# Technical Design Document – File Search

# Author : Anil Adusumalli

# Date : 14/03/2020

# Version:1.0

* 1 [Purpose](#id-.IDDDocument-CreateDocumentvSprint12)
* 2 [Objectives](#id-.IDDDocument-CreateDocumentvSprint12)
* 3 [Interface Version to Design Document matrix](#id-.IDDDocument-CreateDocumentvSprint12)
* 4 [Key Contacts](#id-.IDDDocument-CreateDocumentvSprint12)
* 5 [Business Overview](#id-.IDDDocument-CreateDocumentvSprint12)
  + 5.1 [Business Outcome](#id-.IDDDocument-CreateDocumentvSprint12)
* 6 [Technical Design](#id-.IDDDocument-CreateDocumentvSprint12)
  + 6.1 [Interface Structure](#id-.IDDDocument-CreateDocumentvSprint12)
    - 6.1.1 [Usage](#id-.IDDDocument-CreateDocumentvSprint12)
    - 6.1.2 [Error/Alternate Behaviour](#id-.IDDDocument-CreateDocumentvSprint12)
  + 6.2 [Structural Design](#id-.IDDDocument-CreateDocumentvSprint12)
    - 6.2.1 [Request Payload Definition](#id-.IDDDocument-CreateDocumentvSprint12)
    - 6.2.2 [Response Payload Definition](#id-.IDDDocument-CreateDocumentvSprint12)
  + 6.3 [Interface Qualities](#id-.IDDDocument-CreateDocumentvSprint12)
  + 6.4 [Behavioural Design](#id-.IDDDocument-CreateDocumentvSprint12)
* 7 [Constraint](#id-.IDDDocument-CreateDocumentvSprint12)
* 8 [Outstanding Questions](#id-.IDDDocument-CreateDocumentvSprint12)
* 9 [Governance Summary](#id-.IDDDocument-CreateDocumentvSprint12)
* 10 [Change Requests](#id-.IDDDocument-CreateDocumentvSprint12)
* 11 [Related Design Dependencies](#id-.IDDDocument-CreateDocumentvSprint12)

# Purpose

The purpose of this document is to describe following Integration items at a detailed level. This document describes the interface usage, behaviour and constraints.

# Objectives

The objectives of this document are to define:

1. The functional (business) scope of the interface or set of business activities that the interface (and where applicable its operations) enables / realises.
2. Specific technical details such as; Creating a Hadoop Cluster, Aparche Solr usage requirements and patterns that the interface supports.
3. Interface system qualities (Non Functional Requirements) such as; performance and volume, security and privacy constraints.

# Interface Version to Design Apache Spark, Tika and Solr Search Platform

The table below lists each version of the built interface/application that this specification complies with.

|  |  |  |  |
| --- | --- | --- | --- |
| **Integration Version** | **Build Date** | **Environment** | **Summary of Change** |
|  |  |  |  |

# Key Contacts

The following table list the key technical and business contacts associated with the interface defined within this document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | **Full Name** | **Responsibility** | **Email** | **Phone** |
| Integration Architect | Anil A | Integration Architecture |  |  |
| Senior Solution Designer |  | Integration Architecture |  |  |
| Senior Manager |  | Integration Architecture |  |  |

# Business Overview

**Use Case:**

We have received a palette of hard drives containing text documents. We do not know how many text documents exist at this time but we expect at least one more palette in the future. Initial estimate suggests 900 Million plus documents.

We need the ability to execute multiple searches to locate any of these text documents containing one or more key words. E.g. find me all documents containing the word “elephant” Results should be returned in reasonable time nominally under 1 hr.

## Business Outcome

|  |
| --- |
|  |
| **Key** | **Summary** | **T** | **Created** | **Updated** | **Due** | **Assignee** | **Reporter** | **P** | **Status** | **Resolution** |

### Solution Design:

Apache Solr Cloud/Elastice search

Files in HDD

Solr /Elastice search

Apache Spark + Tika

/Hive

Files in HDD

### Apache Tika :

### Apache Tika is a content detection and analysis framework, written in Java, stewarded at the Apache Software Foundation. It detects and extracts metadata and text from over a thousand different file types, and as well as providing a Java library, has server and command-line editions suitable for use from other programming languages.

