Apache Solr Enterprise Search Server
Third Edition

Solr is a widely popular open source enterprise search server that delivers powerful search and faceted navigation features—features that are elusive with databases. Solr supports complex search criteria, faceting, result highlighting, query-completion, query spell-checking, relevancy tuning, geospatial searches, and much more.

This book is a comprehensive resource for just about everything Solr has to offer, and it will take you from first exposure to development and deployment in no time. Even if you wish to use Solr 5, you should find the information to be just as applicable due to Solr’s high regard for backward compatibility. The book includes some useful information specific to Solr 5.

Who this book is written for

This book is for developers who want to learn how to get the most out of Solr in their applications, whether you are new to the field, have used Solr but don’t know everything, or simply want a good reference. It would be helpful to have some familiarity with basic programming concepts, but no prior experience is required.

What you will learn from this book

- Design a schema to include text indexing details such as tokenization, stemming, and synonyms
- Import data from databases using various formats including CSV and XML and extract text from different document formats
- Search using Solr’s rich query syntax, perform geospatial searches, “join” relationally, and influence relevancy order
- Build a query auto-complete/suggester capability with knowledge of the fundamental types of suggestion and ways to implement them
- Enhance standard searches with faceting for navigation or analytics
- Deploy Solr to production taking into account logging, security, and monitoring
- Integrate a host of technologies with Solr including web crawlers, Hadoop, Java, JavaScript, Ruby, PHP, Drupal, and others
- Tune Solr and use SolrCloud for horizontal scalability


Free Sample
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'Quick Starting Solr'
- A synopsis of the book’s content
- More information on Apache Solr Enterprise Search Server Third Edition

About the Authors

Born to code, **David Smiley** is a software engineer who's passionate about search, Lucene, spatial, and open source. He has a great deal of expertise with Lucene and Solr, which started in 2008 at MITRE. In 2009, as the lead author, along with the coauthor Eric Pugh, he wrote *Solr 1.4 Enterprise Search Server*, the first book on Solr, published by Packt Publishing. It was updated in 2011, *Apache Solr 3 Enterprise Search Server, Packt Publishing*, and again for this third edition.

After the first book, he developed 1- and 2-day Solr training courses, delivered half a dozen times within MITRE, and he has also delivered training on LucidWorks once. Most of his excitement and energy relating to Lucene is centered on Lucene's spatial module to include Spatial4j, which he is largely responsible for. He has presented his progress on this at Lucene Revolution and other conferences several times. He currently holds the status of committer & Project Management Committee (PMC) member with the Lucene/Solr open source project. Over the years, David has staked his career on search, working exclusively on such projects, formerly for MITRE and now as an independent consultant for various clients. You can reach him at dsmiley@apache.org and view his LinkedIn profile here: [http://www.linkedin.com/in/davidwsmiley](http://www.linkedin.com/in/davidwsmiley).
Fascinated by the "craft" of software development, **Eric Pugh** has been involved in the open source world as a developer, committer, and user for the past decade. He is an emeritus member of the Apache Software Foundation.

In biotech, financial services, and defense IT, he has helped European and American companies develop coherent strategies to embrace open source software. As a speaker, he has advocated the advantages of Agile practices in search, discovery, and analytics projects.

Eric became involved in Solr when he submitted the patch SOLR-284 to parse rich document types, such as PDF and MS Office formats, that became the single-most popular patch, as measured by votes! The patch was subsequently cleaned up and enhanced by three other individuals, demonstrating the power of the free / open source models to build great code collaboratively. SOLR-284 was eventually refactored into Solr Cell.


---

**Kranti Parisa** has more than a decade of software development expertise and a deep understanding of open source, enterprise software, and the execution required to build successful products.

He has fallen in love with enterprise search technologies, especially Lucene and Solr, after his initial implementations and customizations carried out in early 2008 to build a legal search engine for bankruptcy court documents, docket entries, and cases. He is an active contributor to the Apache Solr community. One of his recent contributions, along with Joel Bernstein, SOLR-4787, includes scalable and nested join implementations.

Kranti is currently working at Apple. Prior to that, he worked as a lead engineer and search architect at Comcast Labs, building and supporting a highly scalable search and discovery engine for the X1/X2 platform—the world's first entertainment operating system.

An entrepreneur by DNA, he is the cofounder and technical advisor of multiple start-ups focusing on cloud computing, SaaS, big data, and enterprise search based products and services. He holds a master's degree in computer integrated manufacturing from the National Institute of Technology, Warangal, India.

You can reach him on LinkedIn: [http://www.linkedin.com/in/krantiparisa](http://www.linkedin.com/in/krantiparisa).
Matt Mitchell studied music synthesis and performance at Boston’s Berklee College of Music, but his experiences with computers and programming in his younger years inspired him to pursue a career in software engineering. A passionate technologist, he has worked in many areas of software development, is active in several open source communities, and now has over 15 years of professional experience. He had his first experiences with Lucene and Solr in 2008 at the University of Virginia Library, where he became a core contributor to an open source search platform called Backlight. Matt is the author of many open source projects, including a Solr client library called RSolr, which has had over 1 million downloads from rubygems.org. He has been responsible for the design and implementation of search systems at several tech companies, and he is currently a senior member of the engineering team at LucidWorks, where he’s working on a next generation search, discovery, and analytics platform.

You can contact Matt on LinkedIn at https://www.linkedin.com/in/mattmitchell4.
If you are a developer building an application today, then you know how important a good search experience is. Apache Solr, built on Apache Lucene, is a wildly popular open source enterprise search server that easily delivers the powerful search and faceted navigation features that are elusive with databases. Solr supports complex search criteria, faceting, result highlighting, query-completion, query spellcheck, relevancy tuning, and more.

*Apache Solr Enterprise Search Server, Third Edition* is a comprehensive resource to almost everything Solr has to offer. It serves the reader right from initiation to development to deployment. It also comes with complete running examples to demonstrate its use and show how to integrate Solr with other languages and frameworks—even Hadoop.

