        "id":"936df16b-f6fe-4147-874d-ad4525bf7e9f",
        "title_str":["Jonathan Livingston Seagull"],
        "author_str":["Richard Bach"],
        "_version_":1588019816078245888}]
  }}
```

Once you execute the query, you should see that the index also contains the new document that we added using the JSON format.

In order to add multiple documents, you have to pass a JSON array instead of a single document:

```
[
    {
        "title": "Red sails to Capri",
        "author": "Ann Weil"
    },
    {
        "title": "Five Point Someone",
        "author": "Chetan Bhagat"
    }
]
```

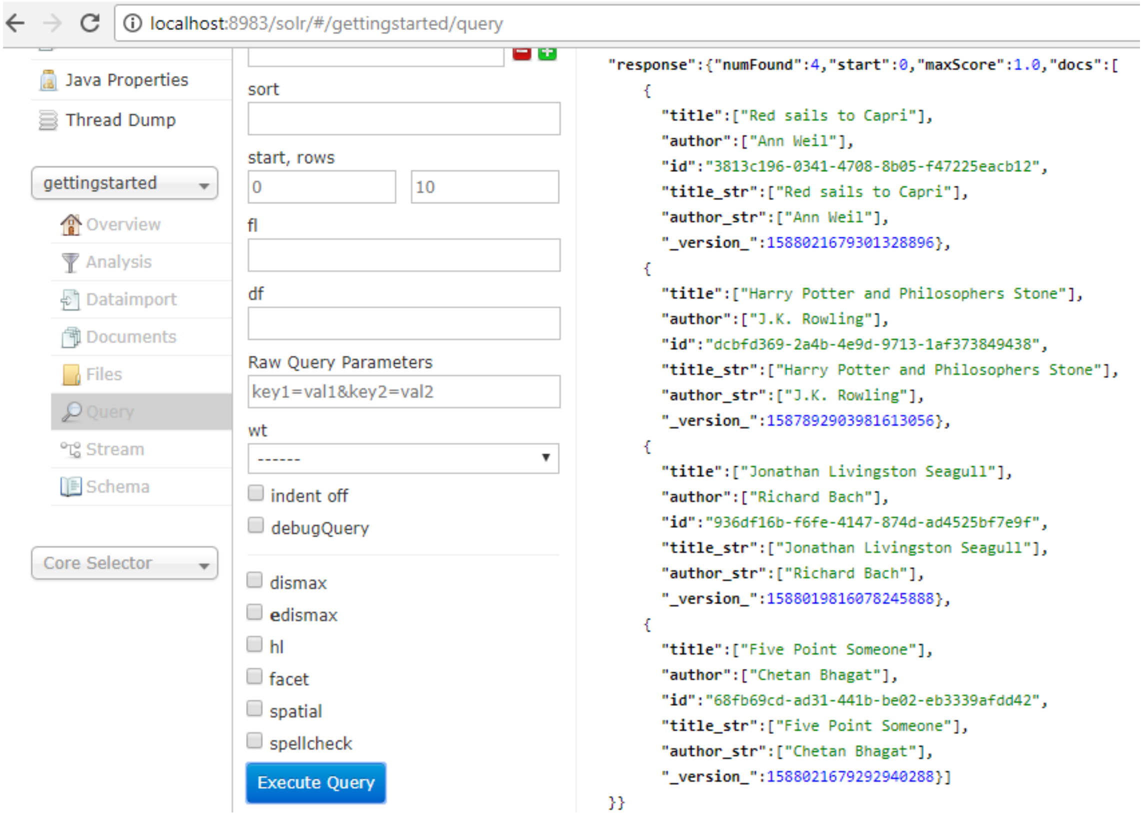
The URL has to be `http://localhost:8983/solr/gettingstarted/update/` , as shown in the following screenshot:



```
[
    {
        "title": "Red sails to Capri",
        "author": "Ann Weil"
    },
    {
        "title": "Five Point Someone",
        "author": "Chetan Bhagat"
    }
]
```

```
{
  "responseHeader":{
    "status":0,
    "QTime":49}}
```

You will get a success response with `status` set to `0` once you execute the request. In order to validate, navigate to the Solr admin panel and execute the query:



sort

start, rows 0 10

fl

df

Raw Query Parameters key1=val1&key2=val2

wt ------

□ indent off
□ debugQuery

□ dismax
□ edismax
□ hl
□ facet
□ spatial
□ spellcheck

```
"response":{"numFound":4,"start":0,"maxScore":1.0,"docs":[
    {
      "title":["Red sails to Capri"],
      "author":["Ann Weil"],
      "id":"3813c196-0341-4708-8b05-f47225eacb12",
      "title_str":["Red sails to Capri"],
      "author_str":["Ann Weil"],
      "_version_":1588021679301328896},
    {
      "title":["Harry Potter and Philosophers Stone"],
      "author":["J.K. Rowling"],
      "id":"dcbfd369-2a4b-4e9d-9713-1af373849438",
      "title_str":["Harry Potter and Philosophers Stone"],
      "author_str":["J.K. Rowling"],
      "_version_":1587892903981613056},
    {
      "title":["Jonathan Livingston Seagull"],
      "author":["Richard Bach"],
      "id":"936df16b-f6fe-4147-874d-ad4525bf7e9f",
      "title_str":["Jonathan Livingston Seagull"],
      "author_str":["Richard Bach"],
      "_version_":1588019816078245888},
    {
      "title":["Five Point Someone"],
      "author":["Chetan Bhagat"],
      "id":"68fb69cd-ad31-441b-be02-eb3339afdd42",
