LucidWorks Search

DOCUMENTATION

LucidWorks™

Table of Contents

LucidWorks Search Documentation

The LucidWorks Search Documentation is organized into several guides that cover all aspects of using and implementing a search application with LucidWorks Search, whether on-premise or hosted on AWS or Azure.

Installation & Upgrade Guide

- Installing LucidWorks Search
- System Directories and Logs
- Upgrade instructions for v2.5
- Review changes from LucidWorks v2.1 to v2.5

System Configuration Guide

- Troubleshooting crawl issues
- Alerts configuration
- Query options
- Custom fields, field types, and other index customizations
- Performance considerations and system monitoring
- Distributed search and indexing
- Security options

Lucid Query Parser

- How the default query parser handles user requests
- Customization options

LucidWorks REST API Reference

- Configure data sources and administer crawls
- Set system settings
- Manage fields, field types, and collections
- Example clients in C#, Perl and Python

Custom Connector Guide

- Introduction to Lucid Connector Framework
- How To Create A Connector

How to Use this Documentation

Audience and Scope

This guide is intended for search application developers and administrators who want to use LucidWorks Search to create world class search applications for their websites.

While LucidWorks Search is built on Solr, and many of its features are implementations of Solr and Lucene features, this Guide does not cover basic Solr or Lucene configuration. We do, however, point out where LucidWorks Search deviates from Solr or Lucene standard configuration practices, and have provided links to Solr and Lucene documentation where possible for further explanation if the functionality in LucidWorks Search is identical to Solr or Lucene.

One important note to remember is that LucidWorks is multi-core enabled by default, with `collection1` as the default core. This means that standard Solr paths such as `http://localhost:port/solr/*`, as shown in Solr documentation, would be `http://localhost:port/solr/collection1/*` in LucidWorks Search.

Conventions

Special notes are included throughout these pages.

Note Type	Look & Description
Information	Notes with a blue background are used for information that is important for you to know.
Notes	Notes are further clarifications of important points to keep in mind while using LucidWorks.

Tip	Notes with a green background are Helpful Tips.
Warning	Notes with a red background are warning messages.
Cloud	**Information for LucidWorks Search in the Cloud Users** Information specifically for LucidWorks Search customers on the AWS or Azure Platform.

REST API Conventions

Many of the LucidWorks Search REST APIs support several methods (such as POST, GET, PUT, DELETE) and each is documented with detailed attribute descriptions and examples of inputs and outputs. Each description includes the path to the API endpoint, parameters for input, and the attributes returned as a result of the request.

Parameters

Several of the paths shown in the API documentation include parameters that need to be modified for your installation and specific configuration. These are indicated in *italics*.

For example, getting the details of a data source is shown as:

```
GET /api/collection/collection/datasources/id.
```

If you were using '`collection1`' and data source '`3`', you would enter:

```
GET /api/collection/collection1/datasources/3.
```

Server Addresses

The LucidWorks Search REST API uses the Core component, installed at http://localhost:8888/ by default in LucidWorks Search. Many examples in this Guide use this as the server location. If you have installed LucidWorks Search locally, and you changed this location on install, be sure to change the destination of your API requests accordingly.

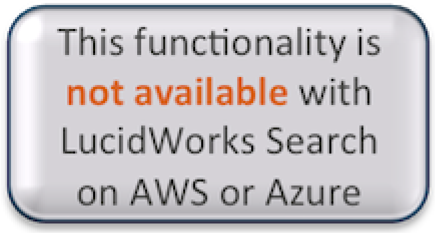
Customers hosted on AWS or Azure should see the section for Customers of LucidWorks Search on AWS or Azure below.

Customers of LucidWorks Search on AWS or Azure

All of the preceding information on this page applies to customers who have LucidWorks Search hosted on either AWS or Azure Platforms, with a few small exception which are detailed below.

Configuration Options

Certain configuration options are available with on-premise installations only (such as installation options, manual configuration file changes, etc.). The following panel will appear on any page or section that does not apply or is not available for LucidWorks Search on the AWS or Azure platforms:


This functionality is **not available** with LucidWorks Search on AWS or Azure

API Conventions for LucidWorks Search on AWS or Azure

Nearly all of the documented REST APIs will work for customers on AWS or Azure, but the example API calls must be modified to include either the Access Key or the API Key and used as authentication credentials. Customers are being transitioned from a simple Access Key to a more secure Basic authentication system that requires a unique API Key.

1. Customers who only have an Access Key can see the key on the My Search Server page and the main Collections Overview page of your instance (click the REST API button above the usage graphs). Example URLs for API calls used in this documentation would then be changed from `http://localhost:8989/api/...` to `http://access.lucidworks.io/<access key>/api/...`. This access key is specific to your instance and should be treated as securely as possible to prevent unauthorized access via the APIs to your system.

2. Customers with Basic authentication have instances which use an URL with "https://s-**XXXXXXXX**.lucidworks.io" where XXXXXXXX is 8 characters (letters or numbers). So, if your instance URL is "https://s-9sdff10b.lucidworks.io/" you would use that in place of any example API calls that used "http://localhost:8888". For example, this call to get all collections:

```
curl 'http://localhost:8888/api/collections'
```

would be changed to:

```
curl -u 'API_Key:password' 'https://s-9sdff10b.lucidworks.io/api/collections'
```

The API_Key can be found by logging in to your LucidWorks Search instance, and clicking "My Account" at the upper right of the screen. Click "API Access" on the left to view the API key. The password is 'x' by default. There is not currently a way to change the default password. You should take care not to expose this key when posting to our forums, as that information could be seen by other LucidWorks Search customers.

For users on LucidWorks Search for Windows Azure, the above URL would be: `'https://s-9sdff10b.azure.lucidworks.io/api/collections'`.

Getting Support & Training

There are several options to get answers to questions besides this documentation:

- The LucidWorks Search Forum is a place to ask questions and share information about your implementation.
- The LucidWorks Search KnowledgeBase has articles written by our support and consulting staff around common issues and questions.
- Training Videos produced by the LucidWorks training team.
- Premium support is also available, providing access to a help desk ticketing system. For more information see Lucene/Solr Support.

Features of LucidWorks Search

LucidWorks Search is the cutting-edge search platform that accelerates and simplifies development of highly accurate, scalable, and cost-effective search applications. It delivers full open source Apache Lucene/Solr community innovation in a convenient, enterprise-ready package, with support from the world's leading experts in open source search.

LucidWorks Search is available for on premise installation, as a cloud-based platform, or as a hybrid. No matter the distribution of choice, organizations take full advantage of all capabilities LucidWorks Search delivers.

Open source power and innovation

- **Built on Solr** Complete, integrated, tested release of Apache Solr 4.0, which adds advanced features such as flexible indexing, fuzzy search and pivot faceting.
- **Distributed Search** Massive scalability of search and indexing managed through Apache Zookeeper, delivering search integrity and consistency that scales.
- **Near-real-time search** Flexible control to rapidly update and delete index data in segments as it becomes available, avoiding the cost of full re-indexing at every update.

Streamlined search configuration, deployment and operations

- **Admin UI** An easy-to-use user interface allows you to configure and tune crawlers, field parameters, user management, security, and user experience options.
- **508 Compliance** User interfaces have been tested with JAWS, Achecker, and Total Validator Pro to ensure compliance with Section 508 standards.

Broad-based content acquisition, with versatile access and data security

- **Cloud Connectors** Built-in support for Hadoop and Amazon S3; traverses HDFS file system hierarchy and permissions in two modes; searches AWS S3 content in specified buckets under specific paths.
- **Enterprise Connectors** Crawls and indexes web and enterprise data sources including DBMS, websites, file systems, and SharePoint. SharePoint and Windows shares use associated Access Control Lists (ACLs) for early-binding of document security attributes.
- **Customizable** Open connector framework is extensible to any repository.
- **Security** Integrated end-to-end management of security, including user and document controls and supporting different models such as LDAP and Sharepoint ACL. Optional HTTPS encryption is also available.

Advanced search experience enhancements easily integrated into your application and infrastructure

- **REST API** Automates administration of search as a service to programmatically manage search optimizations, content acquisition, field behavior, monitoring and infrastructure operations.
- **Monitoring** Integrates with common Open Source tools such as JMX, Zabbix and Nagios to deliver integrated application level statistics and performance data consistent with your IT operational environment.
- **Click Scoring** Included tools can adjust scoring and ranking of documents based on analysis of historical click-through log data; tracks which results were selected by users for individual queries, boosting relevance based on document popularity.
- **User Alerts** Automated notifications keep users up-to-date as new search results are available; end users can can select, define, and manage any valid search query as an alert for automated notification of new search results.

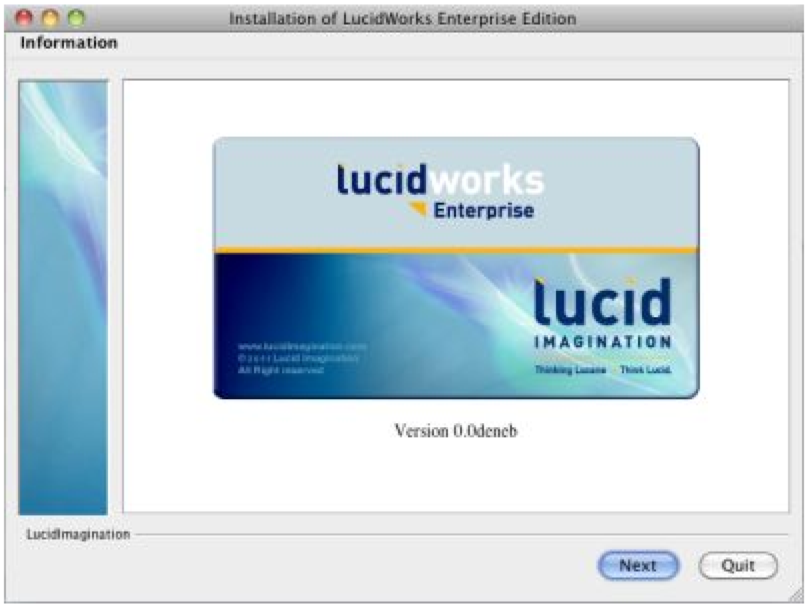
Commercial grade expertise & global support 24x7

- A range of one-year subscriptions including bundled services from the world's leading experts in Solr/Lucene open source.
- Online forums and knowledgebase articles written by LucidWorks consulting and support staff.

Getting Started

The steps to get started with LucidWorks Search are not very different from getting started with any new search platform. One needs to consider the nature of the documents to be indexed, how users will expect to find them, and how results will be presented to users. This section outlines those activities and points to parts of documentation to help you understand how to accomplish the necessary tasks for a successful search application.

If you are new to search applications, the articles in the Introduction provide an overview of how search engines work and the basic concepts of indexing documents and handling user queries.



The obvious first step is to install the application (if you are using LucidWorks Search On-Premise; LucidWorks Search on AWS or Azure, of course, is already installed).

- Installation

In general, LucidWorks Search provides two modes of interacting with the system: the Admin UI or the REST API. When just starting out, it's easier to use the Admin UI, but when developing your search application, you may want to use the API, depending on your needs. LucidWorks Search is split into two components, and it's worth getting a sense for them before diving too deep into application development.

- Working With LucidWorks Search Components

Before any user can send queries to your search applications, you need to index data. LucidWorks Search requires configuring data sources for each content repository that will be added to the index and several types of repositories are supported. These can be created via the Admin UI or with the REST API.

- Creating Data Sources with the Admin UI

- Creating Data Sources with the REST API

When first starting out, it's best to use a small set of documents and test that they are being indexed according to the needs of your users. The built-in Search UI was designed to be used during implementation. Queries can also be sent directly to Solr using the standard Solr syntax.

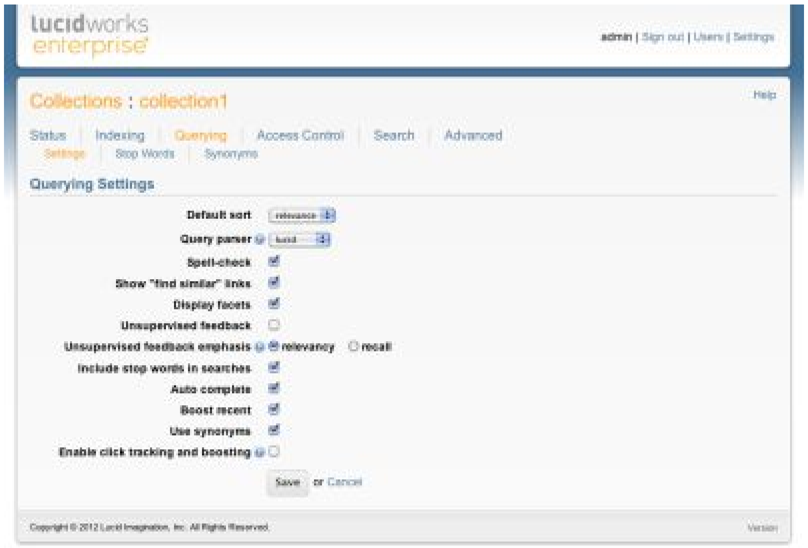
- Using the Search UI
- Solr Direct Access
- Query and Response Examples

Once you see the results of initial crawls, you may realize that some of your documents don't appear as expected, or facets important to you are not appearing as you'd like.

Raw documents are broken up into various fields during the crawling and indexing processes, and the fields contained in your documents may vary from the default fields provided by LucidWorks Search through a file called `schema.xml`. While we've tried to anticipate the needs of most customers, you may find tweaks are required for your content.

In addition, LucidWorks Search provides the ability to separate indexed content into collections, that each have their own field definitions, data sources, synonym lists, activity schedules, query settings and other configurations. It's worth considering if you need to break up your content in this way, and create new collections as needed.

- Understanding Collections
- Creating Collections with the Admin UI
- Creating Collections with the REST API
- Customizing the Field Schema
- Managing Fields with the Admin UI
- Managing Fields with the REST API


lucidworks enterprise
admin | Sign out | Users | Settings
Collections : collection1
Help
Status | Indexing | Querying | Access Control | Search | Advanced
Settings | Stop Words | Synonyms
Save or Cancel
Copyright © 2012 Lucid Imagination, Inc. All Rights Reserved.
Version

Querying Settings

Default sort	relevance
Query parser	lucid
Spell-check	✓
Show "find similar" links	✓
Display facets	✓
Unsupervised feedback	☐
Unsupervised feedback emphasis	[✓ relevancy] recall
Include stop words in searches	✓
Auto complete	✓
Boost recent	✓
Use synonyms	✓
Enable click tracking and boosting	☐

Once the content is being indexed as you expect, you can modify the way user queries are handled and how results are shown to users. There are many features available, such as synonyms, auto-complete, alerting users of new results, boosting documents based on user clicks among other features.