By using a large set of metadata, including artists, releases, and tracks, courtesy of the MusicBrainz.org project, you will have a testing ground for Solr and will learn how to import this data in various ways. You will then learn how to search this data in different ways, including Solr's rich query syntax and boosting match scores based on record data. Finally, we'll cover various deployment considerations to include indexing strategies and performance-oriented configuration that will enable you to scale Solr to meet the needs of a high-volume site.
What This Book Covers

Chapter 1, Quick Starting Solr, introduces Solr to you so that you understand its unique role in your application stack. You'll get started quickly by indexing example data and searching it with Solr's sample / browse UI. This chapter is oriented to Solr 5, but the majority of content applies to Solr 4 too.

Chapter 2, Schema Design, guides you through an approach to modeling your data within Solr into one or more Solr indices and schemas. It covers the schema thoroughly and explores some of Solr's field types.

Chapter 3, Text Analysis, covers how to customize text tokenization, stemming, synonyms, and related matters to have fine control over keyword search matching. It also covers multilingual strategies.

Chapter 4, Indexing Data, explores all of the options Solr offers for importing data, such as XML, CSV, databases (SQL), and text extraction from common documents. This includes important information on commits, atomic updates, and real-time search.

Chapter 5, Searching, covers the query syntax, from the basics to Boolean options to more advanced wildcard and fuzzy searches, join queries, and geospatial search.

Chapter 6, Search Relevancy, explains how Solr scores documents for relevancy ranking. We'll review different options to influence the score, called boosting, and apply it to common examples such as boosting recent documents and boosting by a user vote.

Chapter 7, Faceting, shows you how to use Solr's killer feature—faceting. You'll learn about the different types of facets and how to build filter queries for a faceted navigation interface.

Chapter 8, Search Components, explores how to use a variety of valuable search features implemented as Solr search components. This includes result highlighting, query spellcheck, query suggest / complete, result grouping / collapsing, and more.

Chapter 9, Integrating Solr, explores some external integration options to interface with Solr. This includes some language-specific frameworks for Java, JavaScript, Ruby, and PHP, as well as a web crawler, Hadoop, a quick prototyping option, and more.

Chapter 10, Scaling Solr, covers how to tune Solr to get the most out of it. Then we'll introduce how to scale beyond one instance with SolrCloud.

Chapter 11, Deployment, guides you through deployment considerations to include deploying Solr to Apache Tomcat, to logging, and to security, and setting up Apache ZooKeeper.

Appendix, Quick Reference, serves as a small parameter quick-reference guide you can print to have within reach when you need it.
Quick Starting Solr

Welcome to Solr! You've made an excellent choice to power your search needs. In this chapter, we're going to cover the following topics:

- An overview of what Solr and Lucene are all about
- What makes Solr different from databases?
- How to get Solr, what's included, and what is where?
- Running Solr and importing sample data
- A quick tour of the admin interface and key configuration files
- A brief guide on how to get started quickly

An introduction to Solr

Solr is an open source enterprise search server. It is a mature product powering search for public sites such as CNET, Yelp, Zappos, and Netflix, as well as countless other government and corporate intranet sites. It is written in Java, and that language is used to further extend and modify Solr through various extension points. However, being a server that communicates using standards such as HTTP, XML, and JSON, knowledge of Java is useful but not a requirement. In addition to the standard ability to return a list of search results based on a full text search, Solr has numerous other features such as result highlighting, faceted navigation (as seen on most e-commerce sites), query spellcheck, query completion, and a "more-like-this" feature for finding similar documents.
You will see many references in this book to the term **faceting**, also known as **faceted navigation**. It’s a killer feature of Solr that most people have experienced at major e-commerce sites without realizing it. Faceting enhances search results with aggregated information over all of the documents found in the search. Faceting information is typically used as dynamic navigational filters, such as a product category, date and price groupings, and so on. Faceting can also be used to power analytics. Chapter 7, Faceting, is dedicated to this technology.

**Lucene – the underlying engine**

Before describing Solr, it is best to start with Apache Lucene, the core technology underlying it. Lucene is an open source, high-performance text search engine library. Lucene was developed and open sourced by Doug Cutting in 2000 and has evolved and matured since then with a strong online community. It is the most widely deployed search technology today. Being just a code library, Lucene is not a server and certainly isn’t a web crawler either. This is an important fact. There aren’t even any configuration files.

In order to use Lucene, you write your own search code using its API, starting with indexing documents that you supply to it. A **document** in Lucene is merely a collection of **fields**, which are name-value pairs containing text or numbers. You configure Lucene with a text **analyzer** that will **tokenize** a field’s text from a single string into a series of **tokens** (words) and further transform them by reducing them to their stems, called **stemming**, substitute synonyms, and/or perform other processing. The final indexed tokens are said to be the **terms**. The aforementioned process starting with the analyzer is referred to as **text analysis**. Lucene **indexes** each document into its **index** stored on a disk. The index is an **inverted index**, which means it stores a mapping of a field’s terms to associated documents, along with the ordinal word position from the original text. Finally, you search for documents with a user-provided query string that Lucene parses according to its syntax. Lucene assigns a numeric relevancy **score** to each matching document and only the top scoring documents are returned.

This brief description of Lucene internals is what makes Solr work at its core. You will see these important vocabulary words throughout this book—they will be explained further at appropriate times.
Lucene's major features are:

- An inverted index for efficient retrieval of documents by indexed terms. The same technology supports numeric data with range- and time-based queries too.
- A rich set of chainable text analysis components, such as tokenizers and language-specific stemmers that transform a text string into a series of terms (words).
- A query syntax with a parser and a variety of query types, from a simple term lookup to exotic fuzzy matching.
- A good scoring algorithm based on sound Information Retrieval (IR) principles to produce the best matches first, with flexible means to affect the scoring.
- Search enhancing features. There are many, but here are some notable ones:
  - A highlighter feature to show matching query terms found in context.
  - A query spellchecker based on indexed content or a supplied dictionary.
  - Multiple suggesters for completing query strings.
  - Analysis components for various languages, faceting, spatial-search, and grouping and joining queries too.

To learn more about Lucene, read *Lucene In Action, Second Edition*, Michael McCandless, Erik Hatcher, and Otis Gospodneti, Manning Publications.