### Solr: the powerful search platform from Apache

### Solr (pronounced: solar) is an open source based on ****Lucene Core**** and is written in Java. As a search platform, ****Apache Solr**** is one of the most popular tools for integrating vertical search engines. Among Solr’s advantages are also its wide range of functions (which also includes faceting search results, for example) and accelerated indexing. It also runs on server containers like Apache Tomcat.

### Solr is a standalone enterprise search server with a REST-like API. You put documents in it (called "indexing") via JSON, XML, CSV or binary over HTTP. You query it via HTTP GET and receive JSON, XML, CSV or binary results.

### Advanced Full-Text Search Capabilities

Solr enables powerful matching capabilities including phrases, wildcards, joins, grouping and much more across any data type

### Install Solr :

### You can install Solr in any system where a suitable Java Runtime Environment (JRE) is available, as detailed below. Currently this includes Linux, OS X, and Microsoft Windows. The instructions in this section should work for any platform, with a few exceptions for Windows as noted.

You will need the Java Runtime Environment (JRE) version 1.8 or higher. At a command line, check your Java version like this:

$ java -version

java version "1.8.0\_60"

Solr is available from the Solr website at <http://lucene.apache.org/solr/>.

For Linux/Unix/OSX systems, download the .tgz file. For Microsoft Windows systems, download the .zip file.

When getting started, all you need to do is extract the Solr distribution archive to a directory of your choosing. When you’re ready to setup Solr for a production environment, please refer to the instructions provided on the [Taking Solr to Production](https://lucene.apache.org/solr/guide/6_6/taking-solr-to-production.html#taking-solr-to-production) page.

To keep things simple for now, extract the Solr distribution archive to your local home directory, for instance on Linux, do:

cd ~/

tar zxf solr-x.y.z.tgz

Once extracted, you are now ready to run Solr using the instructions provided in the [Running Solr](https://lucene.apache.org/solr/guide/6_6/running-solr.html#running-solr) section.

## Start the Server

If you didn’t start Solr after installing it, you can start it by running bin/solr from the Solr directory.

bin/solr start

If you are running Windows, you can start Solr by running bin\solr.cmd instead.

bin\solr.cmd start

This will start Solr in the background, listening on port 8983.

When you start Solr in the background, the script will wait to make sure Solr starts correctly before returning to the command line prompt.

The bin/solr and bin\solr.cmd scripts allow you to customize how you start Solr. Let’s work through a few examples of using the bin/solr script (if you’re running Solr on Windows, the bin\solr.cmd works the same as what is shown in the examples below):

### Solr Script Options

The bin/solr script has several options.

#### Script Help

To see how to use the bin/solr script, execute:

bin/solr -help

For specific usage instructions for the **start** command, do:

bin/solr start -help

#### Start Solr in the Foreground

Since Solr is a server, it is more common to run it in the background, especially on Unix/Linux. However, to start Solr in the foreground, simply do:

bin/solr start -f

If you are running Windows, you can run:

bin\solr.cmd start -f

#### Start Solr with a Different Port

To change the port Solr listens on, you can use the -p parameter when starting, such as:

bin/solr start -p 8984

#### Stop Solr

When running Solr in the foreground (using -f), then you can stop it using Ctrl-c. However, when running in the background, you should use the **stop** command, such as:

bin/solr stop -p 8983

The stop command requires you to specify the port Solr is listening on or you can use the -all parameter to stop all running Solr instances.

#### Start Solr with a Specific Bundled Example

Solr also provides a number of useful examples to help you learn about key features. You can launch the examples using the -e flag. For instance, to launch the "techproducts" example, you would do:

bin/solr -e techproducts

#### Check if Solr is Running

If you’re not sure if Solr is running locally, you can use the status command:

bin/solr status

This will search for running Solr instances on your computer and then gather basic information about them, such as the version and memory usage.