- Synonyms
- Stop Words
- Using User Clicks to Boost Results
- Modifying Query Settings with the Admin UI
- Modifying Query Settings with the REST API
- Lucid Query Parser Guide
- Spell Check
- Auto-Complete
- User Alerts

Before going live with your search application, you'll want to consider user authentication and system security issues. LucidWorks can integrate with LDAP and supports SSL. Additionally, Access Control List information from Windows Shares can be incorporated to restrict result sets to only those documents users are allowed to see. You may also want to integrate with a JMX client, Zabbix or Nagios to monitor system performance.

- LDAP Integration
- Restricting Access to Content
- Enabling SSL
- Securing LucidWorks
- Integrating Monitoring Services

There is much more information in the LucidWorks Platform Documentation.

Installation and Upgrade Guide

This section covers all aspects of installing and upgrading LucidWorks, as well as some things to consider if you are migrating from a purely Solr installation.

New Installs

- Review the **Requirements**.
- Install LucidWorks using a **GUI installer** or a **command line installer**.
- Review the Release Notes for your version.

Upgrading

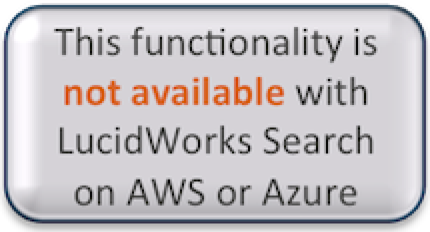
- Review the Upgrade Notes to create a plan for the upgrade.
- Follow the instructions described in Upgrading from a Prior Version.
- Review the Release Notes for your version.

Migrating from Solr to LucidWorks

- Review the **Requirements**.
- Install LucidWorks using a **GUI installer** or a **command line installer**.
- There are some items to consider before migrating your existing Solr configuration files. The section Migrating from Solr to LucidWorks has the details.
- Review the Release Notes for your version.

Information for LucidWorks Search in the Cloud Users
Many of the installation options are not relevant for LucidWorks Search customers who are hosted on AWS and Azure. However, you will want to review the Upgrade Notes to understand what is new in this release. The LucidWorks Operations team will be in touch when your instances will be upgraded. Please contact your support representative if you have questions.

Installation


This functionality is not available with LucidWorks Search on AWS or Azure

There are two ways to install LucidWorks Search on-premise:

- You can run the installer in **graphical mode**, which guides you through a series of dialog boxes, then installs and configures the software.

- You can run the installer in **console mode**, which limits the installer's interface to the command line. If you do not have access to the graphical user interface of the system you are installing the software on, run the installer in console mode.

For Linux and Mac systems, the install file is called `lucidworks-search-installer-2.5.jar`.

For Windows systems, the install file is called `lucidworks-search-installer-2.5.exe`. On Windows, the installer should be run with Administrator privileges to ensure proper installation. During installation, you will be prompted to install LucidWorks Search as a service. While this enables automatic restart, it is not designed as a monitoring or watchdog service that restarts if a dependent process fails.

Requirements

You must have Java 1.6 or higher (JRE or JDK) installed and in your path before you install LucidWorks Search.

Supported Operating Systems:

- 32-bit and 64-bit versions of Windows XP, Windows Vista, Windows 7 and Windows Server 2003
- Linux-based operating systems with a 2.4 kernel or later
- Mac OS X 10.5+
- CentOS v.5.0+

Apache Solr runs in a Java servlet container and the LucidWorks Search installer automatically installs and configures Jetty, an open source Java servlet container. **At this time, Jetty is the only servlet container that can be used with LucidWorks.** LucidWorks Search has been designed around using Jetty as the container and it's not possible to run it in any other container.

Browsers

The LucidWorks Search Admin UI and Search UI require JavaScript to be enabled in the browser to function properly. LucidWorks Search has been tested on Internet Explorer version 8+, and has been tested on modern versions of Firefox, Safari, and Chrome.

Hardware Requirements:

- Minimum: 2GB RAM
- Recommended for 100,000 to 1,000,000 documents: 4GB-8GB RAM, 2GHz dual-core CPU
- Recommended for greater than 1,000,000 documents: 8GB-16GB RAM, 2.4GHz dual-core CPU

Disk Space:

Required disk space depends on the size and number of documents to be indexed. Contact LucidWorks for assistance in determining the disk space needed for your implementation.

Network Access:

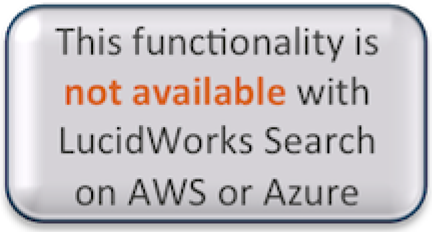
If the LucidWorks crawlers will be used to index documents over the network, the server where the LWE-Connectors component is installed must have access to the servers that host the content repositories to be crawled.

If opting-in to the LucidWorks System Usage Monitor, the LWE-Core component must be able to send outgoing encrypted POST requests to https://heartbeat.demo.lucidworks.io.

Related Topics

- Working With LucidWorks Search Components

Running the Installation Wizard


This functionality is not available with LucidWorks Search on AWS or Azure

Before installing LucidWorks, review the system requirements in the section on Installation.

To run the installation wizard, follow these steps:

1. Double-click the installation file (.JAR or .EXE). The Information screen appears.

   > If the installer does not open when you double-click it, open a command shell or prompt, make sure that Java 6 or greater is in your path, and launch the installer manually with the command `java -jar <file-name.jar>`.

2. Click **Next**. A list of prerequisites for installing LucidWorks Search appears.
3. Make sure your system meets the specified requirements, then click **Next**. The License Agreements screen appears.
4. Read the license. If you accept its terms, click the button that reads, "I accept the terms of this license agreement."
5. Click **Next**. The **Components to Enable** screen appears.

   The installer displays a list of LucidWorks Search components and their default addresses. We recommend that you install all components, unless you are working on a custom installation. See Working With LucidWorks Search Components for more information.

6. Configure the components dialog box to select the LucidWorks Search components and network addresses you want to install.

> ⚠ **Remove Default Addresses to Skip Components**
>
> If you choose not to install a component, be sure to uncheck the box next to the component name *and* remove its default address (or change it to the location where that component is installed). If the address is not removed or changed, the component will not be installed, but the default address will be entered into the `master.conf` configuration file, which will cause installed components to try to access the skipped component at that address.

7. Click **Next**. The Select Installation Path screen appears. Enter or browse to the directory where LucidWorks Search will be installed. It's best to choose a path that does not contain spaces (i.e., not a path like "c:\program files\" or similar). Paths with spaces will cause the crawlers to load improperly when LucidWorks starts. This will be the location of `$LWE_HOME`, which is referenced throughout this Guide when specifying file paths.
8. Click **Next**. The installer will ask you to confirm the installation location before proceeding.
9. Click **OK**. The LucidWorks System Monitor Screen appears. This screen allows you to opt-in to or opt-out of the LucidWorks System Usage Monitor program, which sends anonymous, encrypted data about your system to LucidWorks. A link on the screen will take you to the LucidWorks website for more information, or you can look in this documentation for the section on LucidWorks System Usage Monitor. Uncheck the box to opt-out of the program, or leave the box checked to opt-in.
10. Click **Next**. The Summary of Choices screen appears.
11. Confirm your installation choices, then click **Next**. The LucidWorks Search installation begins.
12. When the installation is finished, click **Next**. The Start LucidWorks Search screen appears. To start LucidWorks Search immediately, check the Start LucidWorks box. If installing in Windows, you will be prompted to start LucidWorks Search as a service. If you opt not to install as a service, the next screen will ask if you want to start LucidWorks Search.
13. Click **Next**. The installer initiates the LucidWorks Search start scripts. In most installations, these start quickly, but it may take up to one minute for the scripts to complete. The installer allows you to continue while the scripts work in the background.
14. Click **Next**.
15. As a final step, the installer can create an automated installation script that includes the settings you chose that you can use to run unattended installations. To generate an automated installation script, click Generate an automatic installation script. Otherwise, click **Done**.

You have now installed LucidWorks Search. If you accepted the default component locations, you can access the Administrative User Interface at http://localhost:8989/. Otherwise, you can find the Administrative User Interface at the URL and port you chose in step six. Refer to the README.txt file under the installation root directory for the default password. You can change the default password using the Users screen in the Admin UI or with an API call described on the Users API page.

Running the Installer in Console Mode

This functionality is **not available** with LucidWorks Search on AWS or Azure

If you are installing LucidWorks Search on a computer without a graphical user interface (i.e., a "headless" machine), you can run the installer in "console" mode.

1. Launch the installer with the command `java -jar <file-name.jar> -console`. (The name of your download may differ from the name displayed here.)

   ```
   bash-3.2$ java -jar lucidworks-search-installer-2.0.jar -console
   ```

2. Read through the license, then press '1' at the end to accept the license terms.

   ```
   ....
   You agree that this Agreement is the complete Agreement for the Programs and
   licenses, and this Agreement
   supersedes all prior or contemporaneous Agreements or representations. If any
   term of this Agreement is
   found to be invalid or unenforceable, the remaining provisions will remain
   effective. This Agreement is
   governed by the substantive and procedural laws of California. You and Lucid
   agree to submit to the exclusive
   jurisdiction of, and venue in, the courts of San Mateo county in California in
   any dispute arising out of or
   relating to this Agreement.
   press 1 to accept, 2 to reject, 3 to redisplay

   1
   ```

3. Select each component to install and choose the address and port each component will run on (this is multiple steps).

```
Please select which LWE components should be enabled on this server, or specify
the remote address of a component if it will run remotely:



  [x] Run LWE-Core Locally
input 1 to select, 0 to deselect:
1
Address [http://127.0.0.1:8888]

  [x] Run LWE-UI Locally
input 1 to select, 0 to deselect:
1
Address [http://127.0.0.1:8989]

Note: Components will communicate with each other using these addresses, and if
enabled on this machine, will run on the port given in the address.
Leave the address blank to keep a component from being used (locally or remotely)
press 1 to continue, 2 to quit, 3 to redisplay

1
```

4. After approving the components to install, select the directory location for the install. This will be the base path for `$LWE_HOME`, which is referenced throughout this Guide when discussing configuration and log file locations.

```
Select target path [/Users/cassandra4work/Downloads]
/Applications/LucidWorks/LucidWorksSearch
press 1 to continue, 2 to quit, 3 to redisplay
1
```

5. After defining the installation location, choose whether to opt-in or opt-out of the LucidWorks System Usage Monitor, which will send anonymous, encrypted statistics about your system to LucidWorks.

```
<html>LucidWorks Search product lets you share optional usage statistics with
LucidWorks to help us improve our products.
To find out more about this feature, including how to disable it, click
<a
href='http://www.lucidworks.com/lucidworks-system-usage-monitor'>here</a>.</html>

  [x] Enable LucidWorks System Usage Monitor
input 1 to select, 0 to deselect:
```

6. The installer installs LucidWorks Search, and asks if you want to start the servers.

```
[ Starting to unpack ]
[ Processing package: LucidWorks (1/1) ]
[ Unpacking finished ]


LucidWorks Search has been installed. Would you like to start LucidWorks Search
now?
NOTE: LucidWorks Search will be started in the background and may take some time
to complete loading.
You may continue on with the installer before it finishes loading.
Depending on your system hardware, LucidWorks Search may take some time to finish
loading even after the installer has been closed.
  [x] Start LucidWorks Search
input 1 to select, 0 to deseclect:
1
```

7. Finally, the installer displays confirmation of the successful installation.

```
Install was successeful
application installed on /Applications/LucidWorks/LucidWorksSearch
[ Console installation done ]
```

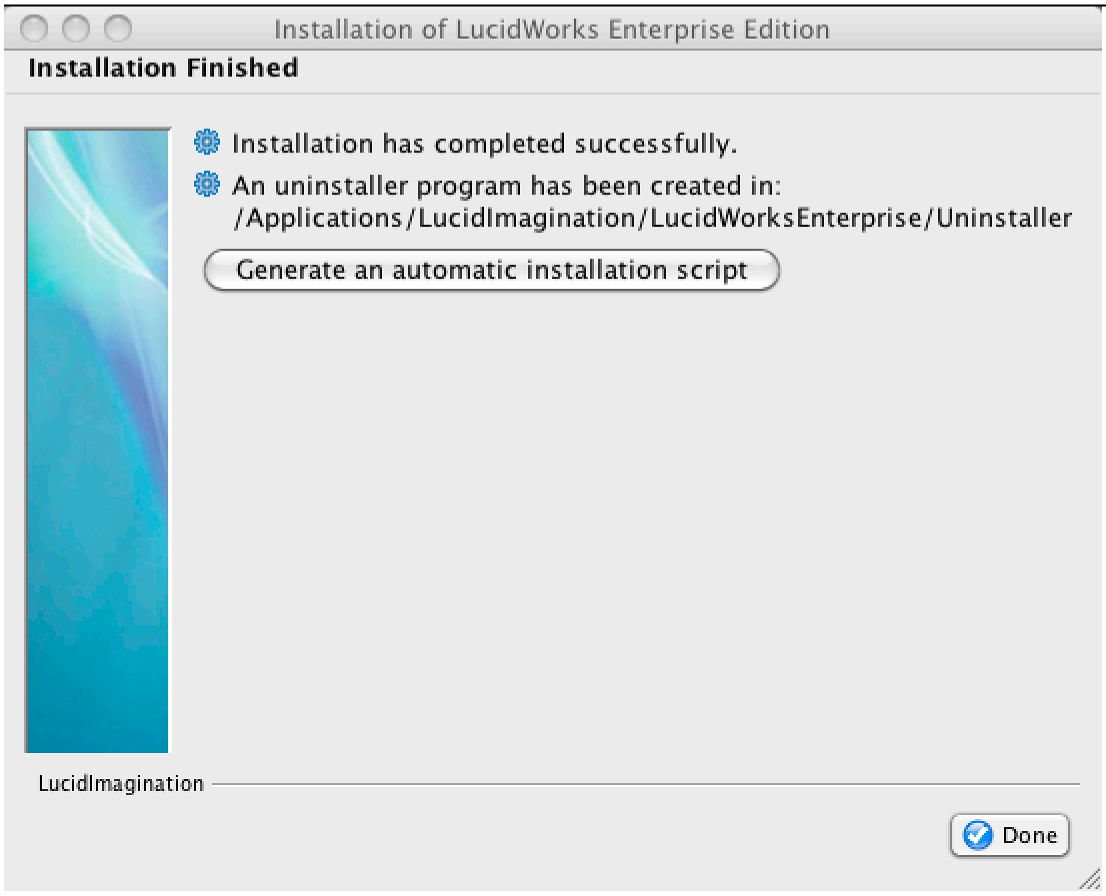
If you specified that the installer should start LucidWorks Search, you can access the Admin UI at the address you specified during the component configuration (the default is http://localhost:8989). A user named "admin" is created during the installation process with a default password that can be found in the README.txt file found under `$LWE_HOME/app`. The default password can be changed with either the User page in the Admin UI or the User API.