**Solr – a Lucene-based search server**

Apache Solr is an enterprise search server that is based on Lucene. Lucene is such a big part of what defines Solr that you'll see many references to Lucene directly throughout this book. Developing a high-performance, feature-rich application that uses Lucene directly is difficult and it's limited to Java applications. Solr solves this by exposing the wealth of power in Lucene via configuration files and HTTP parameters, while adding some features of its own. Some of Solr's most notable features beyond Lucene are as follows:

- A server that communicates over HTTP via multiple formats, including XML and JSON
- Configuration files, most notably for the index's schema, which defines the fields and configuration of their text analysis
Quick Starting Solr

- Several types of caches for faster search responses
- A web-based administrative interface, including the following:
  - Runtime search and cache performance statistics
  - A schema browser with index statistics on each field
  - A diagnostic tool for debugging text analysis
  - Support for dynamic core (indices) administration
- Faceting of search results (note: distinct from Lucene's faceting)
- A query parser called eDisMax that is more usable for parsing end user queries than Lucene's native query parser
- Distributed search support, index replication, and fail-over for scaling Solr
- Cluster configuration and coordination using ZooKeeper
- Solritas—a sample generic web search UI for prototyping and demonstrating many of Solr's search features

Also, there are two contrib modules that ship with Solr that really stand out, which are as follows:

- **DataImportHandler (DIH):** A database, e-mail, and file crawling data import capability. It includes a debugger tool.
- **Solr Cell:** An adapter to the Apache Tika open source project, which can extract text from numerous file types.

As of the 3.1 release, there is a tight relationship between the Solr and Lucene projects. The source code repository, committers, and developer mailing list are the same, and they are released together using the same version number. Since Solr is always based on the latest version of Lucene, most improvements in Lucene are available in Solr immediately.

**Comparison to database technology**

There’s a good chance that you are unfamiliar with Lucene or Solr and you might be wondering what the fundamental differences are between it and a database. You might also wonder if you use Solr, do you need a database.

The most important comparison to make is with respect to the data model—the organizational structure of the data. The most popular category of databases is relational databases—RDBMS. A defining characteristic of relational databases is a data model, based on multiple tables with lookup keys between them and a join capability for querying across them. That approach has proven to be versatile, being able to satisfy nearly any information-retrieval task in one query.
However, it is hard and expensive to scale them to meet the requirements of a typical search application consisting of many millions of documents and low-latency response. Instead, Lucene has a much more limiting document-oriented data model, which is analogous to a single table. Document-oriented databases such as MongoDB are similar in this respect, but their documents can be nested, similar to XML or JSON. Lucene's document structure is flat like a table, but it does support multivalued fields—a field with an array of values. It can also be very sparse such that the actual fields used from one document to the next vary; there is no space or penalty for a document to not use a field.

Lucene and Solr have limited support for join queries, but they are used sparingly as it significantly reduces the scalability characteristics of Lucene and Solr.

Taking a look at the Solr feature list naturally reveals plenty of search-oriented technology that databases generally either don't have, or don't do well. The notable features are relevancy score ordering, result highlighting, query spellcheck, and query-completion. These features are what drew you to Solr, no doubt. And let's not forget faceting. This is possible with a database, but it's hard to figure out how, and it's difficult to scale. Solr, on the other hand, makes it incredibly easy, and it does scale.

Can Solr be a substitute for your database? You can add data to it and get it back out efficiently with indexes; so on the surface, it seems plausible. The answer is that you are almost always better off using Solr in addition to a database. Databases, particularly RDBMSes, generally excel at ACID transactions, insert/update efficiency, in-place schema changes, multiuser access control, bulk data retrieval, and they have second-to-none integration with application software stacks and reporting tools. And let's not forget that they have a versatile data model. Solr falls short in these areas.

For more on this subject, see our article, Text Search, your Database or Solr, at http://bit.ly/uwF1ps, which although it's slightly outdated now, is a clear and useful explanation of the issues. If you want to use Solr as a document-oriented or key-value NoSQL database, Chapter 4, Indexing Data, will tell you how and when it's appropriate.
A few differences between Solr 4 and Solr 5

The biggest change that users will see in Solr 5 from Solr 4 is that Solr is now deployed as its own server process. It is no longer a WAR file that is deployed into an existing Servlet container such as Tomcat or Jetty. The argument for this boiled down to "you don't deploy your MySQL database in a Servlet container; neither should you deploy your Search engine". By owning the network stack and deployment model, Solr can evolve faster; for example, there are patches for adding HTTP/2 support and pluggable authentication mechanisms being worked on. While internally Solr is still using Jetty, that should be considered an implementation detail. That said, if you really want a WAR file version, and you're familiar with Java and previous Solr releases, you can probably figure out how to build one.

As part of Solr 5 being it's own server process, it includes a set of scripts for starting, stopping, and managing Solr collections, as well as running as a service on Linux.

The next most obvious difference is that the distribution directory structure is different, particularly related to the old `example` and new `server` directory.

The rest of this chapter refers to Solr 5, however the remainder of the book was updated for Solr 4, and applies to Solr 5.

Getting started

We will get started by downloading Solr, examining its directory structure, and then finally run it.

This will set you up for the next section, which tours a running Solr 5 server.

1. **Get Solr:** You can download Solr from its website [http://lucene.apache.org/solr/](http://lucene.apache.org/solr/). This book assumes that you downloaded one of the binary releases (not the src (source) based distribution). In general, we recommend using the latest release since Solr and Lucene’s code are extensively tested. For downloadable example source code, and book errata describing how future Solr releases affect the book content, visit our website [http://www.solrenterprisesearchserver.com/](http://www.solrenterprisesearchserver.com/).
2. **Get Java:** The only prerequisite software needed to run Solr is Java 7 (that is, Java Version 1.7). But the latest version is Java 8, and you should use that. Typing `java –version` at a command line will tell you exactly which version of Java you are using, if any.

Java is available on all major platforms, including Windows, Solaris, Linux, and Mac OS X. Visit [http://www.java.com](http://www.java.com) to download the distribution for your platform. Java always comes with the **Java Runtime Environment (JRE)** and that’s all Solr requires. The **Java Development Kit (JDK)** includes the JRE plus the Java compiler and various diagnostic utility programs. One such useful program is JConsole, which we’ll discuss in **Chapter 11, Deployment**, and **Chapter 10, Scaling Solr** and so the JDK distribution is recommended.

