Apache Solr Essentials

Search is everywhere. Users always expect a search facility in mobile or web applications that allows them to find things in a fast and friendly manner.

Apache Solr Essentials is a fast-paced guide to help you quickly learn the process of creating a scalable, efficient, and powerful search application. The book starts off by explaining the fundamentals of Solr and then goes on to cover various topics such as data indexing, ways of extending Solr, client APIs and their indexing and data searching capabilities, an introduction to the administration, monitoring, and tuning of a Solr instance, as well as the concepts of sharding and replication. Next, you’ll learn about various Solr extensions and how to contribute to the Solr community. By the end of this book, you will be able to create excellent search applications with the help of Solr.

Who this book is written for

If you are a competent developer with experience of working with technologies similar to Apache Solr and want to develop efficient search applications, then this book is for you. Familiarity with the Java programming language is required.

What you will learn from this book

- Index your data using formats such as XML, JSON, and CSV
- Manage, monitor, and tune a Solr instance
- Deploy Apache Solr in different environments, depending upon your project requirements
- Refine your search with various Solr client APIs
- Create custom components by leveraging the Apache Solr extension points
- Understand and utilize replication and sharding methods in a distributed search system
- Create and customize your own Solr instance for your project

Leverage the power of Apache Solr to create efficient search applications
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Get Me Up and Running'
- A synopsis of the book’s content
- More information on **Apache Solr Essentials**

---

**About the Author**

**Andrea Gazzarini** is a software engineer. He has mainly focused on the Java technology. Although often involved in analysis and design, he strongly loves coding and definitely likes to be considered a developer.

Andrea has more than 15 years of experience in various software branches, from telecom to banking software. He has worked for several medium- and large-scale companies, such as IBM and Orga Systems.

Andrea has several certifications in the Java programming language (programmer, developer, web component developer, business component developer, and JEE architect), BEA products (build and portal solutions), and Apache Solr (Lucid Apache Solr/Lucene Certified Developer).

In 2009, Andrea stepped into the wonderful world of open source projects, and in the same year, he became a committer for the Apache Qpid project. His adventure with Solr began in 2010, when he joined @Cult, an Italian company that mainly focuses its projects on library management systems, online access public catalogs, and linked data.

He's currently involved in several (too many!) projects, always thinking about a "big" idea that will change his (developer) life.
Acknowledgments

I'd like to begin by thanking the people who made this book what it is. Writing a book is not a single person's work, and help from experienced people that guide you along the path is crucial. Many thanks to Larissa, Kirti, Ankur, and Vikrant for supporting me in this process.

I am also grateful to the technical reviewers of the book, Ahmad Maher Abdelwhab, Markus Kløse, Puneet Singh Ludu, and Julian Lam, for carefully reading my drafts and spotting (hopefully) most of my mistakes. This book would not have been so good without their help and input.

In general, I want to thank everyone who directly or indirectly helped me in creating this book, except for a long-sighted teacher who once told me when I was in university, "Hey, guy with all those earrings! You won't go anywhere!"

Finally, a special thought to my family; to my girls, the actual supporters of the book; my wonderful wife, Nicoletta (to whom I promise not to write another book), my pride and joy, Sofia and Caterina, and my first actual teacher—my mom, Lina. They are the people who really made sacrifices while I was writing and who definitely deserve the credits for the book.

Once again, thank you!
Apache Solr Essentials

As you may have guessed from the title, this is a book about Apache Solr—specifically about Solr essentials. What do I mean by essentials? Nice question! Such a term can be seen from so many perspectives. Solr, mainly from 2010 onwards, witnessed exponential growth in terms of popularity, stakeholders, community, and the capabilities it offers. This rapid growth reflects the rich portfolio of the things that have been developed in these years and are nowadays available. So, strictly speaking, it's not so easy to define the "essentials" of Solr.