That’s it! Solr is running. If you need convincing, use a Web browser to see the Admin Console.

http://localhost:8983/solr/

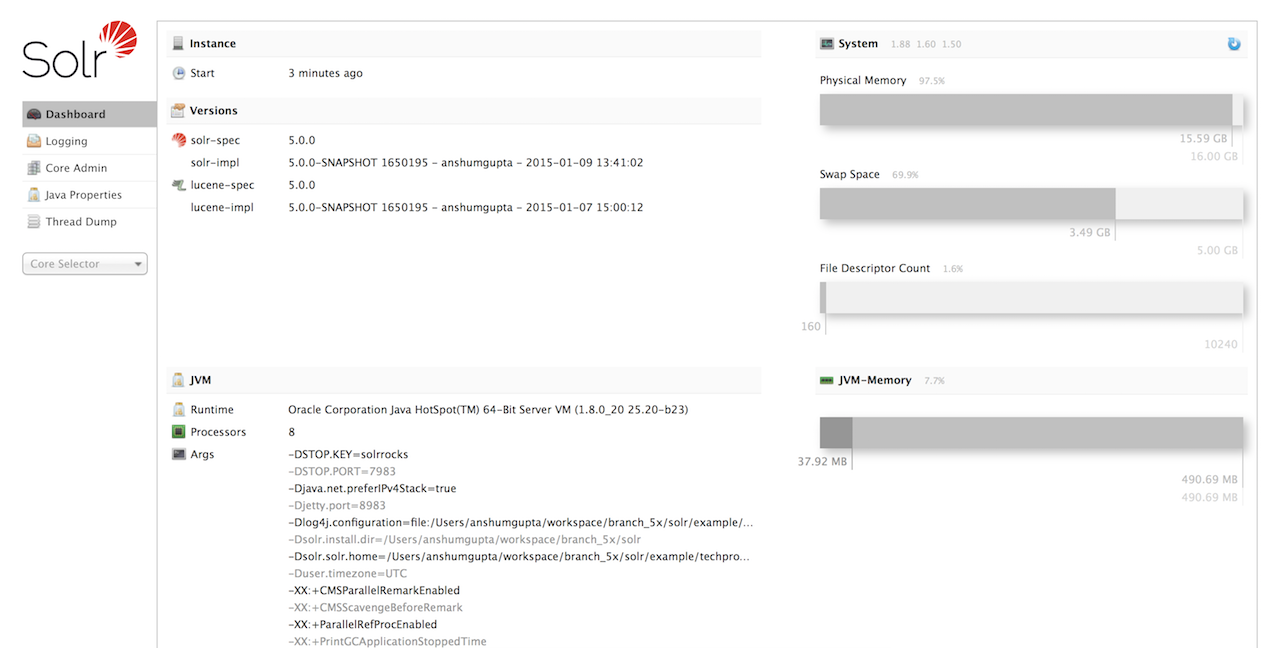


Figure 1. The Solr Admin interface.

If Solr is not running, your browser will complain that it cannot connect to the server. Check your port number and try again.

## Create a Core

If you did not start Solr with an example configuration, you would need to create a core in order to be able to index and search. You can do so by running:

bin/solr create -c <name>

This will create a core that uses a data-driven schema which tries to guess the correct field type when you add documents to the index.

To see all available options for creating a new core, execute:

bin/solr create -help

## Add Documents

Solr is built to find documents that match queries. Solr’s schema provides an idea of how content is structured (more on the schema [later](https://lucene.apache.org/solr/guide/6_6/documents-fields-and-schema-design.html#documents-fields-and-schema-design)), but without documents there is nothing to find. Solr needs input before it can do much.

You may want to add a few sample documents before trying to index your own content. The Solr installation comes with different types of example documents located under the sub-directories of the example/ directory of your installation.