      "title_str":["Five Point Someone"],
      "author_str":["Chetan Bhagat"],
      "_version_":1588021679292940288}]
}}
```

Your response will now have the two documents that you have added and you should see the following documents in the response:

```
{
 "responseHeader":{
 "zkConnected":true,
 "status":0,
 "QTime":35,
 "params":{
 "q":"*:*",
 "_":"1514453950466"}},
 "response":{"numFound":4,"start":0,"maxScore":1.0,"docs":[
     {
     "title":["Red sails to Capri"],
     "author":["Ann Weil"],
     "id":"3813c196-0341-4708-8b05-f47225eacb12",
     "title_str":["Red sails to Capri"],
     "author_str":["Ann Weil"],
     "_version_":1588021679301328896},
     {
     "title":["Harry Potter and Philosophers Stone"],
     "author":["J.K. Rowling"],
     "id":"dcbfd369-2a4b-4e9d-9713-1af373849438",
     "title_str":["Harry Potter and Philosophers Stone"],
     "author_str":["J.K. Rowling"],
     "_version_":1587892903981613056},
     {
```

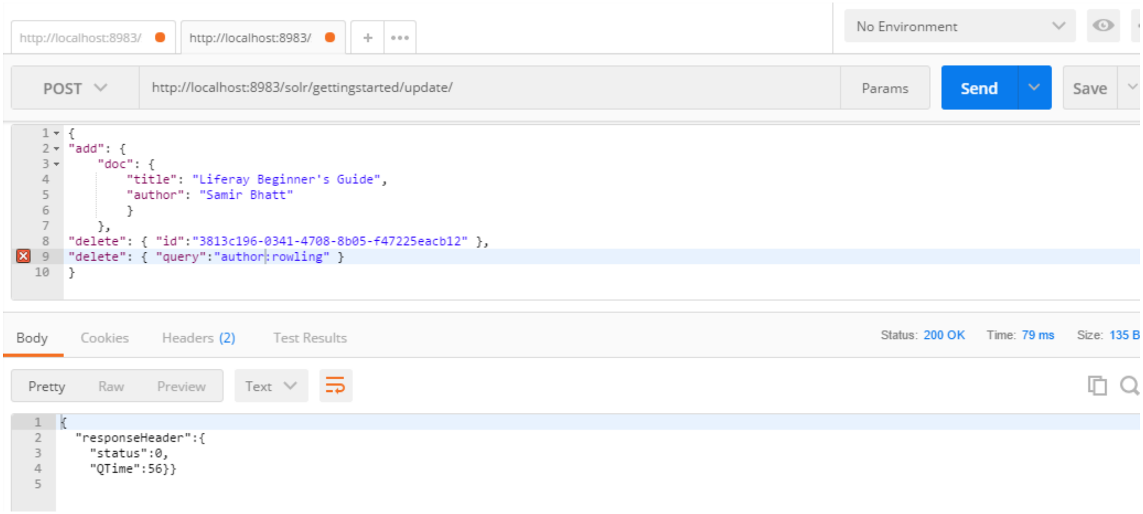
```
     "title":["Jonathan Livingston Seagull"],
     "author":["Richard Bach"],
     "id":"936df16b-f6fe-4147-874d-ad4525bf7e9f",
     "title_str":["Jonathan Livingston Seagull"],
     "author_str":["Richard Bach"],
     "_version_":1588019816078245888},
     {
     "title":["Five Point Someone"],
     "author":["Chetan Bhagat"],
     "id":"68fb69cd-ad31-441b-be02-eb3339afdd42",
     "title_str":["Five Point Someone"],
     "author_str":["Chetan Bhagat"],
     "_version_":1588021679292940288}]
 }}
```

You can also add, update, or delete documents in a single operation. To do this, Let's try to delete some documents and add a new document:

```
{
     "add": {
     "doc": {
         "title": "Liferay Beginner's Guide",
         "author": "Samir Bhatt"
     }
 },
    "delete": { "id":"3813c196-0341-4708-8b05-f47225eacb12" },
    "delete": { "query":"author:rowling" }
}
```

Here, we are executing both `add` and `delete` operations in a single request. We have added a new book, deleted a book with `id`, and deleted a book with `query` where the author is `rowling`.

The request in Postman will look something like this:



POST http://localhost:8983/solr/gettingstarted/update/

```
{
"add": {
    "doc": {
        "title": "Liferay Beginner's Guide",
        "author": "Samir Bhatt"
        }
    },
"delete": { "id":"3813c196-0341-4708-8b05-f47225eacb12" },