The perspective that I will use to explain the term "essentials" is quite simple and pragmatic. I will describe the building blocks of Apache Solr, and at the same time, I will try to put my personal experience on those topics. In recent years, I've worked with Solr in several projects. As a user, I had to learn how to install, configure, tune, troubleshoot, and monitor Solr. As a developer, things were different for me. If you're working in the IT domain and you're reading this book (I guess you are), you probably know that each time you try to implement a solution, there's something in the project that a specific tool doesn't cover. So, after spending a lot of time analyzing, reading documentation, searching on the Internet, reading Wikis, and so on, you realize that you need to add a custom piece of code somewhere. That's because "the product covers the 99.9999 percent of the possible scenarios but..." For this specific case, if this happens or that happens, you always fall under that 0.0001 percent. I don't know about you, but for me, this has always been so. No matter what the project, the company, or the team is, this has been an implicit constant of every project, always.

That's the reason I will try as much as possible to explain things throughout the book using real-world examples directly coming from my personal experience. I hope this additional perspective will be useful for better understanding of what is considered the most popular open source search platform.
What This Book Covers

Chapter 1, *Get Me Up and Running*, introduces the basic concepts of Solr and it provides you with all the necessary steps to quickly get it up and running.

Chapter 2, *Indexing Your Data*, begins our first detailed discussion on Solr. In this chapter, we look at the data indexing process and see how it can be configured, tuned, and customized. This is also where we encounter the first line of code.

Chapter 3, *Searching Your Data*, explores the other specular side of Solr. First, we stored our data; now we explore all that Solr offers in terms of search services.

Chapter 4, *Client API*, covers client-side usage of Solr libraries, providing a description of the main use cases from a client's perspective.

Chapter 5, *Administering and Tuning Solr*, takes you through the available tools for configuring, managing, and tuning Solr.

Chapter 6, *Deployment Scenarios*, illustrates the various ways in which you can deploy Solr, from a standalone instance to a distributed cluster.

Chapter 7, *Solr Extensions*, describes several available Solr extensions and how they can be useful in solving common concrete use cases.

Chapter 8, *Contributing to Solr*, explains the wonderful world of open source software by illustrating the compounding pieces of the process of participation and contribution.
Get Me Up and Running

This chapter describes how to install Solr and focuses on all the required steps to get a complete study and development environment that will guide us through the book.

Specifically, according to the double perspective previously described, I will illustrate two kinds of installations. The first is the installation of a standalone Solr instance (this is very quick). This is a simple task because the download bundle is preconfigured with all that you need to get your first taste of the product. As a developer, the second perspective is what I really need every day in my ordinary job—a working integrated development environment where I can run and debug Solr with my configurations and customizations, without having to manage an external server. In general, such an environment will have all that I need in one place for developing, debugging, and running unit and integration tests.

By the end of the chapter, you will have a running Solr instance on your machine, a ready-to-use Integrated Development Environment (IDE), and a good understanding of some basic concepts.

This chapter will cover the following topics:

- Installation of a simple, standalone Solr instance from scratch
- Setting up of an Integrated Development Environment
- A quick overview about what we installed
- Troubleshooting
Installing a standalone Solr instance

Solr is available for download as an archive that, once uncompressed, contains a fully working instance within a Jetty servlet engine. So the steps here should be pretty easy.

Prerequisites

In this section, we will describe a couple of prerequisites for the machine where Solr needs to be installed.

First of all, Java 6 or 7 is required: the exact choice depends on which version of Solr you want to install. In general, regardless of the version, make sure you have the latest update of your Java Virtual Machine (JVM). The following table describes the association between the latest Solr and Java versions:

<table>
<thead>
<tr>
<th>Solr version</th>
<th>Java version</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7.x</td>
<td>Java 6 or greater</td>
</tr>
<tr>
<td>4.8.x</td>
<td>Java 7 (update 55) or greater; Java 8 is verified to be compatible</td>
</tr>
<tr>
<td>4.9.x</td>
<td>Java 7 (update 55) or greater; Java 8 is verified to be compatible</td>
</tr>
<tr>
<td>4.10.x</td>
<td>Java 7 (update 55) or greater</td>
</tr>
</tbody>
</table>


Other factors such as CPU, RAM, and disk space strongly depend on what you are going to do with this Solr installation. Nowadays, it shouldn't be hard to have a couple of GB available on your workstation. However, bear in mind that at this moment I'm playing on Solr 4.9.0 installed on a Raspberry PI (its RAM is 512 MB). I gave Solr a maximum heap (-Xmx) of 256 MB, indexed about 500 documents, and executed some queries without any problem. But again, those factors really depend on what you want to do: we could say that, assuming you're using a modern PC for a study instance, hardware resources shouldn't be a problem.
Instead, if you are planning a Solr installation in a test or in a production environment, you can find a useful spreadsheet at https://svn.apache.org/repos/asf/lucene/dev/trunk/dev-tools/size-estimator-lucene-solr.xls.