In the bin/ directory is the post script, a command line tool which can be used to index different types of documents. Do not worry too much about the details for now. The [Indexing and Basic Data Operations](https://lucene.apache.org/solr/guide/6_6/indexing-and-basic-data-operations.html#indexing-and-basic-data-operations) section has all the details on indexing.

To see some information about the usage of bin/post, use the -help option. Windows users, see the section for [Post Tool on Windows](https://lucene.apache.org/solr/guide/6_6/post-tool.html#PostTool-WindowsSupport).

bin/post can post various types of content to Solr, including files in Solr’s native XML and JSON formats, CSV files, a directory tree of rich documents, or even a simple short web crawl. See the examples at the end of bin/post -help for various commands to easily get started posting your content into Solr.

Go ahead and add all the documents in some example XML files:

$ bin/post -c gettingstarted example/exampledocs/\*.xml

SimplePostTool version 5.0.0

Posting files to [base] url http://localhost:8983/solr/gettingstarted/update...

Entering auto mode. File endings considered are xml,json,csv,pdf,doc,docx,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log

POSTing file gb18030-example.xml (application/xml) to [base]

POSTing file hd.xml (application/xml) to [base]

POSTing file ipod\_other.xml (application/xml) to [base]

POSTing file ipod\_video.xml (application/xml) to [base]

POSTing file manufacturers.xml (application/xml) to [base]

POSTing file mem.xml (application/xml) to [base]

POSTing file money.xml (application/xml) to [base]

POSTing file monitor.xml (application/xml) to [base]

POSTing file monitor2.xml (application/xml) to [base]

POSTing file mp500.xml (application/xml) to [base]

POSTing file sd500.xml (application/xml) to [base]

POSTing file solr.xml (application/xml) to [base]

POSTing file utf8-example.xml (application/xml) to [base]

POSTing file vidcard.xml (application/xml) to [base]

14 files indexed.

COMMITting Solr index changes to http://localhost:8983/solr/gettingstarted/update...

Time spent: 0:00:00.153

That’s it! Solr has indexed the documents contained in those files.

Now that you have indexed documents, you can perform queries. The simplest way is by building a URL that includes the query parameters. This is exactly the same as building any other HTTP URL.

For example, the following query searches all document fields for "video":

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

Notice how the URL includes the host name (localhost), the port number where the server is listening (8983), the application name (solr), the request handler for queries (select), and finally, the query itself (q=video).

The results are contained in an XML document, which you can examine directly by clicking on the link above. The document contains two parts. The first part is the responseHeader, which contains information about the response itself. The main part of the reply is in the result tag, which contains one or more doc tags, each of which contains fields from documents that match the query. You can use standard XML transformation techniques to mold Solr’s results into a form that is suitable for displaying to users. Alternatively, Solr can output the results in JSON, PHP, Ruby and even user-defined formats.

Just in case you are not running Solr as you read, the following screen shot shows the result of a query (the next example, actually) as viewed in Mozilla Firefox. The top-level response contains a lst named responseHeader and a result named response. Inside result, you can see the three docs that represent the search results.