"delete": { "query":"author:rowling" }
}
```

Status: 200 OK   Time: 79 ms   Size: 135 B

```
{
  "responseHeader":{
    "status":0,
    "QTime":56}}
```

In order to execute this, hit the `Send` button. You will get a success response. In this way, we can add and delete documents in a single operation.

Apache Tika and indexing

We have seen how to index data from a standard file format such as JSON or XML. But what about proprietary file formats such as Word and PDF? Luckily, Solr comes to the rescue with the use of the Apache Tika project. The Tika framework provides a way to incorporate various file formats such as Word and PDF.

Internally, Tika uses the Apache PDFBox parser to parse PDF and Apache POI for the Word format. Solr provides `ExtractingRequestHandler`, which makes use of Tika to upload binary files and to index as well as extract data.

This framework in Solr is known as Solr Cell, which is an abbreviation of Solr content extraction library, the name when this framework was under development.

Solr Cell basics

As we have earlier seen that, the Solr Cell framework leverages the Tika framework. Let's look at some basic concepts about this.

Please specify the MIME type for Tika explicitly to specify the document type. This has to be done with the `stream.type` parameter or else Tika will decide the document type provided on its own.

Tika creates some additional metadata on its own, such as `Title`, `Author`, and `Subject`, which respects `DublinCore`. Some of the file types where metadata can be extracted are as follows:

- HTML
- XML and derived formats such as XHTML, OOXML and ODF
- Formats of MS Office document types
- **OpenDocument** (**ODF**)
- Formats with iWorks document
- PDF
- Email formats
- Crypto formats
- **Rich Text Format** (**RTF**)
- Electronic publication
- Packaging and compression formats such as `.tar`, `.zip`, and `.7zip` files
- Text format
- Help formats
- Feed and syndication formats (RSS and atom feeds)
- Audio formats
- JARs and Java class files
- Video formats
- Cad formats
- Scientific formats
- EXE programs and libraries
- Image formats
- Source code
- Font formats

All extracted text from any of these formats is mapped with content field. Along with these formats, Tika's metadata fields can be mapped to Solr fields.