Solr is a Java-based web application, but you don’t need to be particularly familiar with Java in order to use it. This book assumes no such knowledge on your part.

3. **Get the book supplement:** This book includes a code supplement available at our website [http://www.solrenterprise/searchserver.com](http://www.solrenterprise/searchserver.com); you can also find it on Packt Publishing’s website at [http://www.packtpub.com/books/content/support](http://www.packtpub.com/books/content/support). The software includes a Solr installation configured for data from MusicBrainz.org, a script to download, and indexes that data into Solr—about 8 million documents in total, and of course various sample code and material organized by chapter. This supplement is not required to follow any of the material in the book. It will be useful if you want to experiment with searches using the same data used for the book’s searches or if you want to see the code referenced in a chapter. The majority of the code is for **Chapter 9, Integrating Solr**.

### Solr's installation directory structure

When you unzip Solr after downloading it, you should find a relatively straightforward directory structure (differences between Solr 4 and 5 are briefly explained here):

- **contrib:** The Solr contrib modules are extensions to Solr:
  - **analysis-extras:** This directory includes a few text analysis components that have large dependencies. There are some **International Components for Unicode (ICU)** unicode classes for multilingual support—a Chinese stemmer and a Polish stemmer. You’ll learn more about text analysis in the next chapter.
clustering: This directory will have an engine for clustering search results. There is a one-page overview in Chapter 8, Search Components.

dataimporthandler: The DataImportHandler (DIH) is a very popular contrib module that imports data into Solr from a database and some other sources. See Chapter 4, Indexing Data.

extraction: Integration with Apache Tika—a framework for extracting text from common file formats. This module is also called SolrCell and Tika is also used by the DIH's TikaEntityProcessor—both are discussed in Chapter 4, Indexing Data.

langid: This directory contains a contrib module that provides the ability to detect the language of a document before it's indexed. More information can be found on the Solr’s Language Detection wiki page at http://wiki.apache.org/solr/LanguageDetection.

map-reduce: This directory has utilities for working with Solr from Hadoop Map-Reduce. This is discussed in Chapter 9, Integrating Solr.

morphlines-core: This directory contains Kite Morphlines, a document ingestion framework that has support for Solr. The morphlines-cell directory has components related to text extraction. Morphlines is mentioned in Chapter 9, Integrating Solr.

uima: This directory contains library for Integration with Apache UIMA—a framework for extracting metadata out of text. There are modules that identify proper names in text and identify the language, for example. To learn more, see Solr's UIMA integration wiki at http://wiki.apache.org/solr/SolrUIMA.

velocity: This directory will have a simple search UI framework based on the Velocity templating language. See Chapter 9, Integrating Solr.

dist: In this directory, you will see Solr's core and contrib JAR files. In previous Solr versions, the WAR file was found here as well. The core JAR file is what you would use if you’re embedding Solr within an application. The Solr test framework JAR and /test-framework directory contain the libraries needed in testing Solr extensions. The SolrJ JAR and /solrj-lib are what you need to build Java based clients for Solr.

docs: This directory contains documentation and "Javadocs" for the related assets for the public Solr website, a quick tutorial, and of course Solr's API.
If you are looking for documentation outside of this book, you are best served by the Solr Reference Guide. The docs directory isn't very useful.

- **example**: Pre Solr 5, this was the complete Solr server, meant to be an example layout for deployment. It included the Jetty servlet engine (a Java web server), Solr, some sample data and sample Solr configurations. With the introduction of Solr 5, only the example-DIH and exampledocs are kept, the rest was moved to a new server directory.
  - example/example-DIH: These are DataImportHandler configuration files for the example Solr setup. If you plan on importing with DIH, some of these files may serve as good starting points.
  - example/exampledocs: These are sample documents to be indexed into the default Solr configuration, along with the post.jar program for sending the documents to Solr.

- **server**: The files required to run Solr as a server process are located here. The interesting child directories are as follows:
  - server/contexts: This is Jetty's WebApp configuration for the Solr setup.
  - server/etc: This is Jetty's configuration. Among other things, here you can change the web port used from the presupplied 8983 to 80 (HTTP default).
  - server/logs: Logs are by default output here. Introduced in Solr 5 was collecting JVM metrics, which are output to solr_gc.log. When you are trying to size your Solr setup they are a good source of information.
  - server/resources: The configuration file for Log4j lives here. Edit it to change the behavior of the Solr logging, (though you can also changes levels of debugging at runtime through the Admin console).
  - server/solr: The configuration files for running Solr are stored here. The solr.xml file, which provides overall configuration of Solr lives here, as well as zoo.cfg which is required by SolrCloud. The subdirectory /configsets stores example configurations that ship with Solr.
  - example/webapps: This is where Jetty expects to deploy Solr from. A copy of Solr's WAR file is here, which contains Solr's compiled code and all the dependent JAR files needed to run it.
  - example/solr-webapp: This is where Jetty deploys the unpacked WAR file.
Running Solr

Solr ships with a number of example collection configurations. We're going to run one called *techproducts*. This example will create a collection and insert some sample data.

The addition of scripts for running Solr is one of the best enhancements in Solr 5. Previously, to start Solr, you directly invoked Java via `java -jar start.jar`. Deploying to production meant figuring out how to migrate into an existing Servlet environment, and was the source of much frustration.

First, go to the `bin` directory, and then run the main Solr command. On Windows, it will be `solr.cmd`, on *nix systems it will be just `solr`. Jetty's `start.jar` file by typing the following command:

```bash
>cd bin
>>./solr start -e techproducts
```

The `>>` notation is the command prompt and is not part of the command. You'll see a few lines of output as Solr is started, and then the `techproducts` collection is created via an API call. Then the sample data is loaded into Solr. When it's done, you'll be directed to the Solr admin at `http://localhost:8983/solr`.

To stop Solr, use the same Solr command script:

```bash
>>./solr stop
```

A quick tour of Solr

Point your browser to Solr's administrative interface at `http://localhost:8983/`. The admin site is a single-page application that provides access to some of the more important aspects of a running Solr instance.