Although it cannot encompass all the peculiarities of your environment, it is definitely a good starting point for RAM and disk space estimation.

**Downloading the right version**

The latest version of Solr at the time of writing is 4.10.3, but a lot of things we will discuss in the book are valid for previous versions as well.

You might already have Solr somewhere and might not want to redownload another instance, your customer might already have a previous version, or, in general, you might not want the latest version. Therefore, I will try to refer to several versions in the book—from 4.7.x to 4.10.x—as often as possible. Each time a feature is described, I will indicate the version where it appeared first.

The download bundle is usually available as a tgz or zip archive. You can find that at https://lucene.apache.org/solr/downloads.html.

**Setting up and running the server**

Once the Solr bundle has been downloaded, extract it in a folder. We will refer to that folder as `$INSTALL_DIR`. Type the following command to extract the Solr bundle:

```
# tar -xvf $DOWNLOAD_DIR/solr-x.y.z.tar.gz -C $INSTALL_DIR
```

or

```
# unzip $DOWNLOAD_DIR/solr-x.y.z.zip -d $INSTALL_DIR
```

depending on the format of the bundle.
Get Me Up and Running

At the end, you will find a new solr-x.y.z folder in your $INSTALL_DIR folder. This folder will act as a container for all Solr instances you may want to play with. Here is a screenshot of the solr-x.y.z folder on my machine, where you can see I have three Solr versions:

![Screenshot of solr-x.y.z folder]

The solr-x.y.z directory contains Jetty, a fast and small servlet engine, with Solr already deployed inside. So, in order to start Solr, we need to start Jetty. Open a new shell and type the following commands:

```
# cd $INSTALL_DIR/solr-x.y.z/example
# java -jar start.jar
```
You should see a lot of log messages ending with something like this:

```
...  
SocketConnector@0.0.0.0:8983 
...  
searcher Searcher@66b66d7[collection1] main{StandardDirectoryReader(segments_2:3:nrt _0(4.9):C32)}
```

These messages tell you Solr is up-and-running! Open a web browser and type http://127.0.0.1:8983/solr.

You should see the following page:

![Solr admin console](image)

This is the Solr administration console.
Setting up a Solr development environment

This section will guide you through the necessary steps to have a working development environment that allows you to have a place to write and execute your code or configurations against a running and debuggable Solr instance.

If you aren't interested in such a perspective because, for instance, your usage scenario falls within the previous section, you can safely skip this and proceed with the next section.

The source code included with this book contains a ready-to-use project for this section. I will later explain how to get it into your workspace in one shot.

Prerequisites

The development workstation needs to have some software. As you can see, I kept the list small and minimal.

Firstly, you need the Java Development Kit 7 (JDK), of which I recommend the latest update, although the older version of Solr covered by this book (4.7.x) is able to run with Java 6. Java 7 is supported from 4.7.x to 4.10.x, so it is definitely a recommended choice.

Lastly, we need an IDE. Specifically, I will use Eclipse to illustrate and describe the developer perspective, so you should download a recent JSE version (that is, Eclipse IDE for Java Developers) from https://www.eclipse.org/downloads.

Do not download the EE version of Eclipse because it contains a lot of things we don't need in this book.

Starting from Eclipse Juno, all the required plugins are already included. However, if you love an older version of Eclipse (such as Indigo) like I do, then Maven integration for Eclipse—also known as M2Eclipse (M2E)—needs to be installed. You can find this in the Eclipse marketplace (go to Help | Eclipse Marketplace, then search for m2e, and click on the Install button).