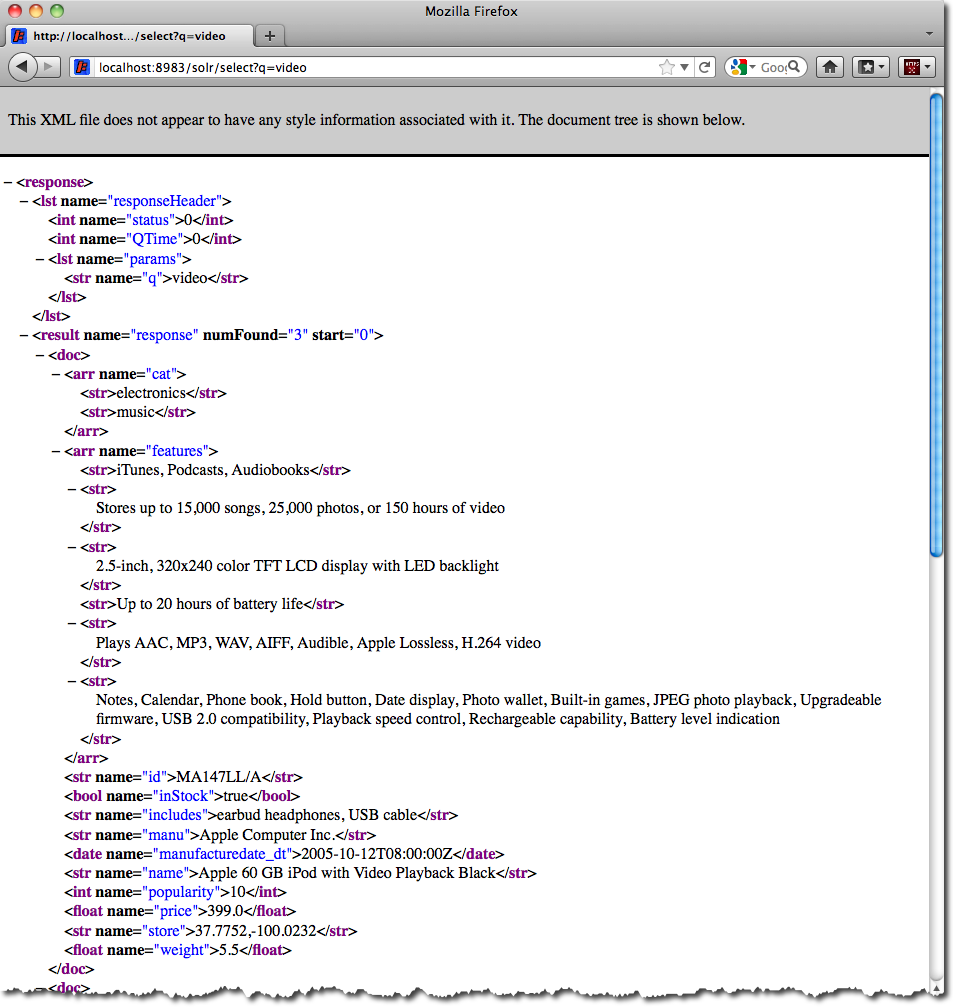


Figure 2. An XML response to a query.

Once you have mastered the basic idea of a query, it is easy to add enhancements to explore the query syntax. This one is the same as before but the results only contain the ID, name, and price for each returned document. If you don’t specify which fields you want, all of them are returned.

http://localhost:8983/solr/gettingstarted/select?q=video&fl=id,name,price

Here is another example which searches for "black" in the name field only. If you do not tell Solr which field to search, it will search default fields, as specified in the schema.

http://localhost:8983/solr/gettingstarted/select?q=name:black

You can provide ranges for fields. The following query finds every document whose price is between $0 and $400.

http://localhost:8983/solr/gettingstarted/select?q=price:0%20TO%20400&fl=id,name,price

[Faceted browsing](https://lucene.apache.org/solr/guide/6_6/faceting.html#faceting) is one of Solr’s key features. It allows users to narrow search results in ways that are meaningful to your application. For example, a shopping site could provide facets to narrow search results by manufacturer or price.

Faceting information is returned as a third part of Solr’s query response. To get a taste of this power, take a look at the following query. It adds facet=true and facet.field=cat.

http://localhost:8983/solr/gettingstarted/select?q=price:0%20TO%20400&fl=id,name,price&facet=true&facet.field=cat

In addition to the familiar responseHeader and response from Solr, a facet\_counts element is also present. Here is a view with the responseHeader and response collapsed so you can see the faceting information clearly.

**An XML Response with faceting**

**<response>**

**<lst** name="responseHeader"**>**

...

**</lst>**

**<result** name="response" numFound="9" start="0"**>**

**<doc>**

**<str** name="id"**>**SOLR1000**</str>**

**<str** name="name"**>**Solr, the Enterprise Search Server**</str>**

**<float** name="price"**>**0.0**</float></doc>**

...