The administrative interface is currently being completely revamped, and the below interface may be deprecated.
This tour will help you get your bearings in navigating around Solr.

![Dashboard screenshot]

In the preceding screenshot, the navigation is on the left while the main content is on the right. The left navigation is present on every page of the admin site and is divided into two sections. The primary section contains choices related to higher-level Solr and Java features, while the secondary section lists all of the running Solr cores.

The default page for the admin site is Dashboard. This gives you a snapshot of some basic configuration settings and stats, for Solr, the JVM, and the server. The Dashboard page is divided into the following subareas:

- **Instance**: This area displays when Solr started.
- **Versions**: This area displays various Lucene and Solr version numbers.
- **JVM**: This area displays the Java implementation, version, and processor count. The various Java system properties are also listed here.
- **System**: This area displays the overview of memory settings and usage; this is essential information for debugging and optimizing memory settings.
- **JVM-Memory**: This meter shows the allocation of JVM memory, and is key to understanding if garbage collection is happening properly. If the dark gray band occupies the entire meter, you will see all sorts of memory related exceptions!
The rest of the primary navigation choices include the following:

- **Logging**: This page is a real-time view of logging, showing the time, level, logger, and message. This section also allows you to adjust the logging levels for different parts of Solr at runtime. For Jetty, as we're running it, this output goes to the console and nowhere else. See Chapter 11, Deployment, for more information on configuring logging.

- **Core Admin**: This section is for information and controls for managing Solr cores. Here, you can unload, reload, rename, swap, and optimize the selected core. There is also an option for adding a new core.

- **Java Properties**: This lists Java system properties, which are basically Java-oriented global environment variables. Including the command used to start the Solr Java process.

- **Thread Dump**: This displays a Java thread dump, useful for experienced Java developers in diagnosing problems.

Below the primary navigation is a list of running Solr cores. Click on the Core Selector drop-down menu and select the `techproducts` link. You should see something very similar to the following screenshot:

The default page labeled **Overview** for each core shows core statistics, information about replication, an **Admin Extra** area. Some other options such as details about **Healthcheck** are grayed out and made visible if the feature is enabled.
You probably noticed the subchoice menu that appeared below `techproducts`. Here is an overview of what those subchoices provide:

- **Analysis**: This is used for diagnosing query and indexing problems related to text analysis. This is an advanced screen and will be discussed later.
- **Data Import**: Provides information about the DataImport handler (the DIH). Like replication, it is only useful when DIH is enabled. The DataImport handler will be discussed in more detail in Chapter 4, *Indexing Data*.
- **Documents**: Provides a simple interface for creating a document to index into Solr via the browser. This includes a Document Builder that walks you through adding individual fields of data.
- **Files**: Exposes all the files that make up the core's configuration. Everything from core files such as `schema.xml` and `solrconfig.xml` to `stopwords.txt`.
- **Ping**: Clicking on this sends a ping request to Solr, displaying the latency. The primary purpose of the ping response is to provide a health status to other services, such as a load balancer. The ping response is a formatted status document and it is designed to fail if Solr can't perform a search query that you provide.
- **Plugins / Stats**: Here you will find statistics such as timing and cache hit ratios. In Chapter 10, *Scaling Solr*, we will visit this screen to evaluate Solr's performance.
- **Query**: This brings you to a search form with many options. With or without this search form, you will soon end up directly manipulating the URL using this book as a reference. There's no data in Solr yet, so there's no point in using the form right now.
- **Replication**: This contains index replication status information, and the controls for disabling. It is only useful when replication is enabled. More information on this is available in Chapter 10, *Scaling Solr*.
- **Schema Browser**: This is an analytical view of the schema that reflects various statistics of the actual data in the index. We'll come back to this later.
- **Segments Info**: Segments are the underlying files that make up the Lucene data structure. As you index information, they expand and compress. This allows you to monitor them, and was newly added to Solr 5.

You can partially customize the admin view by editing a few templates that are provided. The template filenames are prefixed with `admin-extra`, and are located in the `conf` directory.
Quick Starting Solr

Loading sample data

Solr comes with some sample data found at example/exampledocs. We saw this data loaded as part of creating the techproducts Solr core when we started Solr. We're going to use that for the remainder of this chapter so that we can explore Solr more, without getting into schema design and deeper data loading options. For the rest of the book, we'll base the examples on the digital supplement to the book — more on that later.

We're going to re-index the example data by using the post.jar Java program, officially called SimplePostTool. Most JAR files aren't executable, but this one is. This simple program takes a Java system variable to specify the collection: -Dc=techproducts, iterates over a list of Solr-formatted XML input files, and HTTP posts it to Solr running on the current machine — http://localhost:8983/solr/techproducts/update. Finally, it will send a commit command, which will cause documents that were posted prior to the commit to be saved and made visible. Obviously, Solr must be running for this to work. Here is the command and its output:

```
>> cd example/exampledocs
>> java -Dc=techproducts -jar post.jar *.xml
SimplePostTool version 5.0.0
update using
content-type application/xml...
POSTing file gb18030-example.xml
POSTing file hd.xml
etc.
14 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/techproducts/
update...
```

If you are using a Unix-like environment, you have an alternate option of using the /bin/post shell script, which wraps the SimplePostTool.

The post.sh and post.jar programs could be used in a production scenario, but they are intended just as a demonstration of the technology with the example data.
Let's take a look at one of these XML files we just posted to Solr, `monitor.xml`:

```
<add>
  <doc>
    <field name="id">3007WFP</field>
    <field name="name">Dell Widescreen UltraSharp 3007WFP</field>
    <field name="manu">Dell, Inc.</field>
    <!-- Join -->
    <field name="manu_id_s">dell</field>
    <field name="cat">electronics</field>
    <field name="cat">monitor</field>
    <field name="features">30" TFT active matrix LCD, 2560 x 1600, .25mm dot pitch, 700:1 contrast</field>
    <field name="includes">USB cable</field>
    <field name="weight">401.6</field>
    <field name="price">2199</field>
    <field name="popularity">6</field>
    <field name="inStock">true</field>
    <!-- Buffalo store -->
    <field name="store">43.17614,-90.57341</field>
  </doc>
</add>
```

The XML schema for files that can be posted to Solr is very simple. This file doesn't demonstrate all of the elements and attributes, but it shows the essentials. Multiple documents, represented by the `<doc>` tag, can be present in series within the `<add>` tag, which is recommended for bulk data loading scenarios. This subset may very well be all that you use. More about these options and other data loading choices will be discussed in Chapter 4, *Indexing Data*.