Importing the sample project of this chapter

It's time to see some code, in order to touch things with your hands. We will guide you through the necessary steps to have your Eclipse configured with a sample project, where you will be able to start, stop, and debug Solr with your code.
First, you have to import to Eclipse the sample project in your local ch1 folder. I assume you already got the source code from the publisher's website or from Github, as described in the Preface. Open Eclipse, create a new workspace, and go to File | Import | Maven | Existing Maven Projects.

### Downloading the example code

You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

Alternatively, you can also download the examples from GitHub, on https://github.com/agazzarini/apache-solr-essentials. There, you can download the whole content as a zip file from https://github.com/agazzarini/apache-solr-essentials/archive/master.zip or, if you have git installed on your machine, you can clone the repository by issuing the following command:

```bash
# git clone https://github.com/agazzarini/apache-solr-essentials.git <path-to-your-work-dir>
```

Where `<path-to-your-work-dir>` is the destination folder where the project will be cloned.

In the dialog box that appears, select the ch1 folder and click on the Finish button. Eclipse will detect the Maven layout of that folder and will create a new project on your workspace, as illustrated in the following screenshot (Project Explorer view):
Understanding the project structure

The project you've imported is very simple and contains just few lines of code, but it is useful for introducing some common concepts that will guide us through the book (the other chapters use examples with a similar structure).

The following table shows the structure of the project:

<table>
<thead>
<tr>
<th>Folder or File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/main/java</td>
<td>The main source folder. It is empty at the moment, but it will contain the Solr extensions (and dependent classes) you want to implement. You won't find this directory in this first project because we don't have the source files yet.</td>
</tr>
<tr>
<td>src/main/resources</td>
<td>This contains project resources such as properties and configuration files. You won't find this directory in this first project because we don't have any resources yet.</td>
</tr>
<tr>
<td>src/test/java</td>
<td>This source folder contains Unit and Integration tests. For this first project, you will find a single integration test here.</td>
</tr>
<tr>
<td>src/test/resources</td>
<td>This contains test resources such as properties and configuration files. It includes a sample logging configuration (log4j.xml).</td>
</tr>
<tr>
<td>src/dev/eclipse</td>
<td>Preconfigured Eclipse launchers used to run Solr and the examples in the project.</td>
</tr>
<tr>
<td>src/solr-home</td>
<td>This contains the Solr configuration files. We will describe the content of this directory later.</td>
</tr>
<tr>
<td>pom.xml</td>
<td>This is the Maven Project definition. Here, you can configure any feature of your project, including dependencies, properties, and so on.</td>
</tr>
</tbody>
</table>
Within the Maven project definition (that is, pom.xml), you can do a lot of things. For our purposes right now, it is important to underline the plugin section, where you can see the Maven Cargo Plugin (http://cargo.codehaus.org/Maven2+plugin) configured to run an embedded Jetty 7 container and deploy Solr. Here's a screenshot that shows the Cargo Plugin configuration section:

```xml
<plugIn>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <version>1.4.9</version>
  <configuration>
    <container>
      <systemProperties>
        <cargo.servlet.port>8083</cargo.servlet.port>
        <solr.solr.home>${basedir}/src/solr-home</solr.solr.home>
        <solr.data.dir>${project.build.directory}/solr/solr.data.dir>
        <log4j.configuration>file:${basedir}/src/test/resources/log4j.xml</log4j.configuration>
      </systemProperties>
      <containerId>jetty7x</containerId>
      <type>embedded</type>
      <dependencies/>
    </container>
    <deployables>
      <deployable>
        <groupId>org.apache.solr</groupId>
        <artifactId>solr</artifactId>
        <type>war</type>
        <properties>
          <context>/solr</context>
        </properties>
      </deployable>
    </deployables>
    <wait>false</wait>
  </configuration>
</plugIn>
```

If you have the Build automatically flag set (the default behavior in Eclipse), most probably Eclipse has already downloaded all the required dependencies. This is one of the great things about Apache Maven.

So, assuming that you have no errors, it's now time to start Solr. But where is Solr?