First, Tika produces an XHTML stream, which is passed to the SAX `ContentHandler`, and then Solr acts on various SAX events; finally it creates the fields to index. Since there is an XML-based parser, we can apply an XPath expression to XHTML to filter the content.

Indexing a binary using Tika

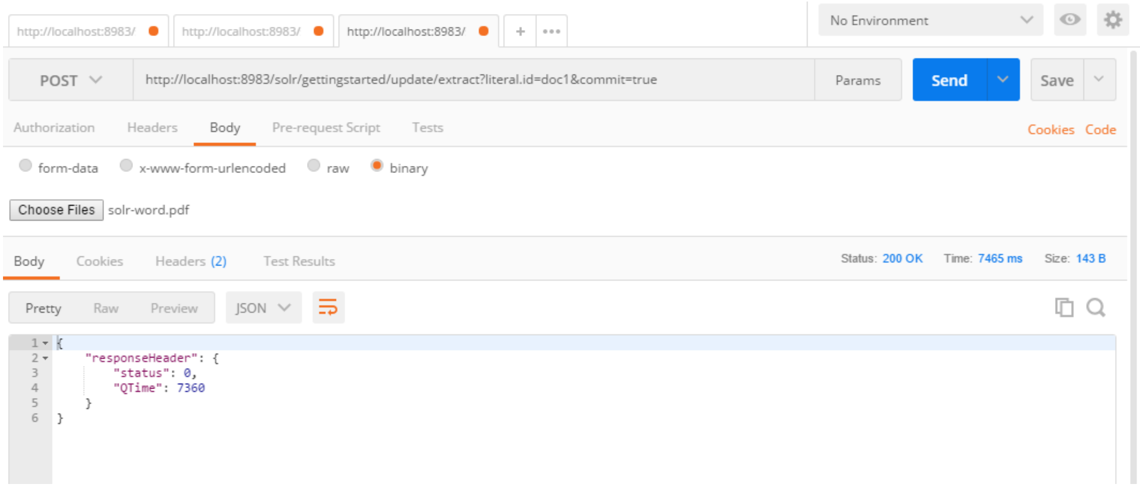
Now Let's get our hands dirty and start putting Tika to use. For this example, we will use the `gettingstarted` schema. I will just start one cloud node that created earlier for demo purposes.

To start only one node, issue a command as follows. Note that the path of the node may change as per your setup:

```
solr start -cloud -p 8983 -s E:\book\solr\solr-7.2.0\example\cloud\node1\solr
```

We will index a sample PDF provided by Solr. The PDF is available at `SOLR_HOME//opt/solr/example/exampledocs` by the name of `solr-word.pdf` .