**A simple query**

Point your browser to `http://localhost:8983/solr/#/techproducts/query`—this is the query form described in the previous section. The search box is labeled `q`. This form is a standard HTML form, albeit enhanced by JavaScript. When the form is submitted, the form inputs become URL parameters to an HTTP GET request to Solr. That URL and Solr's search response is displayed to the right. It is convenient to use the form as a starting point for developing a search, but then subsequently refine the URL directly in the browser instead of returning to the form.
Quick Starting Solr

Run a query by replacing the *:* in the q field with the word lcd, then clicking on the Execute Query button. At the top of the main content area, you will see a URL like this: http://localhost:8983/solr/techproducts/select?q=monitor&wt=json&indent=true. The URL specifies that you want to query for the word lcd, and that the output should be in indented JSON format.

Below this URL, you will see the search result; this result is the response of that URL.

By default, Solr responds in XML, however the query interface specifies JSON by default. Most modern browsers, such as Firefox, provide a good JSON view with syntax coloring and hierarchical controls. All response formats have the same basic structure as the JSON you’re about to see. More information on these formats can be found in Chapter 4, Indexing Data.

The JSON response consists of a two main elements: responseHeader and response. Here is what the header element looks like:

```
"responseHeader": {  
  "status": 0,  
  "QTime": 1,  
  "params": {    
    "q": "lcd",    
    "indent": "true",    
    "wt": "json"  
  }  
}
```

The following are the elements from the preceding code snippet:

- **status**: This is always zero, unless there was a serious problem.
- **QTime**: This is the duration of time in milliseconds that Solr took to process the search. It does not include streaming back the response. Due to multiple layers of caching, you will find that your searches will often complete in a millisecond or less if you’ve run the query before.
- **params**: This lists the request parameters. By default, it only lists parameters explicitly in the URL; there are usually more parameters specified in a <requestHandler/> in solrconfig.xml. You can see all of the applied parameters in the response by setting the echoParams parameter to true.

More information on these parameters and many more is available in Chapter 5, Searching.
Next up is the most important part, the results:

```json
"response": {
    "numFound": 5,
    "start": 0,
}
```

The `numFound` value is the number of documents matching the query in the entire index. The `start` parameter is the index offset into those matching (ordered) documents that are returned in the response below.

Often, you'll want to see the score of each matching document. The document score is a number that represents how relevant the document is to the search query. This search response doesn't refer to scores because it needs to be explicitly requested in the `fl` parameter—a comma-separated field list. A search that requests the score via `fl=*`, `score` will have a `maxScore` attribute in the "response" element, which is the maximum score of all documents that matched the search. It's independent of the sort order or result paging parameters.

The content of the result element is a list of documents that matched the query. The default sort is by descending score. Later, we'll do some sorting by specified fields.

```json
{
    "id": "9885A004",
    "name": "Canon PowerShot SD500",
    "manu": "Canon Inc.",
    "manu_id_s": "canon",
    "cat": [
        "electronics",
        "camera"
    ],
    "features": [
        "3x zoom, 7.1 megapixel Digital ELPH",
        "movie clips up to 640x480 @30 fps",
        "2.0" TFT LCD, 118,000 pixels",
        "built in flash, red-eye reduction"
    ],
    "includes": "32MB SD card, USB cable, AV cable, battery",
    "weight": 6.4,
    "price": 329.95,
    "price_c": "329.95,USD",
    "popularity": 7,
    "inStock": true,
    "manufacturedate_dt": "2006-02-13T15:26:37Z",
    "store": "45.19614,-93.90341",
    "_version_": 1500358264225792000
}
,...
Quick Starting Solr

The document list is pretty straightforward. By default, Solr will list all of the stored fields. Not all of the fields are necessarily stored—that is, you can query on them but not retrieve their value—an optimization choice. Notice that it uses the basic data types of strings, integers, floats, and Booleans. Also note that certain fields, such as features and cat are multivalued, as indicated by the use of [] to denote an array in JSON.

This was a basic keyword search. As you start using more search features such as faceting and highlighting, you will see additional information following the response element.

Some statistics

Let's take a look at the statistics available via the Plugins/Stats page. This page provides details on all the components of Solr. Browse to CORE and then pick a Searcher. Before we loaded data into Solr, this page reported that numDocs was 0, but now it should be 32.

Now take a look at the update handler stats by clicking on the UPDATEHANDLER and then expand the stats for the update handler by clicking on the updateHandler toggle link on the right-hand side of the screen. Notice that the /update request handler has some stats too:

```
class: org.apache.solr.update.DirectUpdateHandler2  
version: 1.0  
description: Update handler that efficiently directly updates the on-disk main lucene index  
src: null  