The first question that probably comes to mind is: "I didn't download Solr! Where is it?" The answer is still Apache Maven, which is definitely a great open source tool for software management and something that simplifies your life.
Maven is already included in your Eclipse (by means of the m2e plugin), and the project you previously imported is a fully compliant Maven project.

So don't worry! When we start a Maven build, Solr will be downloaded automatically. But where? In your local Maven repository, and you don't need to concern yourself with that.

Within the pom.xml file, you will find a property, <solr.version>, with a specific value. If you want to use a different version, just change the value of this property.

Different ways to run Solr

It's time to start Solr in your IDE for the first time but, prior to that, it's important to distinguish the two ways to run Solr:

- **Background server**: As a background server, so that you can start and stop Solr for debugging purposes
- **Integration test server**: As an integration test server so that you can have a dedicated Solr instance to run your integration tests suite

Background server

The first thing you will need in your IDE is a server instance that you can start, stop, and (in general) manage with a few simple commands.

In this way, you will be able to have Solr running with your configurations. You can index your data and execute queries in order to (manually) ensure that things are working as expected.

To get this type of server, follow these instructions:

1. Right-click on the project and create a new Maven (Debug) launch configuration (Debug As | Maven build...).
2. In the dialog, type cargo:run in the Goals text field.
3. Next, click on the **Debug** button as shown in the following screenshot:

![Debug Configuration](image)

The very first time you run this command, Maven will download all the required dependencies and plugins, including Solr. At the end, it will start an embedded Jetty instance.

**Why a Debug instead of a Run configuration?**

You must use a **Debug** configuration so that you will be able to stop the server by simply pressing the red button on the Eclipse console. **Run** configurations have an annoying habit: Eclipse will say the process is stopped, but Jetty will be still running, often leaving an orphan process.
Get Me Up and Running

You should see the following output in the Eclipse console:

```
[INFO] ------------------------------------------------------------
[INFO] Building Chapter 1 Project 1.0
[INFO] ----------------------------------------------------------
Downloading: http://repo1.maven.org/maven2/org/apache/solr/solr/4.9.0/
solr-4.9.0.war
Downloaded: http://repo1.maven.org/maven2/org/apache/solr/solr/4.8.0/
solr-4.9.0.war (28585 KB at 432.5 KB/sec)
...
[INFO] Jetty 7.6.15.v20140411 Embedded started on port [8983]
```

This means that Solr is up and running and it is listening on port 8983. Now open your web browser and type `http://127.0.0.1:8983/solr`. You should see the Solr administration console.

Integration test server

Another important thing you could (or should, in my opinion) do in your project is to have an integration test suite. Integration tests are classes that, as the name suggests, run verifications against a running server.

When you're working on a project with Solr and you want to implement an extension, a search component, or a plugin, you will obviously want to ensure that it is working properly. If you're running an external Solr server, you need to pack your classes in a jar, copy that bundle somewhere (later, we will see where), start the server, and execute your checks.

There are a lot of drawbacks with this approach. Each time you get something wrong, you need to repeat the whole process: fix, pack, copy, restart the server, prepare your data, and run the check again. Also, you cannot easily debug your classes (or Solr classes) during that iterative check. All of this will most probably end with a lot of statements in your code as follows:

```
System.out.println("BLABLABLA");
```

I suppose you know what I'm talking about.
This is where integration tests become very helpful. You can code your checks and your assertions as normal Java classes, and have an automated test suite that does the following each time it is executed:

- Starts an embedded Solr instance
- Executes your tests against that instance
- Stops the Solr instance
- Produces useful reports

The project we set up previously has that capability already, and there's a very basic integration test in the `src/test/java` folder to simply add and query some data.