Open Postman and create a new POST request with the URL `http://localhost:8983/solr/gettingstarted/update/extract?literal.id=doc1&commit=true` . Upload the file in the **`binary`** section. Finally submit the request as follows:



POST http://localhost:8983/solr/gettingstarted/update/extract?literal.id=doc1&commit=true

Choose Files solr-word.pdf

Status: 200 OK Time: 7465 ms Size: 143 B

```
{
    "responseHeader": {
        "status": 0,
        "QTime": 7360
    }
}
```

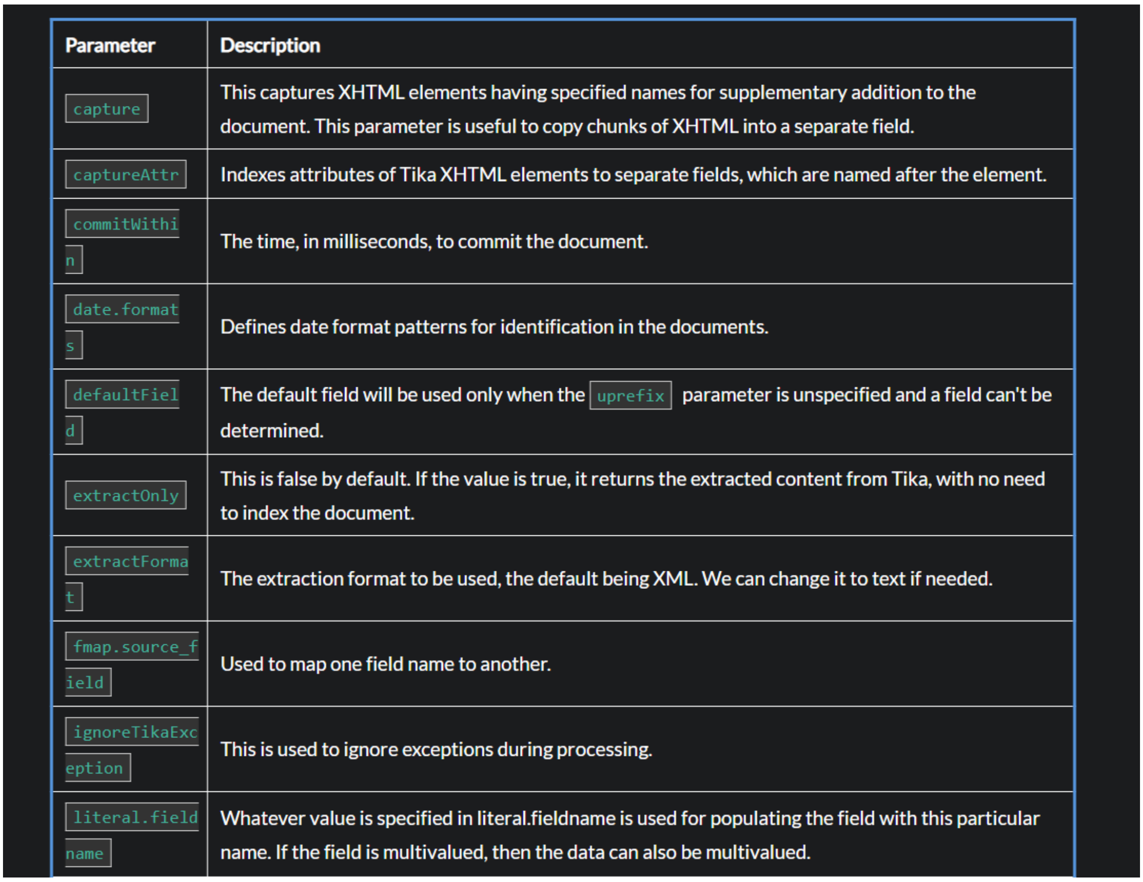
You will see that the request is successful, with `status` code `0` . In order to verify what data has been indexed, we can use Postman to query for PDF documents using the following URL: `http://localhost:8983/solr/gettingstarted/select?q=pdf` .