stats:  
.commits: 4  
.autocommit.maxTime: 15000ms  
.autocommits: 1  
.soft.autocommits: 1  
.optimizes: 0  
.rollback: 0  
.expungeDeletes: 0  
.docPending: 0  
.adds: 0  
.deletesById: 0  
.deletesByQuery: 0  
.errors: 0  
.cumulative_adds: 65  
.cumulative_deletesById: 0  
.cumulative_deletesByQuery: 0  
.cumulative_errors: 0  
.transaction_updates_total_size: 17503  
.transaction_updates_total_number: 3
```
If you think of Solr as a RESTful server, then the various public end points are exposed under the **QUERYHANDLER** menu. Solr isn't exactly REST-based, but it is very similar. Look at the `/update` to see the indexing performance, and `/select` for query performance.

These statistics are accumulated since when Solr was started or reloaded, and they are not stored to disk. As such, you cannot use them for long-term statistics. There are third-party SaaS solutions referenced in *Chapter 11, Deployment*, which capture more statistics and persist it long-term.

The sample browse interface

The final destination of our quick Solr tour is to visit the so-called browse interface—available at [http://localhost:8983/solr/techproducts/browse](http://localhost:8983/solr/techproducts/browse). It's for demonstrating various Solr features:

- **Standard keyword search**: Here, you can experiment with Solr's syntax.
- **Query debugging**: Here, you can toggle display of the parsed query and document score "explain" information.
- **Query-suggest**: Here, you can start typing a word like *enco* and suddenly "encoded" will be suggested to you.
- **Highlighting**: Here, the highlighting of query words in search results is in bold, which might not be obvious.
- **More-like-this**: This returns documents with similar words.
- **Faceting**: This includes field value facets, query facets, numeric range facets, and date range facets.
- **Clustering**: This shows how the search results cluster together based on certain words. You must first start Solr as the instructions describe in the lower left-hand corner of the screen.
- **Query boosting**: This influences the scores by product price.
- **Geospatial search**: Here, you can filter by distance. Click on the spatial link at the top-left to enable this.

This is also a demonstration of **Solritas**, which formats Solr requests using templates that are based on Apache Velocity. The templates are VM files in `example/techproducts/solr/techproducts/conf/velocity`. Solritas is primarily for search UI prototyping. It is not recommended for building anything substantial. See *Chapter 9, Integrating Solr*, for more information.
Quick Starting Solr

The browse UI as supplied assumes the default example Solr schema. It will not work out of the box against another schema without modification.

Here is a screenshot of the browse interface; not all of it is captured in this image:

![Solr Browse Interface Screenshot](image)

**Configuration files**

When you start up Solr using the `–e techproducts` parameter, it loads the configuration files from `/server/solr/configsets/sample_techproducts_configs`. These configuration files are extremely well documented.

A Solr core's instance directory is laid out like this:

- **conf**: This directory contains configuration files. The `solrconfig.xml` and `schema.xml` files are most important, but it will also contain some other `.txt` and `.xml` files, which are referenced by these two.
- **conf/schema.xml**: This is the schema for the index, including field type definitions with associated analyzer chains.
- **conf/solrconfig.xml**: This is the primary Solr configuration file.
• **conf/lang**: This directory contains language translation .txt files that is used by several components.

• **conf/xslt**: This directory contains various XSLT files that can be used to transform Solr's XML query responses into formats such as Atom and RSS. See Chapter 9, Integrating Solr.

• **conf/velocity**: This includes the HTML templates and related web assets for rapid UI prototyping using Solritas, covered in Chapter 9, Integrating Solr. The previously discussed browse UI is implemented with these templates.

• **lib**: Where extra Java JAR files can be placed that Solr will load on startup. This is a good place to put contrib JAR files, and their dependencies. You’ll need to create this directory on your own, though; it doesn't exist by default.

Unlike typical database software, in which the configuration files don't need to be modified much (if at all) from their defaults, you will modify Solr's configuration files extensively—especially the schema. The as-provided state of these files is really just an example to both demonstrate features and document their configuration and should not be taken as the only way of configuring Solr. It should also be noted that in order for Solr to recognize configuration changes, a core must be reloaded (or simply restart Solr).

Solr’s schema for the index is defined in schema.xml. It contains the index’s fields within the `<fields>` element and then the field type definitions within the `<types>` element. You will observe that the names of the fields in the documents we added to Solr intuitively correspond to the sample schema. Aside from the fundamental parts of defining the fields, you might also notice the `<copyField>` elements, which copy an input field as provided to another field. There are various reasons for doing this, but they boil down to needing to index data in different ways for specific search purposes. You’ll learn all that you could want to know about the schema in the next chapter.

Each Solr core’s `solrconfig.xml` file contains lots of parameters that can be tweaked. At the moment, we’re just going to take a peek at the request handlers, which are defined with the `<requestHandler>` elements. They make up about half of the file. In our first query, we didn't specify any request handler, so we got the default one:

```
<requestHandler name="/select" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these will be overridden by parameters in the request -->
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
  </lst>
</requestHandler>
```
Each HTTP request to Solr, including posting documents and searches, goes through a particular request handler. Handlers can be registered against certain URL paths by naming them with a leading /. When we uploaded the documents earlier, it went to the handler defined like this, in which /update is a relative URL path:

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

Requests to Solr are nearly completely configurable through URL parameters or POST'ed form parameters. They can also be specified in the request handler definition within the `<lst name="defaults">` element, such as how rows is set to 10 in the previously shown request handler. The well-documented file also explains how and when they can be added to appends, or invariants named `lst` blocks. This arrangement allows you to set up a request handler for a particular application that will be searching Solr, without forcing the application to specify all of its search parameters. More information on request handlers can be found in Chapter 5, Searching.

### What's next?

You now have an excellent, broad overview of Solr! The numerous features of this tool will no doubt bring the process of implementing a world-class search engine closer to reality. But creating a real, production-ready search solution is a big task. So, where do you begin? As you're getting to know Solr, it might help to think about the main process in three phases: indexing, searching, and application integration.

### Schema design and indexing

In what ways do you need your data to be searched? Will you need faceted navigation, spelling suggestions, or more-like-this capabilities? Knowing your requirements up front is the key in producing a well-designed search solution. Understanding how to implement these features is critical. A well-designed schema lays the foundation for a successful Solr implementation.

However, during the development cycle, having the flexibility to try different field types without changing the schema and restarting Solr can be very handy. The dynamic fields feature allows you to assign field types by using field name conventions during indexing. Solr provides many useful predefined dynamic fields. Chapter 2, Schema Design, will cover this in-depth.
However, you can also get started right now. Take a look at the stock dynamic fields in `/server/solr/configsets/sample_techproducts_configs/conf/schema.xml`. The dynamicField, XML tags represent what is available. For example, the dynamicField named *_b* allows you to store and index Boolean data types; a field named admin_b would match this field type.

For the stock dynamic fields, here is a subset of what's available from the schema.xml file:

- _i_: This includes the indexed and stored integers
- _ss_: This includes the stored and indexed, multi-valued strings
- _dt_: This includes the indexed and stored dates
- _p_: This includes the indexed and stored lat/lon types

To make use of these fields, you simply name your fields using those suffixes—example/exampledocs/ipod_other.xml makes good use of the *_dt* type with its manufacturedate_dt field. Copying an example file, adding your own data, changing the suffixes, and indexing (via the SimplePost tool) is all as simple as it sounds. Give it a try!

**Text analysis**

It's probably a good time to talk a little more about text analysis. When considering field types, it's important to understand how your data is processed. For each field, you'll need to know its data type, and whether or not the value should be stored and/or indexed. For string types, you'll also need to think about how the text is analyzed.

Simply put, text analysis is the process of extracting useful information from a text field. This process normally includes two steps: tokenization and filtering. Analyzers encapsulate this entire process, and Solr provides a way to mix and match analyzer behaviors by configuration.

Tokenizers split up text into smaller chunks called tokens. There are many different kinds of tokenizers in Solr, the most common of which splits text on word boundaries, or whitespace. Others split on regular expressions, or even word prefixes. The tokenizer produces a stream of tokens, which can be fed to an optional series of filters.

Filters, as you may have guessed, commonly remove *noise*—things such as punctuation and duplicate words. Filters can even lower/upper case tokens, and inject word synonyms.

Once the tokens pass through the analyzer processor chain, they are added to the Lucene index. *Chapter 2, Schema Design*, covers this process in detail.
Quick Starting Solr

**Searching**

The next step is, naturally, searching. For most applications processing user queries, you will want to use the [e]dismax query parser, set with `defType=edismax`. It is not the default but arguably should be in our opinion; [e]dismax handles end-user queries very well. There are a few more configuration parameters it needs, described in *Chapter 5, Searching*.

Here are a few example queries to get you thinking.

Be sure to start up Solr and index the sample data by following the instructions in the previous section.

Find all the documents that have the phrase **hard drive** in their **cat** field:

http://localhost:8983/solr/techproducts/select?q=cat:"hard+drive"

Find all the documents that are in-stock, and have a popularity greater than 6:

http://localhost:8983/solr/techproducts/select?q=+inStock:true+AND+popularity:[6+TO+*]

Here's an example using the eDisMax query parser:

http://localhost:8983/solr/techproducts/select?q=ipod&defType=edismax&qf=name^3+manu+cat&fl=*,score

This returns documents where the user query in q matches the name, manu, and cat fields. The \(^3\) after the manu field tells Solr to boost the relevancy of the document scores when the manu field matches. The fl param tells Solr what fields to return—The \(\ast\) means return all fields, and score is a number that represents how well the document matched the input query.

Faceting and statistics can be seen in this example:

http://localhost:8983/solr/techproducts/select?q=ipod&defType=dismax&qf=name^3+manu+cat&fl=*,score&rows=0&facet=true&facet.field=manu_id_s&facet.field=cat&stats=true&stats.field=price&stats.field=weight

This builds on the previous, dismax example, but instead of returning documents (rows=0), Solr returns multiple facets and stats field values.

For detailed information on searching, see *Chapter 5, Searching*. 
Integration

If the previous tips on indexing and searching are enough to get you started, then you must be wondering how you integrate Solr and your application. By far, the most common approach is to communicate with Solr via HTTP. You can make use of one of the many HTTP client libraries available. Here's a small example using the Ruby library, RSolr:

```ruby
require "rsolr"
client = RSolr.connect
params = {:q => "ipod", :defType => "dismax", :qf => "name^3 manu
cat", :fl => "*,score"}
result = client.select(:params => params)
result["response"]["docs"].each do |doc|
  puts doc.inspect
end
```

Using one of the previous sample queries, the result of this script would print out each document, matching the query `ipod`.

There are many client implementations, and finding the right one for you is dependent on the programming language your application is written in. Chapter 9, *Integrating Solr*, covers this in depth, and will surely set you in the right direction.

Resources outside this book

The following are some Solr resources other than this book:


- *Apache Solr Reference Guide* is a detailed, online resource contributed by Lucidworks to the Solr community. You can find the latest version at [https://cwiki.apache.org/confluence/display/solr/Apache+Solr+Reference+Guide](https://cwiki.apache.org/confluence/display/solr/Apache%2BSolr%2BReference%2BGuide). Consider downloading the PDF corresponding to the Solr release you are using.

- Solr's Wiki at [http://wiki.apache.org/solr/](http://wiki.apache.org/solr/) has a lot of great documentation and miscellaneous information. For a Wiki, it's fairly organized too. In particular, if you use a particular app-server in production, then there is probably a Wiki page there on specific details.
Quick Starting Solr

• Within the Solr installation, you will also find that there are README.txt files in many directories within Solr and that the configuration files are very well documented. Read them!

The solr-user@lucene.apache.org mailing list contains a wealth of information. If you have a few discriminating keywords, then you can find nuggets of information in there with a search engine. The mailing lists of Solr and other Lucene subprojects are best searched at http://find.searchhub.org or http://search-lucene.com/solr or http://nabble.com.

We highly recommend that you subscribe to the Solr-users mailing list. You'll learn a lot and potentially help others, too.

• Solr's issue tracker contains information on enhancements and bugs. It's available at http://issues.apache.org/jira/browse/SOLR and it uses Atlassian's JIRA software. Some of the comments attached to these issues can be extensive and enlightening.

Notation convention

Solr's JIRA issues are referenced like this—SOLR-64. You'll see such references in this book and elsewhere. You can easily look these up at Solr's JIRA. You might also see issues for Lucene that follow the same convention, for example, LUCENE-1215.

There are, of course, resources for Lucene, such as Lucene In Action, Second Edition, Michael McCandless, Erik Hatcher, and Otis Gospodneti, Manning Publications. If you intend to dive into Solr's internals, then you will find Lucene resources helpful, but that is not the focus of this book.

Summary

This completes a quick introduction to Solr. In the following chapters, you're really going to get familiar with what Solr has to offer. We recommend that you proceed in order from the next chapter through Chapter 8, Search Components, because these build on each other and expose nearly all of the capabilities in Solr. These chapters are also useful as a reference to Solr's features. You can, of course, skip over sections that are not interesting to you. Chapter 9, Integrating Solr, is one you might peruse at any time, as it may have a section applicable to your Solr usage scenario. Finally, be sure that you don't miss the appendix for a search quick-reference cheat-sheet.
Where to buy this book

You can buy Apache Solr Enterprise Search Server Third Edition 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.