**</result>**

**<lst** name="facet\_counts"**>**

**<lst** name="facet\_queries"**/>**

**<lst** name="facet\_fields"**>**

**<lst** name="cat"**>**

**<int** name="electronics"**>**6**</int>**

**<int** name="memory"**>**3**</int>**

**<int** name="search"**>**2**</int>**

**<int** name="software"**>**2**</int>**

**<int** name="camera"**>**1**</int>**

**<int** name="copier"**>**1**</int>**

**<int** name="multifunction printer"**>**1**</int>**

**<int** name="music"**>**1**</int>**

**<int** name="printer"**>**1**</int>**

**<int** name="scanner"**>**1**</int>**

**<int** name="connector"**>**0**</int>**

**<int** name="currency"**>**0**</int>**

**<int** name="graphics card"**>**0**</int>**

**<int** name="hard drive"**>**0**</int>**

**<int** name="monitor"**>**0**</int>**

**</lst>**

**</lst>**

**<lst** name="facet\_dates"**/>**

**<lst** name="facet\_ranges"**/>**

**</lst>**

**</response>**

The facet information shows how many of the query results have each possible value of the cat field. You could easily use this information to provide users with a quick way to narrow their query results. You can filter results by adding one or more filter queries to the Solr request. This request constrains documents with a category of "software".

http://localhost:8983/solr/gettingstarted/select?q=price:0%20TO%20400&fl=id,name,price&facet=true&facet.field=cat&fq=cat:software

## Interface Structure

### Usage

**Interface – FileSearch Utility (Document Search)**

### Error/Alternate Behaviour

|  |  |
| --- | --- |
| **Error/Alternate Condition** | **Behaviour** |
| Any Technical Error while using Solr API’s | Fault is returned by Solr Search with relevant error text and error code. e.g. System is down, network is down. If the error is related to payload, payload can be fixed and message can be resubmitted. |
|  |  |

**Error Scenarios for File Search Service:**

| **Scenario** | **High-Level Handling** |
| --- | --- |
| Dialogue Wrapper System    unavailable or system not picking up service requests | The Search Controller will    provide an error call-back to user with error:  <Status>    Failure </Status>  <Description>    Fail to generate </Description> |
| Invalid Request Data    (Dialogue Wrapper) | The search  Controller will provide an error call-back to user with error:  <Status>    Failure </Status>  <Description>    Fail to generate </Description> |
|  |  |

## Structural Design

### Request Payload Definition

### Response Payload Definition

## Interface Qualities

**Performance Architecture Blueprint:**

Document Search - Less than 30 seconds

Document Storage (Document Upload) - Maximum 10 seconds

End to end request/store and distribute - Average 3 seconds or less, maximum 10 seconds.

## Behavioural Design

* Flow chats
* Sequence diagrams

# Constraint

Delivery mechanism is a mandatory in the Search Platform.

# Outstanding Questions

* how big are the files ? (Average file size)
* if you plan to add more files later (update the index would be needed, and maybe a scalable solution too for a lot more than 1000 files)
* how many times you want to search a particular string?
* how fast should be the system under heavy concurrency (multiple users searching in the same time)
* how fast do you want the results returned (seconds is fine or milliseconds is desired?)
* what would be the search criteria’s?
* how to store the resulted documents?
* what are the expected search results from source (hdd) systems?
* Is search documents should get file content and file details ( metadata ) ?
* What’s the documents storage mechanism ?

# Governance Summary

The following section is used to summarise the change process and maintenance procedure for the given interface

# Change Requests

|  |
| --- |
|  |
| **Key** | **Summary** | **T** | **Created** | **Updated** | **Due** | **Assignee** | **Reporter** | **P** | **Status** | **Resolution** |

# Related Design Dependencies

* SRS Document ( System Requirements )
* Enterprise solution architecture
* Additional documents
* Use Case Documents
* JIRA / Confluence links