You will see a response as follows:

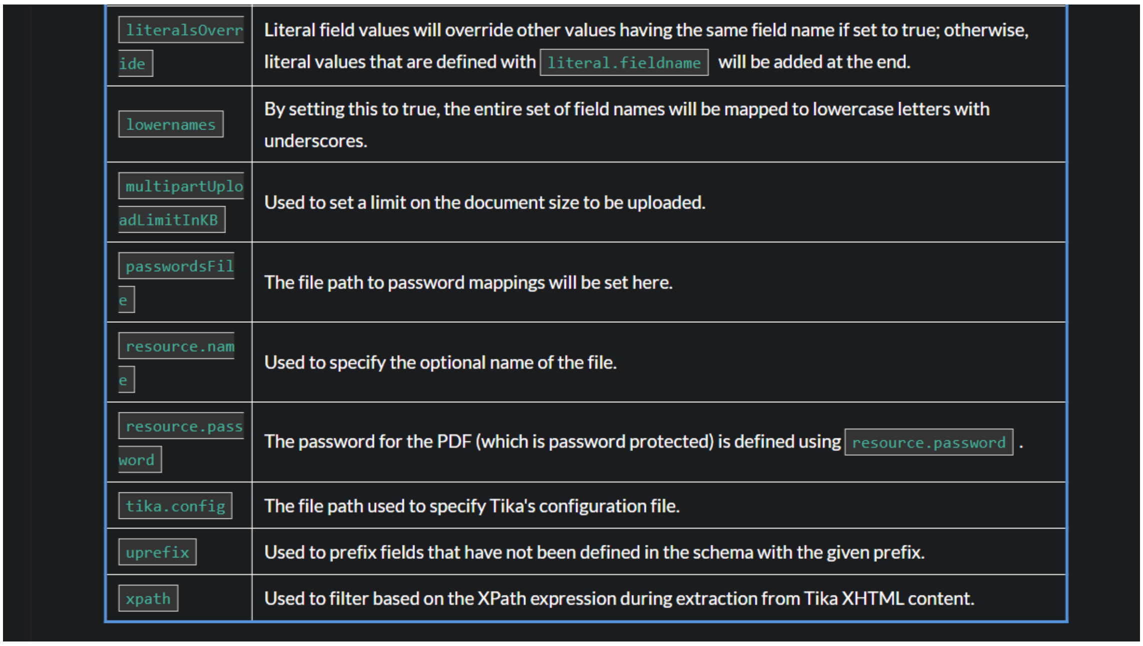
```
{
 "responseHeader":
    {
        "zkConnected": true,"status": 0,"QTime": 46,"params": {"q": "pdf"}
    },
  "response": {"numFound": 1,"start": 0,"maxScore": 0.55955875,
     "docs": [{"id": "doc1",
         "date": ["2008-11-13T13:35:51Z"],
         "pdf_docinfo_custom_aapl_keywords": ["solr, word, pdf"],
         "pdf_pdfversion": [1.3],
         "pdf_docinfo_title": ["solr-word"],
         "xmp_creatortool": ["Microsoft Word"],
         "stream_content_type": ["text/plain"],
         "access_permission_can_print_degraded": [true],
         "subject": ["solr word"],
         "dc_format": ["application/pdf; version=1.3"],
         "pdf_docinfo_creator_tool": ["Microsoft Word"],
         "access_permission_fill_in_form": [true],
         "pdf_encrypted": [false],
         "dc_title": ["solr-word"],
         "modified": ["2008-11-13T13:35:51Z"],
         "cp_subject": ["solr word"],
         "pdf_docinfo_subject": ["solr word"],
.
.
.
```

While making the request, we specified `literal.id=doc1`, which tells Solr to use `doc1` as the unique ID for this particular document.

Other parameters that Solr's extracting request handler accepts are covered in this table:



Parameter	Description
`capture`	This captures XHTML elements having specified names for supplementary addition to the document. This parameter is useful to copy chunks of XHTML into a separate field.
`captureAttr`	Indexes attributes of Tika XHTML elements to separate fields, which are named after the element.
`commitWithin`	The time, in milliseconds, to commit the document.
`date.formats`	Defines date format patterns for identification in the documents.
`defaultField`	The default field will be used only when the `uprefix` parameter is unspecified and a field can't be determined.
`extractOnly`	This is false by default. If the value is true, it returns the extracted content from Tika, with no need to index the document.
`extractFormat`	The extraction format to be used, the default being XML. We can change it to text if needed.
`fmap.source_field`	Used to map one field name to another.
`ignoreTikaException`	This is used to ignore exceptions during processing.
`literal.fieldname`	Whatever value is specified in literal.fieldname is used for populating the field with this particular name. If the field is multivalued, then the data can also be multivalued.



literalsOverride	Literal field values will override other values having the same field name if set to true; otherwise, literal values that are defined with literal.fieldname will be added at the end.
lowernames	By setting this to true, the entire set of field names will be mapped to lowercase letters with underscores.
multipartUploadLimitInKB	Used to set a limit on the document size to be uploaded.
passwordsFile	The file path to password mappings will be set here.
resource.name	Used to specify the optional name of the file.
resource.password	The password for the PDF (which is password protected) is defined using resource.password .
tika.config	The file path used to specify Tika's configuration file.
uprefix	Used to prefix fields that have not been defined in the schema with the given prefix.
xpath	Used to filter based on the XPath expression during extraction from Tika XHTML content.