In order to run the integration test suite, create a new Maven run configuration (right-click on the project and go to Run As | Maven build...), and, in the dialog box, type `clean install` in the Goals text field:
After clicking on the **Run** button, you should see something like this:

...  
**[INFO]**  Jetty 7.6.15.v20140411 Embedded starting...  
...  
**[INFO]**  Reading Solr Schema from schema.xml  
...  
**[INFO]**  Jetty 7.6.15.v20140411 Embedded started on port [8983]  
...  
-------------------------------------------------------

**TESTS**

-------------------------------------------------------

Running `org.gazzax.labs.solr.ase.ch1.it.FirstQueryITCase`

...  
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

As before, under the `src/dev/eclipse` folder, there is already a preconfigured Eclipse launcher for this scenario. Right-click on the `start-embedded-solr.launch` file and go to **Debug As** | **run-the-example-as-integration-test**.

From the Eclipse log, you can see that a test (specifically, an integration test) has been successfully executed. You can find the source code of that test in the project we checked out before. The name of the class that is reported in the log is `FirstQueryITCase` (IT stands for Integration Test), and it is in the `org.gazzax.labs.solr.ase.ch1.it` package.

The `FirstQueryITCase.java` class demonstrates a basic interaction flow we can have with Solr:

```java
// This is the (input) Data Transfer Object between your client and SOLR.
final SolrInputDocument input = new SolrInputDocument();

// 1. Populates with (at least required) fields
input.setField("id", 1);
input.setField("title", "Apache SOLR Essentials");
input.setField("author", "Andrea Gazzarini");
```

---

[20]
input.setField("isbn", "972-2-5A619-12A-X");

// 2. Adds the document
client.add(input);

// 3. Commit changes
client.commit();

// 4. Builds a new query object with a "select all" query.
final SolrQuery query = new SolrQuery("*:*");

// 5. Executes the query
final QueryResponse response = client.query(query);

// 6. Gets the (output) Data Transfer Object.
final SolrDocument output = response.getResults().iterator().next();

final String id = (String) output.getFieldValue("id");
final String title = (String) output.getFieldValue("title");
final String author = (String) output.getFieldValue("author");
final String isbn = (String) output.getFieldValue("isbn");

// 7.1 In case we are running as a Java application print out the query results.
System.out.println("It works! I found the following book: ");
System.out.println("--------------------------------------");
System.out.println("ID: " + id);
System.out.println("Title: " + title);
System.out.println("Author: " + author);
System.out.println("ISBN: " + isbn);

// 7. Otherwise asserts the query results using standard JUnit procedures.
assertEquals("1", id);
assertEquals("Apache SOLR Essentials", title);
assertEquals("Andrea Gazzarini", author);
assertEquals("972-2-5A619-12A-X", isbn);

FirstQueryITCase is an integration test and a main class at the same time. This means that you can run it in three ways: as described earlier, as a main class, and as a JUnit test. If you prefer the second or the third option, remember to start Solr before (using the run-ch1-example-server.launch). You can find the launchers under the src/dev/eclipse folder. Just right-click on one of them and run the example in one way or an other.
What do we have installed?

Regardless of the kind of installation, you should now have a Solr instance up and running, so it's time to have a quick overview of its structure.

Solr is a standard JEE web application, packaged as a .war archive. If you downloaded the bundle from the website, you can find it under the webapps folder of Jetty, usually under:

```
$INSTALL_DIR/solr-x.y.z/example/webapps
```

Instead, if you followed the developer way, Maven downloaded that war file for you, and it is now in your local repository (usually a folder called .m2 under your home directory).

Solr home

In any case, Solr has been installed and you don't need to concern yourself with where it is physically located, mainly because all that you have to provide to Solr must reside in an external folder, usually referred to as the **Solr home**.

In the download bundle, there's a preconfigured Solr home folder that corresponds to the `$INSTALL_DIR/solr-x.y.z/example/solr` folder. Within your Eclipse project, you can find that under the src folder; it is called (not surprisingly) `solr-home`.

In a Solr home folder, you will typically find a file called `solr.xml`, and one or more folders that correspond to your Solr cores (we will see what a core is, in *Chapter 2, Indexing Your Data*). Each folder has a subfolder called conf where the configuration for that specific core resides.

solr.xml

The first file you will find within the Solr home directory is `solr.xml`. It declares some configuration parameters about the instance.

Previously (in Solr 4.4), you had to declare all the cores of your instance in this file. Now there's a more intelligent autodiscovery mechanism that helps you avoid explicit declarations about the cores that are part of your configuration.