Language detection

Solr uses the langid `UpdateRequestProcessor` to identify languages and then map from text to the language-specific field while indexing.

There are two implementations provided by Solr for language detection:

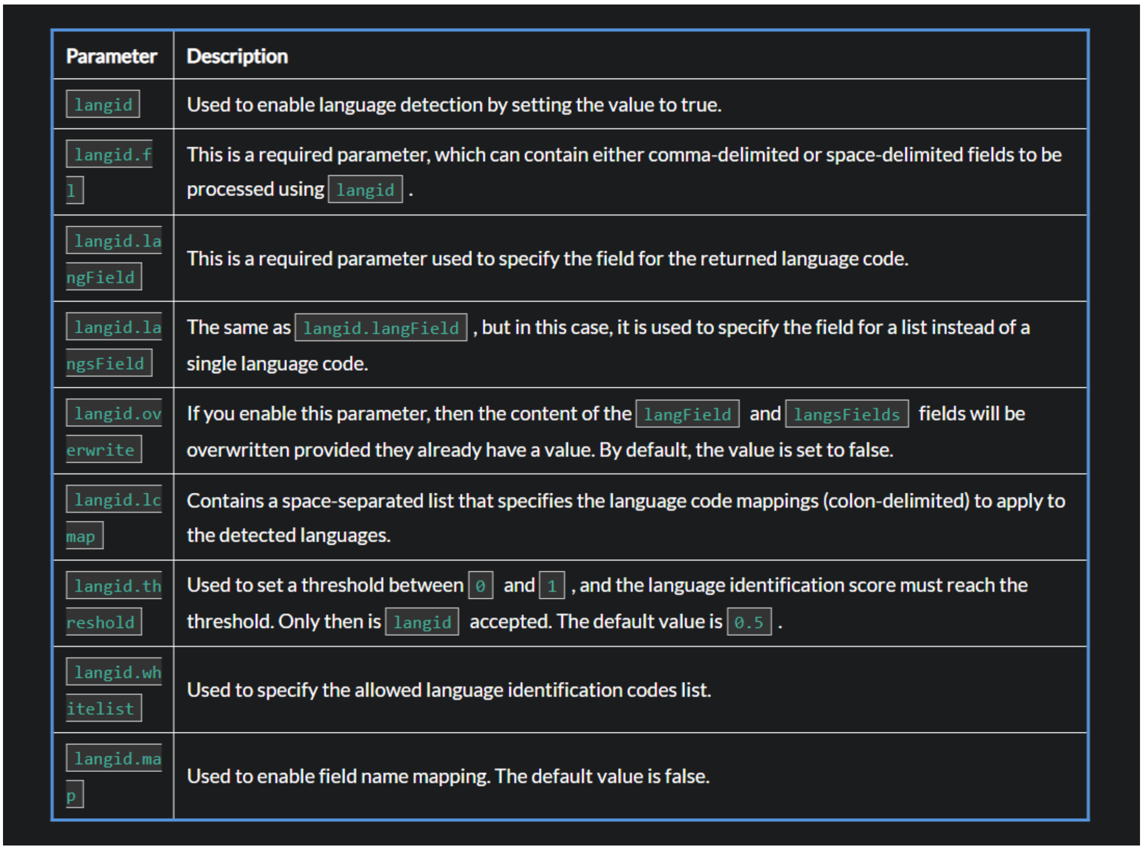
- Tika language detection
- Langdetect language detection

Language detection configuration

The configuration for language detection is done in `solrconfig.xml` and both Tika as well as langdetect language detection use the same parameters, as follows:

```
<processor
class="org.apache.solr.update.processor.TikaLanguageIdentifierUpdateProcessorFactory">
    <lst name="defaults">
        <str name="langid.fl">title,subject,text,keywords</str>
        <str name="langid.langField">language_s</str>
    </lst>
</processor>
<processor class=
"org.apache.solr.update.processor.LangDetectLanguageIdentifierUpdateProcessorFactory">
    <lst name="defaults">
        <str name="langid.fl">title,subject,text,keywords</str>
        <str name="langid.langField">language_s</str>
    </lst>
</processor>
```

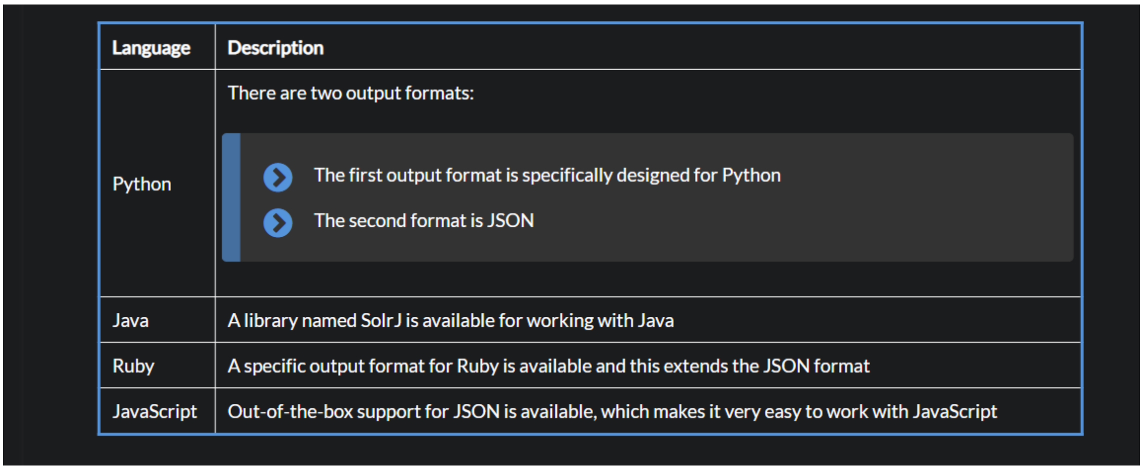
As you can see, both the configurations use the same parameters, the only difference being the `processor` class. The list of parameters is given here:



Parameter	Description
`langid`	Used to enable language detection by setting the value to true.
`langid.fl`	This is a required parameter, which can contain either comma-delimited or space-delimited fields to be processed using `langid` .
`langid.langField`	This is a required parameter used to specify the field for the returned language code.
`langid.langsField`	The same as `langid.langField` , but in this case, it is used to specify the field for a list instead of a single language code.
`langid.overwrite`	If you enable this parameter, then the content of the `langField` and `langsFields` fields will be overwritten provided they already have a value. By default, the value is set to false.
`langid.lcmap`	Contains a space-separated list that specifies the language code mappings (colon-delimited) to apply to the detected languages.
`langid.threshold`	Used to set a threshold between `0` and `1` , and the language identification score must reach the threshold. Only then is `langid` accepted. The default value is `0.5` .
`langid.whitelist`	Used to specify the allowed language identification codes list.
`langid.map`	Used to enable field name mapping. The default value is false.

Client APIs

There are various client APIs available to add data to Solr indexes. This table shows the available client APIs:



Language	Description
Python	There are two output formats: The first output format is specifically designed for Python The second format is JSON
Java	A library named SolrJ is available for working with Java
Ruby	A specific output format for Ruby is available and this extends the JSON format
JavaScript	Out-of-the-box support for JSON is available, which makes it very easy to work with JavaScript

More details on the client API will be covered in a later lab.

Summary

In this lab, we saw various techniques to index data. We went through index handlers and how they help us in indexing data using XML and JSON formats. We made use of the Solr Cell framework to index binary data formats.

We then saw how language detection works. We finally touched on various client APIs available for indexing, though this will be covered in detail in a later lab.

In the next lab, we will see in detail how searching works in Solr. We will cover faceting, spell checking, highlighting, ranking, pagination, and many other features related to searches.