In the download bundle, you will find an example of a Solr home with only one core:

```
$INSTALL_DIR/solr-x.y.z/example/solr
```
There is also an example with two cores:

```
$INSTALL_DIR/solr-x.y.z/example/multicore
```

This directory is built using the old style we mentioned previously, with all the cores explicitly declared. In the Eclipse project, you can find the single core example in a directory called `solr-home`. The multicore example is in the `example-solr-home-with-multicore` folder.

**schema.xml**

Although the `schema.xml` file will be described in detail later, it is important to briefly mention it because this is the place where you can declare how your index (of a specific core) is composed, in terms of fields, types, and analysis, both at index time and query time. In other words, this is the schema of your index and (most probably) the first thing you have to design as part of your Solr project.

In the download bundle you can find the `schema.xml` sample under the `$INSTALL_DIR/solr-x.y.z/example/solr/collection1/conf` folder, which is huge and full of comments. It basically illustrates all the predefined fields and types you can use in Solr (you can create your own type, but that's definitely an advanced topic).

If you want to see something simpler for now, the Eclipse project under the `solr-home/conf` directory has a very simple schema, with a few fields and only one field type.

**solrconfig.xml**

The `solrconfig.xml` file is where the configuration of a Solr core is defined. It can contain a lot of directives and sections but, fortunately for most of them, Solr's creators have set default values to be automatically applied if you don't declare them.

Default values are good for a lot of scenarios. When I was in Barcelona at the Apache Lucene Eurocon in 2011, the speaker asked during a presentation, "How many of you have ever changed default values in solrconfig.xml?" In a large room (200 people), only five or six guys raised their hands.

This is most probably the second file you will have to configure. Once the schema has been defined, you can fine-tune the index chain and search behavior of your Solr instance here.
Other resources
Schema and Solr configurations can make use of other files for several purposes. Think about stop words, synonyms, or other configuration files specific to some component. Those files are usually put in the conf directory of the Solr core.

Troubleshooting
If you have problems related to what we described previously, the following tips should help you get things working.

UnsupportedClassVersionError
You can install more than one version of Java on your machine but, when running a command (for example, java or javac), the system will pick up the java interpreter/compiler that is declared in your path. So if you get the UnsupportedClassVersionError error, it means that you're using a wrong JVM (most probably Java 6 or older). In the Prerequisites section earlier in this chapter, there's a table that will help you. However, this is the short version: Solr 4.7.x allows Java 6 or 7, but Solr 4.8 or greater runs only with (at least) Java 7.

If you're starting Solr from the command line, just type this:

# java -version

The output of this command will show the version of Java your system is actually using. So make sure you're running the right JVM, and also check your JAVA_HOME environment variable; it must point to the right JVM.

If you're running Solr in Eclipse, after checking what is described previously (that is, the JVM that starts Eclipse), make sure you're using a correct JVM by navigating to Window | Preferences | Java | Installed JREs.
The "Failed to read artifact descriptor" message

When running a command for the first time (for example, clean, install, or test), Apache Maven will have to download all the required libraries. In order to do that, your system must have a valid Internet connection.

So if you get this kind of message, it means that Maven wasn't able to download a required dependency. The name of the dependency should be in the message. The reason for failure could be a network issue, either permanent or transient.

In the first case, you should simply check your connection. In the second scenario (that is, a transient network failure during the download), there are some manual steps that need to be done. Assume that the dependency is `org.apache.solr:solr-solrj:jar:4.8.0`. You should go to your local Maven repository and remove the content of the folder that hosts that dependency, like this:

```bash
# rm -rf $HOME/.m2/repository/org/apache/solr/solr-solrj/4.8.0
```

On the next build, Maven will download that dependency again.

Summary

In this chapter, we began our Solr tour with a quick overview, including the steps that must be performed when installing Solr. We illustrated the installation process from both a user's and a developer's perspective. Regardless of the path you followed, you should have a working Solr installed on your machine.

In the next chapter, we will continue our conversation by digging further into the Solr indexing process.
Where to buy this book

You can buy Apache Solr Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.