Automating Installation Options for Installation to Multiple Environments

This functionality is **not available** with LucidWorks Search on AWS or Azure

The last step of the graphical installer gives you the option of creating an automatic installation script. This script is an XML file that contains the configuration decisions that you made during the installation process. You can use this file to repeat the same installation in multiple environments, if needed.

This screen shows the option to create the script:


Installation Finished

Installation has completed successfully.

An uninstaller program has been created in:
/Applications/LucidImagination/LucidWorksEnterprise/Uninstaller

Installation of LucidWorks Enterprise Edition
Generate an automatic installation script
LucidImagination
Done

To generate this configuration file, click **Generate an automatic installation script**.

A dialog box will open prompting you to save the file in a location of your choosing. The default location is the same directory where LucidWorks Search has been installed. Enter a filename such as "config-options.xml," then press enter.

To use the installation script in future installations, run the installer from the command line with a command such as:

```
java -jar lucidworks-search-installer-2.0.jar config-options.xml
```

Generating an Installation Script by Hand

You can also create an installation script without using the graphical installer, although you cannot create it using the installer in console mode, by creating the XML file by hand. This is a sample script that can be used as a start; see below for details on what sections to edit to customize this script locally.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
<com.izforge.izpack.panels.HTMLHelloPanel id="UNKNOWN
(com.izforge.izpack.panels.HTMLHelloPanel)"/>
<com.izforge.izpack.panels.HTMLInfoPanel id="UNKNOWN
(com.izforge.izpack.panels.HTMLInfoPanel)"/>
<com.izforge.izpack.panels.LicencePanel id="UNKNOWN
(com.izforge.izpack.panels.LicencePanel)"/>
<com.izforge.izpack.panels.UserInputPanel id="enabled">
<userInput>
<entry key="EnableLWEUI" value="true"/>
<entry key="LWEUIAddress" value="http://127.0.0.1:8989"/>
<entry key="EnableLWECore" value="true"/>
<entry key="LweCoreAddress" value="http://127.0.0.1:8888"/>
</userInput>
</com.izforge.izpack.panels.UserInputPanel>
<com.lucid.izpack.StopLWPanel id="UNKNOWN (com.lucid.izpack.StopLWPanel)"/>
<com.lucid.izpack.CheckPortPanel id="UNKNOWN (com.lucid.izpack.CheckPortPanel)"/>
<com.lucid.izpack.FindLWPanel id="UNKNOWN (com.lucid.izpack.FindLWPanel)"/>
<com.lucid.izpack.StopLWPanel id="UNKNOWN (com.lucid.izpack.StopLWPanel)"/>
<com.lucid.izpack.CheckPortPanel id="UNKNOWN (com.lucid.izpack.CheckPortPanel)"/>
<com.izforge.izpack.panels.TargetPanel id="UNKNOWN
(com.izforge.izpack.panels.TargetPanel)">
<installpath>/Applications/LucidImagination/LucidWorksSearch</installpath>
</com.izforge.izpack.panels.TargetPanel>
<com.lucid.izpack.SummaryPanel id="UNKNOWN (com.lucid.izpack.SummaryPanel)"/>
<com.izforge.izpack.panels.InstallPanel id="UNKNOWN
(com.izforge.izpack.panels.InstallPanel)"/>
<com.izforge.izpack.panels.UserInputPanel id="startlwe">
<userInput>
<entry key="start.lw" value="true"/>
</userInput>
</com.izforge.izpack.panels.UserInputPanel>
<com.lucid.izpack.NoLogProcessPanel id="UNKNOWN (com.lucid.izpack.NoLogProcessPanel)"/>
<com.izforge.izpack.panels.ShortcutPanel id="UNKNOWN
(com.izforge.izpack.panels.ShortcutPanel)"/>
<com.izforge.izpack.panels.FinishPanel id="UNKNOWN
(com.izforge.izpack.panels.FinishPanel)"/>
</AutomatedInstallation>
```

There are three areas to edit:

Enable Components

This section defines where each component of LucidWorks Search is installed so they know how to talk to each other.

```
<userInput>
<entry key="EnableLWEUI" value="true"/>
<entry key="LWEUIAddress" value="http://127.0.0.1:8989"/>
<entry key="EnableLWECore" value="true"/>
<entry key="LweCoreAddress" value="http://127.0.0.1:8888"/>
</userInput>
```

To skip the installation of a component, set the "Enable" value to `false` and remove the URL in the associated "Address" value. See the section on Working With LucidWorks Search Components for more information about what each component does.

Set the Installation Path

This is the directory and path of the LucidWorks Search installation. Change to the proper path as needed.

```
<installpath>/Applications/LucidImagination/LucidWorksSearch</installpath>
```

Start LucidWorks Search after Script Completion

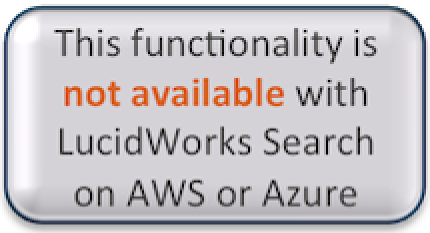
LucidWorks Search start scripts can be initiated immediately following completion of the installation. The default is `true`; if you do not want to initiate the start scripts immediately, change the value to `false`.

```
<userInput>
<entry key="start.lw" value="true"/>
</userInput>
```

Once you have made the appropriate edits, save the file with a name such as `config-options.xml`. To use it in future installations, run the installer from the command line with a command such as (be sure to replace our sample file name with the one created locally):

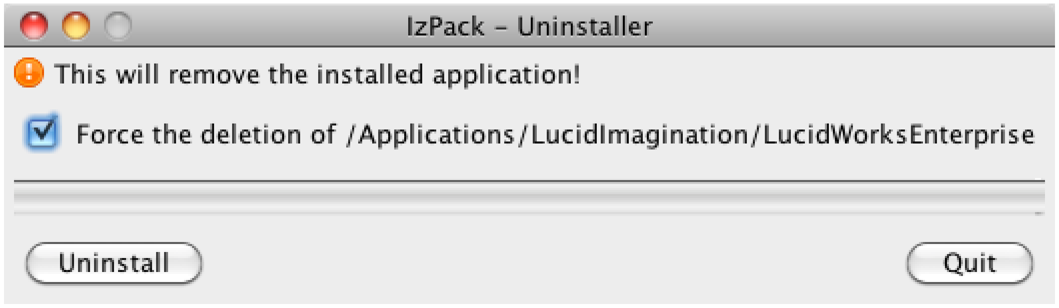
```
java -jar lucidworks-search-installer-2.0.jar config-options.xml
```

Uninstalling LucidWorks


This functionality is not available with LucidWorks Search on AWS or Azure

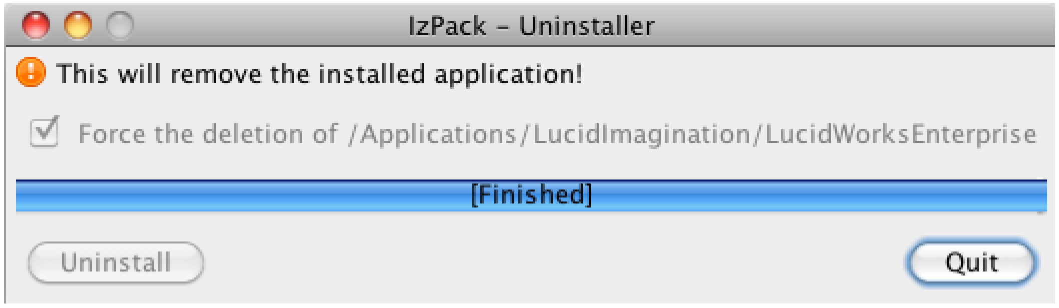
You can uninstall LucidWorks Search by running the uninstaller found in the `$LWE_HOME/app/uninstaller` directory. Two files are available: `uninstaller.exe` for Windows systems and `uninstaller.jar` for Linux and Mac systems.

1. Launch the appropriate file. The IzPack - Uninstaller dialog box appears:


IzPack - Uninstaller
This will remove the installed application!
Force the deletion of /Applications/LucidImagination/LucidWorksEnterprise
Uninstall
Quit

2. Select **Force the deletion of** *your/installation/directory* to remove the parent installation directory. If you do not force the deletion of the installation directory, the application will be removed but the installation directory will remain.
3. Click **Uninstall**.

   The uninstaller displays the progress bar.


IzPack - Uninstaller
This will remove the installed application!
Force the deletion of /Applications/LucidImagination/LucidWorksEnterprise
[Finished]
Uninstall
Quit

4. When the uninstallation is complete, click **Quit** to close the uninstaller.

Uninstall from the Command Line

You cannot run the uninstaller from the command line in console mode. To remove LucidWorks Search on a server without GUI access, stop all running LucidWorks processes, then manually delete the parent directory. This will remove all indexes and associated data.

Upgrading from a Prior Version

This functionality is **not available** with LucidWorks Search on AWS or Azure

The upgrade process for LucidWorks Search includes several steps to update the system configuration files, the search indexes, the LucidWorks software and any embedded software from third-party vendors (such as Solr and Lucene, etc.).

It's recommended that the process described below be performed on a test system before upgrading your production application.

Supported Versions

The upgrade process described in this section only applies to upgrades within v2.x (including v2.0, v2.0.1, v2.1, and v2.1.1). If you are using v1.7 or v1.8 of LucidWorks, and would like to move to v2.1, you will first need to move to v2.0. Contact Lucid Imagination Support for help with an upgrade from v1.x to v2.x.

If you are using v2.0.x, you can upgrade directly to v2.1.x or v2.5.

In version 2.0 of LucidWorks Search, we used a process for migrating from v1.7 or v1.8 to v2.0 that used the LucidWorks REST API to copy configuration information from the older version of LucidWorks and import it into the new version of LucidWorks. That process was been completely replaced in v2.1. The current process upgrades the older version in place. A parallel installation of LucidWorks is made to get the new code, then a script is run to update the existing configuration files.

Upgrade Steps

Upgrading LucidWorks requires the following steps:

- Preparation: Review the Upgrade Notes
- Step 1: Stop Existing Installation and Create a Backup
- Step 2: Install the New Version of LucidWorks
- Step 3: Run the MetadataUpgradeTool
- Step 4: Upgrade the Indexes or Delete Them
- Step 5: Update the `$LWE_HOME/app` dir
- Step 6: Make Manual Changes to Configuration Files
- Step 7: Start LucidWorks

Preparation: Review the Upgrade Notes

The following instructions contain step-by-step information about the upgrade process. There may, however, be specific changes for each release and for your specific implementation that need to be considered before, during, or after the upgrade. Before starting any upgrade, please review the list of changes for your version in the section Upgrade Notes.

Step 1: Stop Existing Installation and Create a Backup

LucidWorks Search can be stopped with the scripts available in `$LWE_HOME/app/bin`. Use either `stop.sh` or `stop.bat` depending on your operating system.

A backup of your existing installation can be made by copying the entire $LWE_HOME directory to a parallel location. If your installation has been made in a directory called `/usr/local/LucidWorksEnterprise`, then in Unix, for example, the command would be:

```
cp -r /usr/local/LucidWorksEnterprise /tmp/lwe-old-backup
```

You can name the backup whatever you'd like, as long as you remember it later if you need it.

Now is also a good time to make note of any customizations that will be over-ridden by the upgrade, if you haven't done so already. For details of these changes, see the Upgrade Notes for your version.

Step 2: Install the New Version of LucidWorks

Follow the steps outlined in the sections on installing LucidWorks to install the new version of LucidWorks. This can be done in any location on your server, but you should change the default path if your existing LucidWorks is already installed there. You can choose the same ports as are used in your existing installation: this installation will only be used to get an updated `app` directory, the MetadataUpgradeTool, and the IndexUpgradeTool (See Step 3 and Step 4 below).

Do not install the new version of LucidWorks Search over your existing installation!

When the installer asks if you would like to start the new version of LucidWorks, say no (uncheck the box).

The instructions from here on will refer to the top level of this installation as `$LWE_NEW`.

Step 3: Run the MetadataUpgradeTool

The MetadataUpgradeTool is found in the `$LWE_NEW/app/migration_tools` directory. It will update your configuration files to the proper format for the new version and will also move the user database from the `app` dir to the `data` dir if it still exists in the old location.

To run it, issue this command:

```
$ java -jar $LWE_NEW/app/migration_tools/MetadataUpgradeTool.jar -verbose -lwe_conf_dir
<existing conf dir>
-lwe_app_dir <existing app dir> -lwe_data_dir <existing data dir>
```

The parameters `-lwe_conf_dir`, `-lwe_app_dir`, and `-lwe_data_dir` are required and should point to the `$LWE_HOME/conf`, `$LWE_HOME/app` and `$LWE_HOME/data` directories, respectively, of your existing LucidWorks application.

The parameter `-verbose` is optional, but highly recommended. It allows the full output of the tool to show on-screen in case there are problems or failures. If support is needed after using the MetadataUpgradeTool, Lucid Imagination will request this output. The output will look something like:

```
### Upgrading format of collections.yml ...
Writing upgraded collections.yml
### Upgrading similarity configuration in Solr schema ...
Upgrading similarity configuration in
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/collection1_0/conf/schema.
upgraded
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/collection1_0/conf/schema.
similarity configuration in
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/LucidWorksLogs/conf/schema
upgraded
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/LucidWorksLogs/conf/schema
Upgrading field mapping configuration in solrconfig ...
Processing
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/collection1_0/conf/solrcon
...
Writing upgraded
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/collection1_0/conf/solrcon
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/LucidWorksLogs/conf/solrco
...
Writing upgraded
/Applications/LucidImagination/lwe2.0.1-upgrade/conf/solr/cores/LucidWorksLogs/conf/solrco
Removing old autocomplete data files...
### Upgrading UI DB location...
Moved
/Applications/LucidImagination/lwe2.0.1-upgrade/app/webapps/lwe-ui/WEB-INF/db/production.s
/Applications/LucidImagination/lwe2.0.1-upgrade/data/lwe-ui
### Upgrading UI DB...
Executing UI DB upgrade
==  AddExternalProtocolToSettings: migrating ==================================
-- change_table(:settings)
   -> 0.0060s
==  AddExternalProtocolToSettings: migrated (0.0070s) =========================
```

If you have created any collection templates they **will not** be upgraded with the MetadataUpgradeTool. It's recommended that all templates created with previous versions be recreated from scratch using a new or upgraded collection.

One of the changes from v2.0 to v2.1 was to change the names of the data source types "S3" and "S3N". The "S3N" type, for Amazon over S3, is now known as "S3". The former "S3" type, for a Hadoop Filesystem on S3, is now known as "S3H". During the upgrade process, these types will be converted automatically.

However, there is a new requirement for these types of data sources that the value for the URL must include a trailing slash if the URL points to a directory of files and not to a single resource. This is not done automatically by the upgrade. In order to successfully crawl an existing "S3" or "S3H" data source after upgrading, the trailing slash must be added manually if the path specified is not to a single file or resource.

Step 4: Upgrade the Indexes or Delete Them

The MetadataUpgradeTool only upgrades configuration and other system files. However, the search indexes also need to be upgraded as a separate step. If, however, the indexed content is not needed, the indexes can be simply deleted instead of upgraded.

Upgrading the Indexes

In order to upgrade an index from an earlier version of LucidWorks, the Index Upgrade Tool should be run. This tool is found under `$LWE_HOME/app/migration-tools/` in a .jar file called `IndexUpgradeTool.jar`.

It is recommended to run this Upgrade Tool only after consultation with Lucid Imagination Support to be sure you understand the full ramifications of running this tool on your local, customized, index.

While we have provided this index upgrade tool to help you avoid re-indexing your data, it is recommended to do so with every migration to a new version.

Before running the tool, you should make sure LucidWorks is not running to ensure there are no indexing processes taking place while performing the upgrade.

To run the tool, open a command line interface, navigate to the `$LWE_NEW/app/migration_tools` directory and issue the command:

```
$ java -jar IndexUpgradeTool.jar [options] <sourcedir> <destinationdir>
```

The `<sourcedir>` parameter is the existing index that will be upgraded and moved to the `<destinationdir>`. Unlike the MetadataUpgradeTool, which updates configuration files in place, the IndexUpgradeTool makes a copy of the index while upgrading it. The `<sourcedir>` is `$LWE_HOME/data/solr/cores/collection/data/index` (where *collection* is the name of the collection to be upgraded. The `destinationdir` should be a temporary location (preferably a directory, such as /tmp/index-upgrade, as the output is a number of raw index files).

The IndexUpgradeTool can upgrade the index for one collection at a time. If there are multiple collection in the origin LucidWorks installation, these would each need to be converted separately. By default, there are at least two indexes: the one that contains your data (called *collection1* at initial installation) and one that indexes log data called LucidWorksLogs. Don't forget to upgrade the LucidWorksLogs collection or delete the index - skipping that collection will cause LucidWorks to fail on restart.

There is currently one option that can be used with the IndexUpgradeTool, which is `-checkindex`. This will run Lucene's checkindex tool to validate that the index is correct. This is a good idea to do, especially for systems already in production, but will add time to the upgrade process.

The output of the tool is a number of index files in the location you specified with the `<destinationdir>` parameter. Once the tool has finished, delete the existing index files for each collection (found under `$LWE_HOME/data/solr/cores/collection/data/index`) and copy the new index files for each collection to the `index` dir.

Deleting the Indexes

If the index files are not particularly needed, which may be the case for the LucidWorksLogs collection particularly, they can be deleted. To do so, delete all of the subdirectories found under `$LWE_HOME/data/solr/cores/collection/` for each collection.

Step 5: Update the $LWE_HOME/app dir

Neither the MetadataUpgradeTool nor the IndexUpgradeTool update the program files that make up the LucidWorks application. These need to be updated manually by replacing the `app` directory in the original installation with the `app` directory from the new LucidWorks Search installation. To do this, delete the original `$LWE_HOME/app` and copy the `$LWE_NEW/app` directory to `$LWE_HOME`. Assuming default locations of the installations, the sequence would be:

```
$ rm -r /LucidWorks/LucidWorksSearch/app
$ cp -r /LucidWorks/LWS-newVersion/app /LucidWorks/LucidWorksSearch
```

Step 6: Make Manual Changes to Configuration Files

Update `master.conf`

The upgrade process does not add two parameters to the `master.conf` file found in `$LWE_HOME/conf/`. These changes are recommended for optimal performance in typical situations. They are not added automatically by the MetadataUpgradeTool because these or similar changes may have already been made in your implementation depending on your specific situation. Both parameters are entered in the section of the file for *JVM Settings for LWE-UI*:

- Add `-XX:MaxPermSize=256M`
- Add `-Dcom.sun.management.jmxremote`

Depending on the 2.x version you started with, these parameters may already be correct.

Other Changes

Other manual changes may be required depending on the version and your own implementation. Review the Upgrade Notes for your version for details.

Step 7: Start LucidWorks

Start LucidWorks Search using the scripts available in `$LWE_HOME/app/bin`. Use either `start.sh` or `start.bat` depending on your operating system.

After the upgrade, the indexes that drive autocomplete and spell check will have been removed. As described in the section on Spell Check, the spell check indexes will be automatically regenerated by default. The autocomplete indexes are not automatically generated unless there is already an autocomplete activity scheduled (and it will happen on its schedule). If using autocomplete in your search application, and it is not scheduled to run periodically, use the Activity API or the Admin UI to manually kick it off or schedule it.

After checking to be sure the upgrade was successful, the `$LWE_NEW` installation and/or the backup can be deleted. In order to be sure the upgrade was successful, try to index some content and perform some queries, while checking for errors in the "core" log found in `$LWE_HOME/data/logs`.

Related Topics

- Upgrade Notes
- Release Notes

Migrating from Solr to LucidWorks

It is possible to move an existing Solr 3.x implementation to LucidWorks to use some of the LucidWorks features. Many of the LucidWorks features are configured in standard Solr configuration files, and LucidWorks is generally able to add those configuration details to configuration files as they are enabled during implementation of LucidWorks. Examples of this are Click Scoring, which has several components that will be added programmatically when it is properly enabled.

Migrating from any Solr release prior to v3.x (such as 1.4.1) is **not possible** at this time. To move to LucidWorks from Solr 1.4.1 or earlier, you must first migrate to Solr 3.x. The technique to do that is out of scope for this document, but some assistance may be found in a LucidWorks blog post by Chris Hostetter on upgrading his Solr 1.4 instance to 3.1.

In addition, these instructions assume you have a good deal of experience using Solr's configurations, and that you understand the concepts of RequestHandlers, UpdateHandlers and update chains.

Information for LucidWorks Search in the Cloud Users
Customers using Solr who would like to migrate to LucidWorks Search on AWS or Azure need only contact LucidWorks Support for assistance in making the move.

The following items need to be added to the `schema.xml` and `solrconfig.xml` files for each collection:

An existing Solr `schema.xml` and `solrconfig.xml` file should be used as the starting point, but LucidWorks-specific update handlers, request handlers and update chains must be added for a seamless experience. Once these items are added, LucidWorks features that require configuration in `solrconfig.xml` or `schema.xml`, such as Click Scoring, de-duplication, distributed updates (pre-SolrCloud), search filtering, field mapping with built-in crawlers, and some query parser parameters, will be added to those configuration files automatically when they are enabled.

However, configuration definitions for the auto-complete and spell check components are not added. If they are not already in implemented in your Solr configuration, and you do not add them during this process (or at any point by manually editing `solrconfig.xml`), you **will not be able to enable them using the LucidWorks UI or API**. The components can be added manually at any time; once they are added, LucidWorks can be used to manage them (disable them). If they are not added manually, the Admin UI will warn users who try to use it that the components are not defined, so the feature cannot be enabled.

If it's expected that several collections in LucidWorks will be created based on an existing Solr configuration, it may be useful to modify the configuration of a single collection (such as the default collection1), then use that collection to create a template that will be used for other collections. For more on Collection Templates, see the section Using Collection Templates.

Add the /lucid RequestHandler

The `/lucid` RequestHandler is referenced with several of the features of LucidWorks, and must be added to `solrconfig.xml` for each collection. It is a Solr search request handler that is manageable by LucidWorks.

```
<requestHandler class="solr.StandardRequestHandler" name="/lucid" default="true">
    <lst name="defaults">
      <str name="defType">dismax</str>
      <str name="q.alt">*:*</str>
      <str name="qf">title^5.0 body</str>
      <str name="defOp">AND</str>
      <str name="fl">id,title,body,data_source_type,data_source_name,data_source</str>
    </lst>
  </requestHandler>
```

Add the lucid-update-chain Update Request Processor Chain

LucidWorks updates the update chain if you enable some functionality. For example, a signature processor is added if you enable de-duplication. Similarly, a field mapping processor is added if you use the crawlers included with LucidWorks instead of Solr-specific document indexing techniques.

The following must be added to `solrconfig.xml` for each collection in order for these features to be properly enabled.

```
<updateRequestProcessorChain name="lucid-update-chain">
    <processor class="solr.LogUpdateProcessorFactory">
      <int name="maxNumToLog">10</int>
    </processor>
    <processor class="solr.RunUpdateProcessorFactory"/>
  </updateRequestProcessorChain>
```

Add /update RequestHandler

The `/update` RequestHandler is a standard Solr update request handler and should be configured to use the `lucid-update-chain`. Add the following to the `solrconfig.xml` for each collection:

```
<requestHandler class="solr.XmlUpdateRequestHandler" name="/update">
    <lst name="defaults">
      <str name="update.chain">lucid-update-chain</str>
    </lst>
  </requestHandler>
```

Add Required Fields

LucidWorks indexing and searching depends on several fields being configured in the `schema.xml` for each collection. Add the following:

```
<types>
    <fieldType name="string" class="solr.StrField"/>
    <fieldType name="date" class="solr.TrieDateField" omitNorms="true"
precisionStep="0" positionIncrementGap="0"/>
    <fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
      <analyzer type="index">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter catenateAll="0" catenateNumbers="1" catenateWords="1"
class="solr.WordDelimiterFilterFactory" generateNumberParts="1"
           generateWordParts="1" splitOnCaseChange="1"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.ASCIIFoldingFilterFactory"/>
      </analyzer>
      <analyzer type="query">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter catenateAll="0" catenateNumbers="0" catenateWords="0"
class="solr.WordDelimiterFilterFactory" generateNumberParts="1"
           generateWordParts="1" splitOnCaseChange="1"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.ASCIIFoldingFilterFactory"/>
      </analyzer>
    </fieldType>
 </types>
 <fields>
   <dynamicField name="attr_*" type="string" indexed="true" stored="true"
multiValued="true" />

    <field indexed="true" multiValued="false" name="id" omitNorms="true" stored="true"
type="string"/>
    <field indexed="true" multiValued="false" name="data_source_type" omitNorms="true"
stored="true" type="string" omitTermFreqAndPositions="true"/>
    <field indexed="true" multiValued="false" name="data_source_name" omitNorms="true"
stored="true" type="string" omitTermFreqAndPositions="true"/>
    <field indexed="true" multiValued="false" name="data_source" omitNorms="true"
stored="true" type="string"/>
    <field default="NOW" indexed="true" multiValued="false" name="timestamp"
omitNorms="true" stored="true" type="date"/>
    <field indexed="true" multiValued="true" name="text_all" omitNorms="false"
stored="false" type="text_en"/>
    <field indexed="true" multiValued="false" name="batch_id" omitNorms="true"
stored="true" type="string"/>
    <field indexed="true" multiValued="true" name="title" omitNorms="false"
stored="true" type="text_en"/>
    <field indexed="true" multiValued="true" name="body" omitNorms="false"
stored="true" type="text_en"/>
 </fields>
 <uniqueKey>id</uniqueKey>
 <defaultSearchField>text_all</defaultSearchField>
```

Working with Existing Solr Indexes

The indexes in your Solr instance will not work with LucidWorks automatically. There have been changes to the index codecs which require deciding if you will reindex your content from scratch or if you will upgrade the indexes and continue to work with your existing content upload strategy.

To figure out which option works best for you, it helps to understand a little bit about how LucidWorks stores documents and deals with data sources.

First, LucidWorks is using Solr 4, also known as trunk, which contains significant changes to index format. At a minimum, you will need to upgrade your indexes.

In addition, LucidWorks includes several embedded crawlers which are designed to acquire content and add it to the index. These crawlers are configured through the use of data sources, which include information about individual content repositories and how to access them. All of the content added to the index via data sources includes information about what data source added the document to the index, which is used for statistics and other features of the Admin UI. The content added to your existing Solr indexes will not have any of this information, unless you happened to add it, but then it likely will not be stored in fields which the LucidWorks Admin UI is able to understand (for example, the ID of a data source is stored in the `data_source` field, which is unlikely to exist in your existing content or index).

Re-indexing All Documents

This is the recommended approach because it is the least prone to problems going forward. This approach requires deleting your entire Solr indexes and re-indexing the content. Re-indexing the content can be done with the LucidWorks Admin UI or Data Sources API, or with using the same mechanisms used to index content in Solr. If choosing the same mechanisms as used with Solr, LucidWorks provides the option to create "External" data sources, which allow pushing content with your code directly to Solr in a way that links the incoming data with the External data source so the Admin UI and APIs can be fully utilized as though the embedded crawlers discovered the content and indexed it. For more information on External Crawlers, please review the section on Suggestions for External Data Sources.

Upgrading the Indexes

Upgrading the indexes is an alternative option, but is more prone to risk and may limit your use of all the LucidWorks features. LucidWorks includes an IndexUpgradeTool, which updates the format of the indexes. The IndexUpgradeTool **only works on Solr 3.x or higher indexes**. After using this tool, you can continue to use LucidWorks as a shell around Solr (accessing Solr directly as you are accustomed), but statistics about the content of your indexes will not be fully integrated with the LucidWorks Admin UI. This option may be a consideration for those who have a great deal of content in their indexes and for whom re-indexing will take a long period of time or is otherwise not an option.

This tool is found under `$LWE_HOME/app/migration-tools/` in a .jar file called `IndexUpgradeTool.jar`.

It is recommended to run this Upgrade Tool only after consultation with LucidWorks Support to be sure you understand the full ramifications of running this tool on your local index.

While we have provided this index upgrade tool to help you avoid re-indexing your data, it is recommended to do so with every migration to a new version.

Before running the tool, you should make sure LucidWorks Search is not running and ensure there are no indexing processes taking place while performing the upgrade.

To run the tool, open a command line interface, navigate to the `$LWE_NEW/app/migration_tools` directory and issue the command:

```
$ java -jar IndexUpgradeTool.jar [options] <sourcedir> <destinationdir>
```

The `<sourcedir>` parameter is the existing index that will be upgraded and moved to the `<destinationdir>`. Unlike the MetadataUpgradeTool, which updates configuration files in place, the IndexUpgradeTool makes a copy of the index while upgrading it. The `<sourcedir>` is the location of the current indexes to be upgraded. The `destinationdir` should be a temporary location (preferably a directory, such as /tmp/index-upgrade), as the output is a number of raw index files.

The IndexUpgradeTool can upgrade the index for one Solr core at a time. If there are multiple cores in the origin Solr, these would each need to be converted separately.

There is currently one option that can be used with the tool, which is `-checkindex`. This will run Lucene's `check index` tool to validate that the index is correct. This is a good idea to do, especially for systems already in production, but will add time to the upgrade process.

The output of the tool is a number of index files in the location you specified with the `<destinationdir>` parameter. Once the tool has finished, copy the new index files to the appropriate location for each upgraded core (LucidWorks cores are called collections, and are found under `$LWE_HOME/data/solr/cores/collection/data/index`).

Upgrade Notes

Upgrade Notes briefly describe the changes made to the software for each release.

The Upgrade Notes highlight major changes to LucidWorks Search or Solr that may impact a large number of users. If you have made customizations to your configurations and aren't sure of the impact of those customizations on your ability to upgrade, please consult with your support representative or send an email to cases@lucidworks.com

Upgrade Notes for v2.5

With v2.5, LucidWorks Enterprise and the LucidWorks Search Platform are now known as LucidWorks Search. This is a name change only and the documentation has been updated to reflect that change. If upgrading from a prior version, and you see LucidWorks Enterprise in the product, you can use the same documentation.

These upgrade notes apply to v2.5.0 and v2.5.1.

Before Upgrading

1. LucidWorks Search v2.5 contains an upgrade to Jetty, from Jetty 6 to Jetty 8. Jetty6 was included in earlier versions and has very different configuration files. If the MetadataUpgradeTool detects that your existing instance of LucidWorks contains Jetty6 configuration files, it will completely replace them with the defaults from the version of LucidWorks being installed. If you have made changes manually to any of your existing jetty configs, including enabling SSL, you will need to manually re-apply those changes after running the MetadataUpgradeTool.
2. It is likely that the `solr.xml` file for each collection contains absolute paths to collections that you previously created using paths relative to your `${lucidworksDataDir}`. You can edit the `solr.xml` to make these paths use the `${lucidworksDataDir}` variable instead. All collections created in the future will use this variable automatically. If `solr.xml` contains absolute path to the `data` directory and indexes, and the installation is moved or copied (say, for a backup), LucidWorks will not be able to find the indexes for each collection and will not start.

3. Earlier versions of Solr contained sections in the `solrconfig.xml` file for `mainIndex` and `indexDefaults`. Those have been replaced with a new section called `indexConfig`. During the upgrade, all settings in the `mainIndex` section will be used as the basis for a new `indexConfig` section. The old `indexDefaults` section, however, will be removed. If there are any custom settings in the `indexDefaults` section, you should make note of them before starting the upgrade so you can re-insert them in the new `indexConfig` section.
4. As with all upgrades, any configuration files contained inside a collection template will not be upgraded. If you have created collection templates, they will need to be re-created based on an upgraded collection or modified manually as defined in these notes. Configuration files inside existing collections that were made with a template, however, will be upgraded (meaning, the fact that a collection was made with a template does not impact its ability to be upgraded).

System Changes

- A new service called the LucidWorks System Usage Monitor has been included in this upgrade, which allows you to send anonymous, encrypted statistics about your system to LucidWorks.
- It is now possible to run LucidWorks as a service on Windows.
- New log files have been added to log requests between components.

Related Documentation

- LucidWorks System Usage Monitor
- System Directories and Logs
- Installation

Data Source Changes

- The crawler framework has been redesigned so it now runs in it's own JVM. It can be separated to its own server or cluster node if desired.
- Deleting a data source now deletes documents from the data source. It previously did not delete documents. Documents can be retained with a parameter in the delete request.
- The Windows Shares and SharePoint data source types now support multiple new attributes for controlling document security. Due to this, it's now possible to configure access control list (ACL) filtering on a per-data source basis, and for SharePoint data sources.
- Crawlers for MongoDB and MapR have been added to LucidWorks Search.
- Since the SharePoint data source uses the Google Connector Manager (GCM) framework, it is now possible to add additional connectors from GCM.
- Due to changes in the crawler framework, data source IDs are now random 32-character UUID strings (similar to "cd38e3bb181548a1ae7d57f624a44fd1"). If upgrading, the old IDs will be retained, but new data sources will get the new style of ID.

Related Documentation

- Working With LucidWorks Search Components
- Windows Shares Data Sources or Create a New Windows Share data source screen
- SharePoint Data Sources API or Create a New SharePoint data source screen
- MongoDB Data Sources API or Create a New MongoDB data source screen
- MapR Data Sources API or Create a New MapR data source screen
- Integrating Google Connectors

API Changes

- A new API allows managing the Field Mapping settings of data sources.
- A new API allows checking the status of the connector JVM.
- The Settings API now supports setting the `openSearcher` for hard commits.
- The Settings API now supports excluding documents with the `elevations` parameter to define documents that should not appear for specific queries.

Related Documentation

- Field Mapping API
- Crawler Status API
- Settings API

UI Changes

- When running LucidWorks Search in SolrCloud mode, the Admin UI will display the status of each node in the cluster on the main UI Dashboard page.
- The UI can now be used with screen readers.
- By default, all search results will include Solr's "explain" information.
- The Search UI can now support excluding documents from queries in addition to boosting defined documents.
- The ACL Filtering screen has been removed in favor of configuring access control list filtering for each Windows Share and/or SharePoint data source configuration. If you had ACL Filtering configured, those settings will be retained but can only be modified with the Search Handler Components API.

Related Documentation

- Search UI
- Explain Info in search results

Solr Update

The embedded Solr in LucidWorks Search has been updated to include the recently released Solr v4.0. In addition, we have included the following bug fixes and patches:

- SOLR-3861: Refactor SolrCoreState so it's managed by SolrCore.
- SOLR-3897: Preserve multi-value fields during hit highlighting.

- SOLR-3932: SolrCmdDistributorTest either takes 3 seconds or 3 minutes.
- SOLR-3933: Distributed commits are not guaranteed to be ordered within a request.
- SOLR-3938: PrepareCommit command omits commitData causing a failure to trigger replication to slaves.
- SOLR-3939: An empty or just replicated index cannot become the leader of a shard after a leader goes down.
- SOLR-3961: Fixed error with LimitTokenCountFilterFactory.
- SOLR-3971: A collection that is created with numShards=1 turns into a numShards=2 collection after starting up a second core and not specifying numShards.
- SOLR-3981: Fixed bug that resulted in document boosts being compounded in <copyField/> destination fields.
- SOLR-3988: Fixed SolrTestCaseJ4.adoc(SolrInputDocument) to respect field and document boosts.
- SOLR-3992: QuerySenderListener doesn't populate document cache.
- SOLR-3994: Create more extensive tests around unloading cores.
- SOLR-3995: Recovery may never finish on SolrCore shutdown if the last reference to a SolrCore is closed by the recovery process.
- SOLR-3998: Atomic update on uniqueKey field itself causes duplicate document.
- SOLR-4005: If CoreContainer fails to register a created core, it should close it.
- SOLR-4009: OverseerCollectionProcessor is not resilient to many error conditions and can stop running on errors.

For more information about the integration of LucidWorks and Solr, see the section Solr Direct Access.

Release Notes

Release notes describe known issues at the time each release is completed. The same information can be found in an on-premise installation of LucidWorks Search in the `$LWE_HOME/app` directory as a file called `RELEASE_NOTES.txt`.

Release Notes for v2.5

Issues at Release

Please see the `RELEASE_NOTES.txt` file found in the `app` directory of your LucidWorks Search installation for the most up to date list of known issues.

Installer

1. The console mode of the installer has a bug inherited from IZPack: it will create a corrupted uninstaller. To uninstall after using the console mode, delete the entire directory where LucidWorks was installed.

2. On Mac OS, if an installation path of `~/opt` is chosen and a new folder created, the installer will add an additional 'opt' to the end of the path. It is safe to simply remove the additional 'opt' and continue.
3. The Windows uninstaller may fail to remove the entire LucidWorksSearch directory, but will claim that it did so successfully.
4. When doing an automated install with a saved options file, the installer option to start up LucidWorks will not be respected.

LucidWorks Search

1. The web crawler will not properly handle character set encodings on web pages if the character set is defined in the `<meta>` section of the HTML page. If the character set is defined in the HTTP header, the encoding will be correctly recognized and the pages will be properly parsed. This problem only occurs for Web data sources.
2. It is not yet possible to create multiple schedules for a data source (of any type).
3. JVM memory settings should be adjusted as indexes grow in size. Of course, adjusting the memory should only be done on a system with enough memory to handle the increase (i.e., you should not try to increase memory to 4GB on a machine with only 2Gb available). Increase JVM memory by editing the JVM settings section in `conf/master.conf`. Please also contact LucidWorks for assistance in properly sizing your server.
4. The status column on the indexing/settings page may indicate that an activity has been skipped when in reality its execution has been temporarily delayed. In certain cases, the scheduler will queue a job for execution as soon as is feasible instead of executing it immediately. This condition is temporary; once the job starts, the status will be reported correctly.
5. Emptying a collection and all crawl data (either via the UI or using the REST API) may not leave the data directory for the collection completely empty. In some cases crawl data will be logically marked as unusable, but will remaining on disk until overwritten. To completely reclaim all disk space from a collection, you can delete the entire collection.
6. The ability to display files directly through the search UI has been removed pending a more complete implementation.
7. It is not possible to access the search UI as an anonymous user (one who is not logged in).
8. If there are errors while deleting fields, they are not displayed in the Fields screen of the Admin UI.
9. The setting to add failed documents likely will not add failed documents if the option is enabled after an initial crawl has been run. The reason this happens is because LucidWorks crawlers store documents they have seen before (also known as persistent crawl data or crawl history) and generally skip documents they have seen on previous crawls if they are unchanged. To force the crawler to try to process a document again, either clear the crawl history (which will cause all documents to be considered "new" again) or update the document so it appears to the crawler to have been changed.
10. After upgrading, and in some cases after a restart, the lucid.gcm crawler (for SharePoint data sources) may start crawling before LucidWorks has finished starting completely, which will lead to an exception in the logs. The error is harmless; once LucidWorks has started the lucid.gcm crawler will resubmit the crawled content.

11. When using traditional distributed indexing (not SolrCloud functionality) with a SolrXML data source type, the crawl may never end without a restart of LucidWorks. There are two possible solutions to this problem: a) Switch to using SolrCloud, or b) disable the `DistributedUpdateProcessor` on all but the master node.
12. When using the Admin UI to configure search filters, you may need to log out of the UI and then back in to see your changes take effect.
13. Data Source definitions and history are not stored in ZooKeeper when running LucidWorks Search in SolrCloud mode.
14. Due to a bug in Solr, if the installation path includes spaces, the paths displayed in the Solr Admin UI will be incorrect.
15. The High Volume HDFS crawler has not yet been updated for LucidWorks Search v2.5 and may not function as expected.
16. It is not possible to implement SSL with the LWE-Connectors component.
17. It is not possible to use a self-signed certificate with the LWE-Core component, because there is not yet any way to specify client certificate and trustStore for the LWE-Connectors component.
18. It is not possible to use mutual authentication with the LWE-Core component because there is no way to provide a client certificate for either the LWE-Connectors or the LWE-UI components.
19. Occasionally, using the "Find Similar" functionality in the LucidWorks Search search UI will produce a blank page and/or an error about a script taking too long. This is caused by calculations required for the "explain" info that is now displayed by default for every search result.
20. In some cases, recreating the LucidWorksLogs collection may fail. If you expect to use the LucidWorksLogs collection at a later time, the best approach is to temporarily suspend log indexing by modifying it's schedule.

LucidWorks Search User Interface Help

Help for the LucidWorks Search User Interface is located at http://lucidworks.lucidimagination.com/display/help.

System Configuration Guide

The System Configuration Guide provides detailed information about many of the features included with LucidWorks Search. It describes the layout of a LucidWorks Search installation and how to work with many of the configuration options included with the system. It contains the following sections:

- Understanding LucidWorks Search: Introduction, location of logs, working with components
- Collections and Indexes: Setting up collections, designing the index structure
- Crawling Content: Crawling content of different filetypes and in different repositories
- Query and Search Configuration: Configuring the user experience and how to get search results to your application
- Security and User Management: SSL communication between components and user authentication
- Solr Direct Access: Using Solr
- Performance Tips: How to judge performance and strategies for improvement
- Expanding Capacity: SolrCloud, index replication and distributed search
- Integrating Monitoring Services: Using JMX, MBeans, and integrating with Zabbix or Nagios

Information for LucidWorks Search in the Cloud Users
While nearly all of the features described in this section are available to LucidWorks Search customers hosted on AWS or Azure, some of the advanced configuration options are not. When editing a setting requires direct access to a configuration file, instead of accessing the setting via the UI or an API, contact your support representative for information about how you might tweak that setting for your needs.

Understanding LucidWorks Search

This section covers the architecture of LucidWorks Search and nitty-gritty details like where log files and important directories can be found.

We also cover some introductory material: if you're not familiar with search engines, there's a section How Search Engines Work and we continue that with some more information about How LucidWorks Search Works.

Then we get into the details with these sections:

- Working With LucidWorks Search Components
- System Directories and Logs

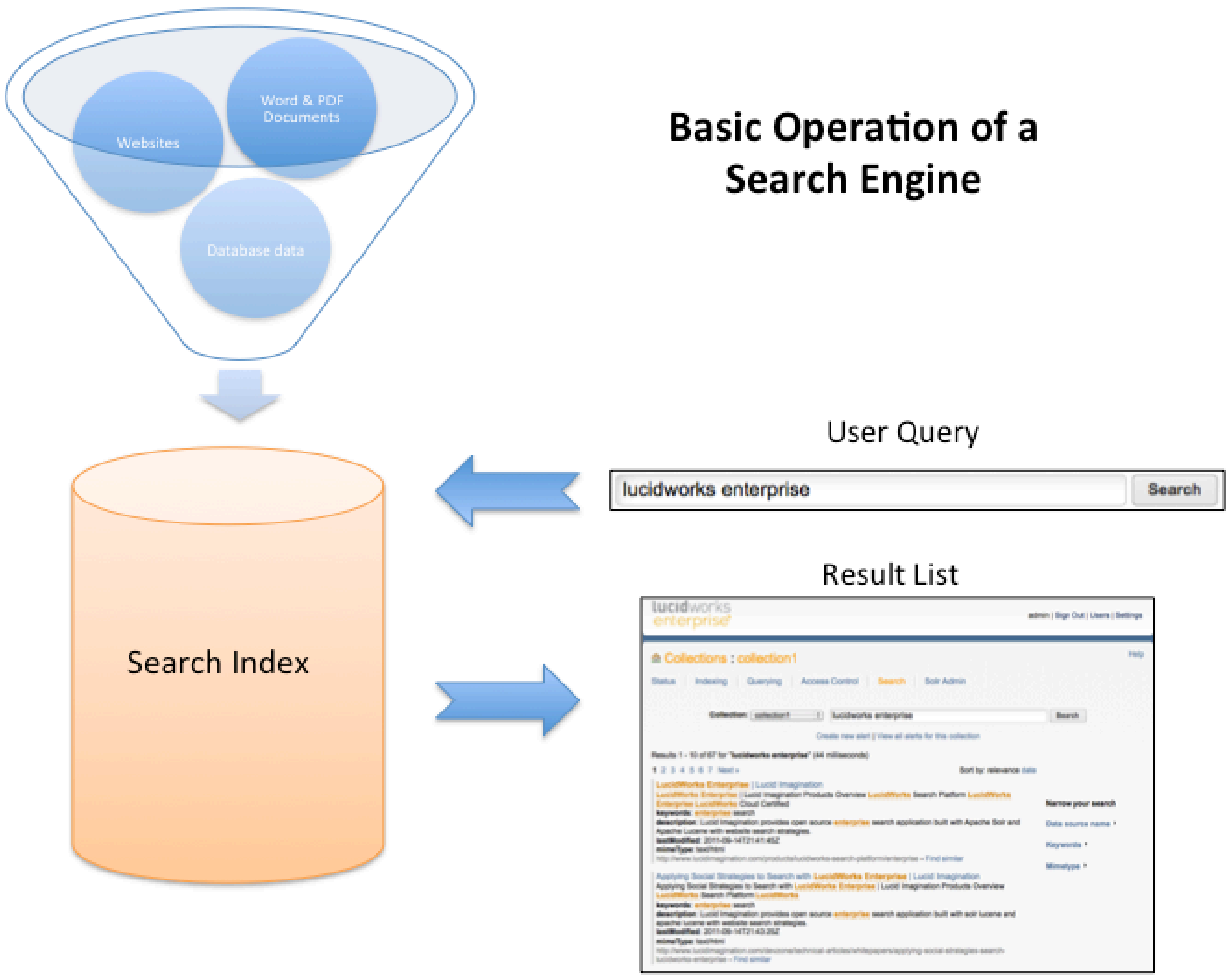
- Starting and Stopping LucidWorks Search
- Configuring Default Settings
- LucidWorks System Usage Monitor

How Search Engines Work

In its simplest form, a search engine is an application that enables a user to query a data set and get a list of documents in response. Most people are familiar with search engines that search the internet, but search engines are also built for more specific purposes. Enterprise documents or websites are not available to the public at large, so they can't be searched with internet search engines such as Google or Yahoo. An organization may have an online store and wish to customize their site to allow customers to find products.

In LucidWorks Search, each unit of text to be searched is a "document", whether it is an article, a website, a product description, or a phone book entry. In an enterprise environment, the administrator determines which of these documents make up the data set to be searched.

This graphic shows the basic operation of a search engine:



Indexing

For a user to search a set of documents, the search engine needs to know what is in them. The process a search engine uses to find out what is in a document is called "indexing". Essentially, an administrator tells the search engine where to find the document or documents, or feeds them to the search engine by way of an uploading process. The search engine then creates an index of all the words it finds, along with a pointer to the document in which it found them. Most information within documents is organized into "fields." Fields contain information that serves a specific, important purpose in the document, such as Title, Author, or Creation Date. Good search engines are able to identify these fields and create an index for each one.

Once the search engine creates an index, lots of interesting features can be added to aid users in their search experience, such as a spelling checker, automatic query completion, faceting of results, and "find similar" functionality.

Searching

Once the search engine has created an index of available content, it is ready to accept a search. This happens when the user enters a keyword or phrase, and the search engine compares that keyword or phrase against the index, returning pointers to any documents that are associated with them.

Of course, people are surprisingly different in the way in which they approach a topic, so search engines need to take these variations into account. The goal of a search engine is to match words entered by a user to words found in a document, so one technique it uses is to "normalize" both the user's query and terms that have been indexed as much as possible to find the best possible match, similar to the way in which you might convert both a target string and the text you are matching to uppercase in order to eliminate case-sensitivity.

Full-text Searching and Challenges

Several inherent challenges complicate full-text search. First, there is currently no way to guarantee the searcher will find the "best" results because there is often no agreement on what the "best" result is for a particular search. That's because evaluating results can be very subjective. Also, users generally enter only a few terms into a search engine, and there is no way for the search system to understand the user's intention for a search. In fact, if the user is doing an initial exploration of a topic area, the user may not even be aware of his or her intention.

A system that understands natural language (that is, the way people speak or write) perfectly is usually considered the ultimate goal in search engine technology, in that it would do as good a job as a person in finding answers. But even that is not perfect, as variations in human communication and comprehension mean that even a person is not guaranteed to find the "right" answer, especially in situations where there may not even be a single "right" answer for a particular question.

Some search engines, such as the LucidWorks Search , are built with features that try to solve, or at least mitigate, these challenges.

How LucidWorks Search Works

Like any other search engine, LucidWorks Search works by indexing several kinds of documents and providing a way for a user to search them. It uses Lucene and Solr to handle the core indexing and query processing tasks, and leverages the latest advancements in those projects. LucidWorks also builds on the work of the open-source community by adding crawling features, a robust REST API, an easy-to-use administration interface, and other features.

This graphic shows the relationship between Lucene, Solr, and LucidWorks Search.

The Apache Solr/Lucene core provides the indexing and searching functionality on which LucidWorks is built. As an application developer using LucidWorks Search, you can access this functionality in the same way that you access a traditional Solr installation. This includes field definition, document analysis, faceting, and basic query interpretation. Customers with LucidWorks Search installed on their own servers can work with the Apache Solr/Lucene core directly if they choose. Customers who use LucidWorks Search on AWS or Azure access much of the same functionality through the Admin UI.

On top of the Apache Solr/Lucene core is LucidWorks Search, which provides programmatic and GUI access to features that are normally difficult to work with directly, such as field definition or data source creation and scheduling.

- The LucidWorks Search Admin User Interface provides configuration and management tools for almost every feature of LucidWorks, including document acquisition, security, and field definitions.
- The REST API provides programmatic access to almost all configuration and management functions within LucidWorks.

Most of the functionality provided by LucidWorks comes from the LWE-Core and LWE-Connectors components, which manage all of these processes and features so administrators can concentrate on building and managing their own applications rather than the underlying search engine.

Working With LucidWorks Search Components

LucidWorks Search has three main components that can each be run together on a single server or deployed on separate servers if desired. While LucidWorks Search customers on AWS or Azure will not often need to interact with these components, an understanding of how they work is helpful for a deeper understanding of the system as a whole.

About the Components

Each component is a single JVM process. The system properties for each JVM can be modified with the `master.conf` file found in the `$LWE_HOME/conf` directory.

LWE-Core

The LucidWorks Search Core component is the main engine of the application. It contains the search index, the index definitions, the query parser, the embedded Solr application and Lucene libraries, as well as serves the REST API (with the exception of Alerts).

LWE-UI

The UI component includes all web-based graphical interfaces for administering the application, a sample search interface, and the enterprise alerts feature. Through the Admin UI, you can modify index fields, configure data sources for content collection, define aspects of the search experience, and monitor system performance. The Search UI provides a front-end for users to submit queries to LucidWorks Search and review results. It is not intended as a production-grade user interface, rather as a sample interface to use while configuring and testing the system. Enterprise Alerts provide a way for users of the front-end Search UI to save searches and receive notifications when new results match their query terms. There is a user interface piece with forms and screens for users to configure and review their alerts, as well as a REST API for programmatic access to the Alerts features.

LWE-Connectors

The Connectors component performs all the crawler functions, which include crawling data sources on demand or at a specific schedule, maintaining a crawl history (as applicable; each crawler varies in their behavior), and saving data source configuration information for use by the crawlers.

Default Installation URLs

This guide will refer to example URLs that will reference the default installation URLs for each component. These defaults are:

Component	Default URL
LWE-Core	http://127.0.0.1:8888/
LWE-UI	http://127.0.0.1:8989/
LWE-Connectors	http://127.0.0.1:8765/

These URLs are used by the installer for two purposes:

1. When the various components communicate with each other, or link to one another, they specify which URL will be used.
2. If the "Enable" check box is selected for a component when using the installer, then that component will be run locally, using the port specified in the URL.

The default LucidWorks start scripts start all components at the same time. However, it is possible to restart or stop a single component. See the section Starting and Stopping LucidWorks Search for details.

Configuring the Components

This functionality is **not available** with LucidWorks Search on AWS or Azure

If all components are run on the same machine, they must be defined with different ports. They can also be configured to run on different servers.

There are three possible ways to configure the components:

1. All components run on the same machine and they are started and stopped together. This is the default for the installer, which automatically prompts for default ports that are different for each component. To use this mode, you only need to run the installer once and follow through the process completely.
2. All components run on the same machine but they are started and stopped separately. This would require running the installer three times on the same machine. See Running Components on Different Servers or Different Ports of the Same Server below for detailed instructions on how to do this.
3. Each component is on a different machine and started and stopped separately. This requires running the installer on each machine. See Running Components on Different Servers or Different Ports of the Same Server below for detailed instructions on how to do this.

Running Components on Different Servers or Different Ports of the Same Server

To run components on different ports of the same server, you must run the installer three times, putting each installation in a different installation path (that is, a distinct "LWE Home"). If running the components on different servers entirely, the default installation path could be used on both servers because they are different machines.

For example, consider a use case where we want to:

- Run the LWE-Core component on the machine called 'server1' on port 8888.
- Run the LWE-UI component on the machine called 'server2' on port 8989.
- Run the LWE-Connectors component on the machine called 'server3' on port 8765.

In this scenario, we need two installations, and our installation steps would be like the following:

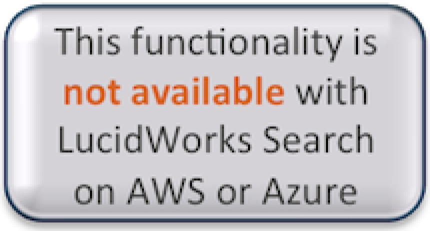
First Installation, LWE-Core on 'server1':

1. Launch the installer and follow the steps until the component selection screen.
2. Change the default URL for LWE-Core to http://server1:8888.
3. Deselect the "Enable" checkbox for the LWE-UI and LWE-Connectors components so they will not be installed on server1.
4. Change the URL for the LWE-UI to http://server2:8989. Even though the other components will not be run as part of this installation, this URL will be used in the LWE-Core configuration to be able to talk to LWE-UI.
5. Change the URL for the LWE-Connectors to http://server3:8765.
6. Advance to the Next screen, and select a path for this installation.
7. Finish the installation.
   Second Installation, LWE-UI on 'server2':
8. Launch the installer again, and advance to the component selection screen.
9. Deselect the "Enable" checkboxes for the LWE-Core and LWE-Connector components so they will not be installed on server2.
10. Change the URL but change the URL to http://server1:8888.
11. Change the URL for LWE-UI to http://server2:8989.
12. Change the URL for LWE-Connectors to http://server3:8765.
13. Advance to the Next screen, and select a path for this installation on server2. If you are installing LWE-UI on the same machine as the other components, you would enter a different installation path.
14. Finish the installation.
    Third Installation, LWE-Connectors on 'server3':
15. Launch the installer again, and advance to the component selection screen.
16. Deselect the "Enable" checkboxes for the LWE-Core and LWE-UI components so they will not be installed on server3.
17. Change the URL for LWE-Core to http://server1:8888.
18. Change the URL for LWE-UI to http://server2:8989.
19. Change the URL for LWE-Connectors to http://server3:8765.
20. Advance to the Next screen, and select a path for this installation on server3. If you are installing LWE-UI on the same machine as the other components, you would enter a different installation path.
21. Finish the installation.

Please note:

- Starting and stopping server processes will need to occur in all three installations; there is no single start/stop script that will work across multiple installations.
- The `$LWE_HOME/conf/master.conf` files in all installations will continue to refer to components that were not installed. You must take care not to enable a component in the wrong installation.
- The Click Scoring Relevance Framework runs as part of the LWE Core, and requires that `click-<collectionName>.log` (generated by the Search UI) be available in the `$LWE_HOME/data/logs/` directory. If LWE-Core and LWE-UI components are enabled in different installations, an external process must be responsible for copying that file into the `$LWE_HOME/data/logs` directory of the LWE-Core installation.

System Directories and Logs


This functionality is not available with LucidWorks Search on AWS or Azure

There are several important directories in the LucidWorks Search installation. System activities are recorded in several log files. Knowing where files and logs are located will make system configuration and troubleshooting easier.

Locating Files and Directories

The following table shows the default location of some directories that may be needed to effectively work with LucidWorks Search. These paths are all relative to the "LucidWorks Home" directory (`$LWE_HOME`) which is specified during installation.

What	Path
Configuration Files	`$LWE_HOME/conf/`
Documentation	`$LWE_HOME/app/docs/` (PDF) or http://lucidworks.lucidimagination.com (Online)
Examples	`$LWE_HOME/app/examples/`
Jetty Libraries	`$LWE_HOME/app/jetty/lib/`
Licenses	`$LWE_HOME/app/legal/`
Logs	`$LWE_HOME/data/logs/` (See below for log file list)
LucidWorks Indexes	`$LWE_HOME/data/solr/cores/collection/data/`
LucidWorks Logs	`$LWE_HOME/data/solr/cores/LucidWorksLogs/data/`
Solr Home	`$LWE_HOME/conf/solr/`
Solr Configuration Files	`$LWE_HOME/conf/solr/cores/collection/conf/`
Solr Source Code	`$LWE_HOME/app/solr-src/`
Start/Stop Scripts	`$LWE_HOME/app/bin/`

Editing Configuration Files on Windows

LucidWorks Search holds configuration files open after reading them, which may cause problems on Windows systems that do not allow editing open files. In this case, stop LucidWorks Search before editing files on Windows to be sure the edits are saved properly.

Configuring LucidWorks Search Directories

After you have installed LucidWorks Search, you can configure the location of of the `app`, `conf`, `data` , and `logs` directories by passing these parameters to the start script (`start.sh` or `start.bat`):

- `-lwe_app_dir`
- `-lwe_conf_dir`
- `-lwe_data_dir`
- `-lwe_log_dir`

For example, to change the location of the `data` directory, pass the following parameter to your start script:

```
start.sh -lwe_data_dir /var/data
```

See the section on Starting and Stopping LucidWorks Search for more information about the start scripts.

Temporary Files

By default, LucidWorks Search uses standard system directories (as detected by the JVM) for creating temporary files. This can be changed by adding a system property to the `master.conf` for `java.io.tmpdir` in the section that controls each JVM for the system. For example, to change the location of temporary files for the LucidWorks Core component, you would follow these steps:

1. Shut down LucidWorks using the instructions found in the section on Starting and Stopping LucidWorks Search.
2. Open `master.conf` with a text editor (found in `$LWE_HOME/conf`.
3. Find the section for `lwecore.jvm.params` and add `-Djava.io.tmpdir=/tmp/files/`.
4. Start LucidWorks.

The directory chosen as the location for temporary files should exist before starting LucidWorks Search, and must be writable by the user running LucidWorks.

System Logs

LucidWorks Search records system activities to rolling log files located in the `$LWE_HOME/data/logs` directory of the installation by default. The table below describes the main purpose of the various log files.

Log Name	Name Pattern	Function
Connector component log	`connectors.<YYYY_MM_DD>.log`	Connectors component operations, including the output of all crawling operations.
Connector request log	`connectors.request.<YYYY_MM_DD>.log`	Requests to the connectors component. These usually come from the Core component.
Core component log	`core.<YYYY_MM_DD>.log`	LucidWorks Core component operations, such as indexing.
Core request log	`core.request.<YYYY_MM_DD>.log`	Requests to the core component. These could come from either the Connectors or the UI component.
Core standard error log	`core-stderr.log`	Errors from Jetty startup (if any).
Core standard output log	`core-stdout.log`	Messages from Jetty startup (if any).
UI component log	`ui.<YYYY_MM_DD>.log`	Information from the Rails application, which runs the Search, Admin and Alerts components.
UI request log	`ui.request.<YYYY_MM_DD>.log`	Requests to the UI component.
Ruby standard error log	`ruby-stderr.log`	Errors from Ruby startup (if any).
Ruby standard output log	`ruby-stdout.log`	Messages from Ruby startup (if any).
Click log	`click-<collectionName>.log`	User click data, for use in relevance boosting (if enabled).
SharePoint crawl log	`google_connectors.feed.log`	SharePoint crawling operations. Note, this file can also include a number in the name, such as `google_connectors.feed0.log`, etc.

The LucidWorks Search Core log is configured by the `$LWE_HOME/conf/log4j-core.xml` properties file. The default is to create a distinct log per date (server time). The LucidWorks Search UI log is configured by the `$LWE_HOME/conf/log4j-ui.xml` properties file. The default is to create a distinct log per date (server time).

Information on how to modify log4j settings for the Core and UI log files is available at http://logging.apache.org/log4j/1.2/manual.html.

Log files are available through the Admin UI, by going to the Server Logs page for a collection and clicking the link at the bottom of the page.

If for some reason the Admin UI is not available, log files could be downloaded with a curl command such as:

```
curl http://localhost:8888/logs/<log_file_name>
```

LucidWorksLogs Collection

LucidWorks Search records log files for your Solr indexes in a collection called LucidWorksLogs, which contains a pre-configured data source also called `lucidworkslogs`. You can view the data for the LucidWorksLogs collection as you would for any other collection. You can also access the log files directly in the `$LWE_HOME/data/solr/cores/LucidWorksLogs/` directory.

The LucidWorksLogs collection powers the error log and all statistics about recent query and indexing activity that is shown in the Admin UI.

The log files on a LWE-Core server are accessible via HTTP at the URL `"http://server:port/logs"`. This URL lists all files currently in the logs directory, and provides links for downloading them individually. This can be useful in situations where you do not have direct shell access to the LWE-Core machine, but would like to review the log files for troubleshooting purposes.

When securing the HTTP Port of LWE-Core installation, consideration should be taken as to whether the "/logs" directory should be secured or not.

Deleting the LucidWorksLogs Collection

It is possible to delete the LucidWorksLogs collection if desired; however, this will disable the server log page within other collections, all activity graphing, and all calculations of Most Popular and Most Recent queries.

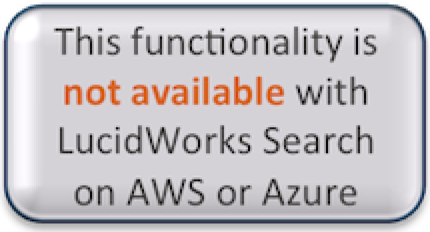
If the collection was deleted in error, or if you'd like to restore it at a later time, go to the Server log page within any collection and click **Recreate the log collection**.

It is also possible to remove the LucidWorksLogs data source from the LucidWorksLogs collection (i.e., retain the collection for possible later use, but remove the mechanism that indexes the logs). However, at the current time it will automatically be re-created and re-scheduled on server restart. If you wish to disable log crawling, you must either remove the entire LucidWorksLogs collection, or modify the LucidWorksLogs data source so that the schedule is not active (you can modify the schedule with the Data Source Schedules API or in the Schedules screen of the Admin UI.

Related Topics

- Working With LucidWorks Search Components
- Starting and Stopping LucidWorks Search

Starting and Stopping LucidWorks Search


This functionality is not available with LucidWorks Search on AWS or Azure

LucidWorks Search can be started and stopped using start and stop scripts provided with the application. These scripts are described below.

Windows users who have configured LucidWorks Search to run as a service should use the Services panel in Windows to manage the starting and stopping LucidWorks.

Starting LucidWorks Search

If you did not allow the installer to start LucidWorks Search, or if shortcuts were not installed, you can still start or stop LucidWorks manually from the command line. This will start all components:

1. Open a command shell, and make sure Java 1.6 or greater is in your path.

2. Change directories to the LucidWorks installation directory, then to the `$LWE_HOME/app/bin` directory.
3. Invoke `start.sh` for UNIX/Mac/Cygwin or `start.bat` for Windows systems.

Stopping LucidWorks Search

To stop LucidWorks Search, use the stop scripts at the command line. This will stop all components and any running processes.

1. Open a command shell, and make sure Java 1.6 or greater is in your path.
2. Change directories to the LucidWorks installation directory, then to the `$LWE_HOME/app/bin` directory.
3. Invoke `stop.sh` for UNIX/Mac/Cygwin or `stop.bat` for Windows systems.

Restarting LucidWorks Search

To restart LucidWorks Search, first stop the servers and start them again using the processes outlined above.

Starting or Stopping Components Separately

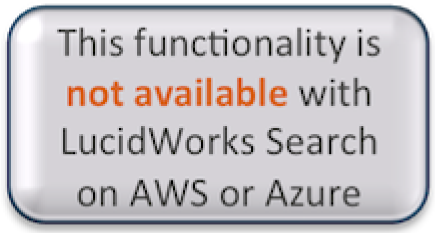
To start or stop any of the components without starting or stopping the other components, you can use the `start.sh`/`start.bat` or `stop.sh`/`stop.bat` scripts with the `-only` parameter, followed by the component name.

- Core component: `lwe-core`
- UI component: `lwe-ui`
- Connectors component: `connectors`

For example, this would start only the connectors using the `start.sh` script:

```
start.sh -only connectors
```

Configuring Default Settings



This functionality is not available with LucidWorks Search on AWS or Azure

You can configure many default settings in LucidWorks Search in the `defaults.yml` file located in the `$LWE_Home/conf/lwe-core` directory. You must restart LucidWorks after editing this file for your changes to take effect.

Some of the default settings you can configure include:

- Default crawl depth
- Default field mappings for crawlers
- Batch crawling of data sources
- Enabling or restricting data sources by crawler
- Default HTTP proxy settings

For example, to set the default crawl depth to 3 (which means that the crawler will follow links/sub-directories up to three steps away from the initial target), set `datasource.crawl_depth: 3`.

Here is an example `defaults.yml` file with comments that explain the various default settings (your default.yml file may vary):

```
file: defaults.yml
initCalled: true
location: CONF
values:
# Set to true to block index updates
  control.blockUpdates: false
# A whitespace-separated list of symbolic crawler names to enable; all crawlers are
enabled if this list is empty
  crawlers.enabled.crawlers: ''
# Absolute path that will be used to resolve relative path of local file system crawls
  crawlers.filesystem.crawl.home: null
# Per-crawler list of enabled datasource types, whitespace-separated. All available
types are enabled if this list is empty.
  crawlers.lucid.aperture.enabled.datasources: ''
# Per-crawler whitespace-separated list of restricted datasource types; all enabled
types are unrestricted if this list is empty
  crawlers.lucid.aperture.restricted.datasources: ''
  crawlers.lucid.external.enabled.datasources: ''
  crawlers.lucid.external.restricted.datasources: ''
  crawlers.lucid.fs.enabled.datasources: ''
  crawlers.lucid.fs.restricted.datasources: ''
  crawlers.lucid.gcm.enabled.datasources: ''
  crawlers.lucid.gcm.restricted.datasources: ''
  crawlers.lucid.jdbc.enabled.datasources: ''
  crawlers.lucid.jdbc.restricted.datasources: ''
  crawlers.lucid.logs.enabled.datasources: ''
  crawlers.lucid.logs.restricted.datasources: ''
  crawlers.lucid.solrxml.enabled.datasources: ''
  crawlers.lucid.solrxml.restricted.datasources: ''
# Default data source bounds: choose none or tree
  datasource.bounds: none
# Batch processing; caching of crawled raw content
  datasource.caching: false
# Explicitly commit when crawl is finished
  datasource.commit_on_finish: true
# Solr's commitWithin setting, in milliseconds
  datasource.commit_within: 900000
```

```
# Default crawl depth: the number of cycles or hops from the root URL/directory. Set to
-1 for unlimited crawl depth
  datasource.crawl_depth: -1
  datasource.follow_links: true
# Set to true to ignore the rules defined in /robots.txt for a site
  datasource.ignore_robots: false
# Perform indexing at the same time as crawling
  datasource.indexing: true
# Default field mapping for Aperture-based crawlers. This is the baseline, the field
mapping for each data source can be customized.
  datasource.mapping.aperture: &id001
!!com.lucid.admin.collection.datasource.FieldMapping
    datasourceField: data_source
    defaultField: null
    dynamicField: attr
    literals: {}
    mappings:
      slide-count: pageCount
      content-type: mimeType
      body: body
      slides: pageCount
      subject: subject
      plaintextmessagecontent: body
      lastmodified: lastModified
      lastmodifiedby: author
      content-encoding: characterSet
      type: null
      date: null
      creator: creator
      author: author
      title: title
      mimetype: mimeType
      created: dateCreated
      plaintextcontent: body
      pagecount: pageCount
      contentcreated: dateCreated
      description: description
      contributor: author
      name: title
      filelastmodified: lastModified
      fullname: author
      fulltext: body
      messagesubject: title
      last-modified: lastModified
      acl: acl
      keyword: keywords
      contentlastmodified: lastModified
      last-printed: null
      links: null
      url: url
      batch_id: batch_id
      crawl_uri: crawl_uri
      filesize: fileSize
      page-count: pageCount
      content-length: fileSize
      filename: fileName
    multiVal:
      fileSize: false
      body: false
      author: true
      title: false
      acl: true
      description: false
      dateCreated: false
    types:
      filesize: LONG
      lastmodified: DATE
      datecreated: DATE
      date: DATE
    uniqueKey: id
# Default field mapping for crawlers that use Tika parsers
  datasource.mapping.tika: *id001
# Maximum size of content to be fetched
  datasource.max_bytes: 10485760
# Maximum number of documents to collect; set to -1 for unlimited documents
  datasource.max_docs: -1
# The maximum number of concurrent requests processed by a data source crawl, for those
crawlers that support multi-threaded crawling.
# As of v2.1, this is only the lucid.fs crawler, which supports the Hadoop, S3 and SMB
data source types.
  datasource.max_threads: 1
# Set to true to apply content parsers to the retrieved raw documents
  datasource.parsing: true
# Defines the host name of an HTTP proxy server to use for web crawling; leave blank if
you are not using a proxy server
  datasource.proxy_host: ''
# HTTP proxy password, if you are using an HTTP proxy server
  datasource.proxy_password: ''
# proxyPort for an HTTP proxy server, if you are using one
  datasource.proxy_port: -1
# Username to authenticate with HTTP proxy server
  datasource.proxy_username: ''
# If true, text extracted from a compound document (one which has other embedded
documents and resources, such as emails with attachments
# or Office documents with OLE attachments, but not .zip, .jar., or similar) will be
appended to the text of the container document.
# If false, each embedded resource is treated as a separate document with a URL in the
form of the container document URL plus ! and
# the embedded document's name or identifier. If documents are treated as separate
documents (when this setting is false),
# the URL of the container document is added to the field "belongsToContainer".
  datasource.tika.parsers.flatten.compound: true
# If false, documents with mime types that start with "image/" are ignored.  If true,
the documents are sent to Tika for parsing,
# which may result in useful metadata being extracted from them but may also result in
a large number of fields and terms.
  datasource.tika.parsers.include.images: false
# If true, and LucidWorks runs in the same JVM as Solr, then crawlers will first try
using direct calls to SolrCore for updates,
# which may result in performance improvements. If false (the default), the SolrJ API
is used for updates.
  datasource.use_direct_solr: false
# If true, datasources will attempt to verify access to the remote repositories.
  datasource.verify_access: true
# HTTP-specific preferences sent in HTTP headers during crawling.
  http.accept.charset: utf-8,ISO-8859-1;q=0.7,*;q=0.7
  http.agent.browser: Mozilla/5.0
  http.agent.email: crawler at example dot com
  http.agent.name: LucidWorks
# The agent.string will allow a completely custom http.agent identifier. If this is not
empty, it will be used verbatim instead of all other 'http.agent.*' settings.
  http.agent.string: ''
  http.agent.url: ''
  http.agent.version: ''
  http.crawl.delay: 2000
# Maximum number of redirections in a redirection chain.
  http.max.redirects: 10
# Number of threads for HTTP crawling.
  http.num.threads: 1
# Socket timeout in milliseconds.
  http.timeout: 10000
# Specify the HTTP version: HTTP/1.1 if true; HTTP/1.0 if false.
  http.use.http11: true
  ssl.auth_require_authorization: false
  ssl.auth_require_secure: false
```

LucidWorks System Usage Monitor

The LucidWorks System Usage Monitor is a voluntary program to allow LucidWorks Search users to anonymously send basic information about their system to LucidWorks. We use this information to analyze the types of systems in use by our customers and how they are used so we can improve our product. At no point does the system collect information that could identify you, your organization, the documents indexed, or the type of content indexed.

Information Collected

The System Usage Monitor collects the following information:

- Operating System version and type

- Java version and type
- LucidWorks Search version and type
- Number of LucidWorks collections created
- Number of LucidWorks data sources created
- Number of LucidWorks documents indexed
- JVM memory free, available, and used
- Number of LucidWorks queries
- Number of documents added since last submission

How the System Usage Monitor Works

When Information is Sent

The System Usage Monitor sends information at each application startup (using the `start.sh` or `start.bat` scripts) and once per week on Saturdays.

How Information is Sent

When LucidWorks Search is started, the System Usage Monitor will transmit data about your system to a server hosted by LucidWorks with two HTTP requests. The first request contains system-level information and if that is successful, the second request will send LucidWorks-specific information, as listed above.

The information is sent via an encrypted POST request to https://heartbeat.demo.lucidworks.io. Each request includes a unique identifier, which is anonymous and can't be used to identify the sender. The IP that sent the request is not stored with the request.

The requests are logged in the `core.YYYY_MM_DD.log`. The requests will appear similar to this:

```
2012-10-23 19:05:56,618 INFO heartbeat.LucidStatsPublisher - Sending heartbeat stats:
uuid='3532f7e9-4280-4714-9e83-ea0a95fe90bd',
data='{product=lwe, current_product_version=0.0Enif, is_cloudy=false,
lwe_git_sha=7568ce8c35a394c4b987e3a17cb5e1b5ae5dac25,
java_version=1.6.0_35 (Apple Inc.), num_cpu_cores=4, os_version=Mac OS X (x86_64)}'
2012-10-23 19:05:58,831 INFO publish.MonitorRegistryMetricPoller - cache refreshed, 8
monitors matched filter,
previous age 1351019158 seconds
2012-10-23 19:05:58,865 INFO heartbeat.LucidStatsPublisher - Sending heartbeat stats:
uuid='3532f7e9-4280-4714-9e83-ea0a95fe90bd',
data='{num_docs=0, num_collections=1, num_datasources=0, jvm_memory_free=506720952,
jvm_memory_max=1065025536, jvm_memory_total=534708224,
num_adds=0, num_search_requests=0}'
```

Subsequent weekly updates are sent as a single request, including only the LucidWorks-specific information like number of documents, number of data sources, etc.

How to Opt-In or Opt-Out

During Installation

During installation of LucidWorks, you will be presented with an option to opt-in to the System Usage Monitor program. This option will appear after defining the installation path for the system. With the graphical installer, the box is checked by default and unchecking the box will opt-out of the program. If using the console installer, choose '0' as a response to opt-out of the program.

Post-Installation

Opting-in to the program will insert a line at the beginning of the `$LWE_HOME/conf/master.conf` file, as so:

```
# LucidWorks System Usage Monitor (comment the next line to disable this feature)
usageStatsServerId=3532f7e9-4280-4714-9e83-ea0a95fe90bd
```

To opt-out:

1. Stop LucidWorks Search
2. Open `master.conf` found in `$LWE_HOME/conf`
3. Comment out the line containing the `usageStatsServerID` by adding a hash mark (#) at the beginning of the line
4. Start LucidWorks Search

The same process can be followed to opt-in if the service was previously disabled, by removing the hash mark instead of inserting it.

More Information

For more information, including details of our commitment to protecting the privacy of your data, please see our website at http://www.lucidworks.com/lucidworks-system-usage-monitor.

Collections and Indexes

This section covers how to configure LucidWorks Search for your data.

Content in LucidWorks is indexed into a collection, which can have different documents, data sources, fields, field types and settings from other collections. Before starting to work with LucidWorks, review the section Working with Collections. Once one collection is configured as you like, it can be used as a template, as described in Using Collection Templates.

Once the collections are considered, then you can think about how to configure LucidWorks Search to index your content. These sections describe the options for setting up the indexes:

- Indexing Documents
- How Documents Map To Fields
- Customizing the Field Schema
- Reindexing Content
- Multilingual Indexing and Search

- Lucid Plural Stemming Rules
- Deleting the Index

Working with Collections

A single installation of LucidWorks Search may be used to index multiple types of content, serve multiple user constituencies, or accommodate multiple overlapping security rules. It does this by supporting the creation and use of multiple "collections". A collection is a set of documents that are grouped together with the same indexing and query rules. Each collection in LucidWorks has its own index and configuration files and is logically separate from all other collections.

For those familiar with Solr, the concept of collections in LucidWorks is very similar to the concept of cores in Solr.

Default Collections

By default, each LucidWorks Search installation includes two collections out of the box: "collection1" and "LucidWorksLogs".

Collection1 is the primary collection used by LucidWorks Search to store indexes and define query settings. It can be used as-is immediately after installation to start indexing documents and using the default Search UI. However, a collection cannot be renamed once created (nor can content be moved from one collection to another without indexing it all from scratch). So, if you think you'll use multiple collections and want to name each one based on what it contains or what it will be used for, you would probably create a new collection and start from there.

The LucidWorksLogs collection is a special collection, used to index logs for easier troubleshooting. It is discussed in more detail in the section on the LucidWorksLogs collection. It can be deleted at any time and recreated later, if desired.

If you want to delete collection1, you can do so after you've created at least one other standard collection, as there must always be at least one collection (not including the LucidWorksLogs collection).

A collection that has been customized can also be used as the basis for future collections; see the section on Collection Templates for more information.

Per-Collection Features

You can configure the following items for each collection individually:

- Data sources
- Fields
- Query settings
- Search UI
- Search Filters
- Schedules
- Solr Admin

After you have created additional collections, you should pay special attention to the collection name you are working with so you edit the proper configuration files or make the correct API calls. This is particularly true when using the REST API or several of the advanced configuration options discussed later in this Guide, but it also applies to the various screens of the Admin UI. Modifying the wrong collection out of context may have unexpected consequences including poorly indexed content or an inconsistent search experience for users.

System-Wide Features

The following items are system-wide and can only be configured for the entire LucidWorks Search installation or instance:

- Collection definition
- Access to user interfaces
- Users
- Alerts (although these take the collection as a parameter to limit the query)

Related Topics

- Creating a collection with the Collections API
- Creating a collection with the Admin UI
- System Directories and Logs

Using Collection Templates

This functionality is **not available** with LucidWorks Search on AWS or Azure

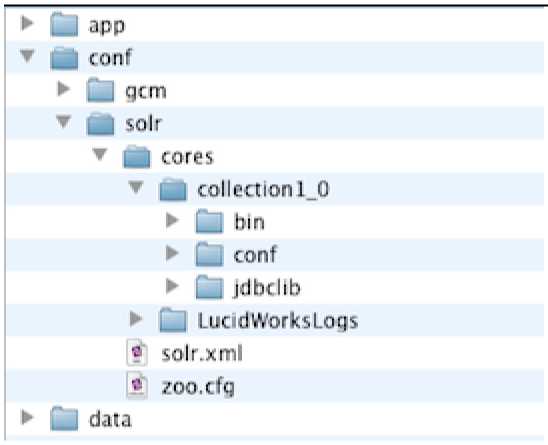
Collection templates allow you to copy the configuration files from a collection and use it as the basis for future templates. Creating a template is as simple as creating a `.zip` file from configuration files in the base collection and explicitly specifying that `.zip` file during new collection creation either via the Admin UI or the REST API.

Included Templates

Two templates are included with LucidWorks out of the box. Both templates can be found in `$LWE_HOME/app/collection_templates`.

- `default.zip`: This has the same default options and out-of-the-box fields as the standard "collection1" that exists by default after LucidWorks Search installation.
- `essential.zip`: This is a stripped-down version of the LucidWorks default configuration that includes only the few fields that are absolutely essential for the system to run (see Customizing the Field Schema for more details on the default field set).

Creating a Template



To make a custom template, create a new collection and configure it as needed, whether that is via the user interface, using the REST API, or manual editing of configuration files. All of the configuration files for a collection reside in the `instance_dir` for the collection, which is found under `$LWE_HOME/conf/solr/cores/collection`, where `collection` is the name of the collection that is being used as the basis for the template.

Then create a `.zip` file from the `instance_dir`. The .zip file can have any name, including `default.zip`, although using the same name would overwrite the system default template, meaning it would not be available at a later time if needed. All templates must be placed in `$LWE_HOME/conf/collection_templates` to be available during collection creation.

We recommend that you use all the sub-directories from the `instance_dir` even if some of the files have not been customized in the base collection.

Related Topics

- Working with Collections

Indexing Documents

The first step to being able to search is to create an index. The index stores all the terms from documents in such a way that results for user queries can be returned as quickly as possible.

Indexes are created by breaking a document into individual words and saving the word list. At the same time, documents are not solely lists of sentences and words, but instead usually contain some sort of structure - an email will likely have "to" and "from" information; Word and PDF documents may have "title" and "author" information, in addition to the main "body"; product descriptions may have "price", "description" or "color" information. These are known as fields within each document. Adding field information to the word list facilitates a user's ability to search for emails from a specific person, or shoes that come in a particular color.

Fields can contain different types of data. A title field, for example, is usually text (character data). A price contains a mix of digits and special characters (such as $ or €). Dates are generally Defining the type of data that a field will contain is a critical first step in defining the fields for the index.

Defining Fields

There are several things to consider when configuring fields. The primary one is whether to store the field or not. Stored fields take up space in the index, but they allow the field to then be indexed (that is, made searchable) or available to users for display. It may be preferable to store a field and use it for display in a results list, but not allow it to be searchable. Alternately, a field can be designated for use in a facet, so it would be stored and indexed, but perhaps not searchable. A careful analysis of documents should occur before indexing to be able to anticipate how it will be indexed. If fields are not correctly configured before a document is indexed, documents will need to be re-indexed at a later time. If that is required, the existing index can be deleted and documents can be added to it from scratch.

Indexing Data Sources

In order for users to be able to search, LucidWorks Search needs to have indexed documents. LucidWorks Search supports two main approaches for document discovery:

- Documents can be pushed directly into the system. Users who are familiar with Solr may already have processes and systems in place to push documents into the index. This is also an option if LucidWorks Search is not able to connect to the repository to pull documents from it.
- Documents can be pulled from remote repositories. LucidWorks Search has several pre-defined types of repositories that it is able to connect to; you configure these connections by creating "data sources" and selecting options appropriate for your needs.

Each of these approaches has several options and caveats to consider, which are covered in more detail **TODO FILL IN LINK HERE**.

Related Topics

- Customizing the Field Schema

How Documents Map To Fields

When LucidWorks Search crawls a data source, it extracts the target data and stores it in fields in the index. The specific mapping from the source data to the indexed fields is determined by the crawler you are using, which is in turn determined by the data source type. For a list of file types supported by LWE, see Supported Filetypes. Let us consider two common file types, both processed by the Aperture crawler: a website and a Microsoft Word document.

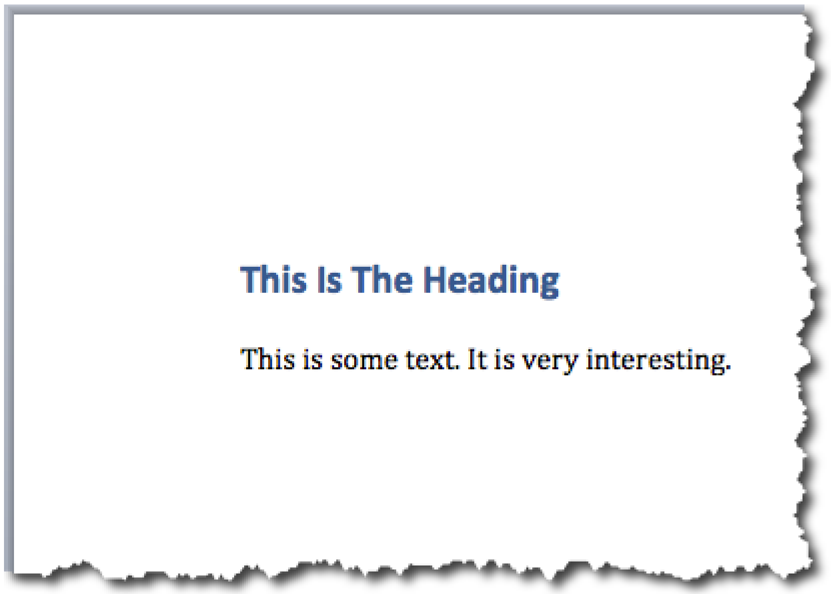
For the website, consider a case where you have crawled http://lucidworks.com with a crawl depth of zero, which means that only the first page is indexed. The Aperture crawler maps the web page as follows (note that this example is not complete or exhaustive):

Data Source	Field Mapping	Field Content
url	url	http://lucidworks.com

content-type	mimeType	html/text
title	title	Lucid Imagination is now LucidWorks. | LucidWorks
body	body	The Future Of Search

And so on.

For the Microsoft Word document, consider this document, included here in its entirety:



This Is The Heading

This is some text. It is very interesting.

Data Source	Field Mapping	Field Content
mimetype	mimeType	application/vnd.openxmlformats-officedocument.wordprocessingml
title	title	Example Word Doc
author	author	Drew Wheeler
body	body	This Is The Heading This is some text. It is very interesting.

For information on which crawlers handle which data source types, see the Data Source REST API documentation. If using the Admin UI, you don't need to worry about the crawler type. The UI also includes screens for modifying how documents are mapped to fields, or the Data Sources API can be used. For more information on fields in LucidWorks Search, see the Table of Fields in the section Customizing the Field Schema.

Customizing the Field Schema

When indexing documents, LucidWorks Search doesn't merely generate a list of all the words found on the page. It also tries to recognize the structure of the document, and remember some of that structure in the index. The structure of indexed documents is represented by the fields defined for the LucidWorks Search index. When terms are saved in the index, they are saved with information about the field in which they were found in the document.

Field definitions are stored in a `schema.xml` file for each collection. Users familiar with Solr will recognize this file, since it is the same `schema.xml` file that is used with a Solr installation. Instead of editing this file by hand, however, LucidWorks Search allows modifying the field and field type definitions with the Admin UI or with the REST API.

By default, LucidWorks Search contains field definitions to support various features of LucidWorks (such as crawling documents and Click Scoring) and to make it easier for users to get up and running. Not all users will need all fields, however, so you may want to add fields unique to your search application or just to trim the default set of fields so the list is easier to work with. This section describes the default fields, how they are used by LucidWorks Search, and if they can be removed for local installations.

One of the primary added values of LucidWorks Search is the integration of content crawlers for web sites, filesystems and other repositories of content. Many of the default fields are for this purpose and should be retained. In many cases, if they are removed from the schema, they will be recreated the next time a crawler needs them. However, if not using the LucidWorks crawlers, they can generally be safely removed. They will be added based on a dynamic rule ("*" rule) in the `schema.xml` file that should be retained to avoid unexpected failures of the crawlers. If this rule is left in place, nearly any field in the schema can be removed as it will be added back if it is needed.

> Only delete the "*" rule if you are absolutely positive other deleted fields will not be needed in your specific implementation. Deleting this rule may also complicate future upgrades, as it is not possible to predict when LucidWorks Search will add new fields to the `schema.xml` file to support future functionality.

Guidelines for Removing Fields from the Schema

Essential Fields

There are two fields that must be retained in `schema.xml`. The Admin UI and the Fields API will not allow deleting them:

- id
- timestamp

There are three additional fields that are considered essential to LucidWorks Search.

- data_source
- data_source_name
- data_source_type
- text_all

The three data source-related fields are considered essential for the Admin UI and APIs to know the source of the content that has been indexed. If not using the Admin UI nor the LucidWorks REST APIs, they could be deleted.

The text_all field is required because `schema.xml` declares it as the default search field for the Lucene RequestHandler (query parser), which is also the default for the basic Solr query parser. If you are using `lucid` or `DisMax`, however, and will never use the Lucene or Solr query parsers, the field could be deleted. However, it may be best to retain it.

We have created a sample schema that includes only the essential fields listed above that can be used for collection creation. See Using Collection Templates for more information.

Built-In Search UI Fields

LucidWorks includes a default search UI that can be used as-is or replaced with a fully local interface. If using it as-is, even for testing or during initial implementation, the following fields must also be retained in `schema.xml`:

- author
- author_display
- body
- dateCreated
- description
- keywords
- keywords_display
- lastModified
- mimeType
- pageCount
- title
- url

The Search UI includes these fields for results display and default faceting, so for it to work properly, these fields should be retained.

Fields to Support Specific Features

Several fields are included in `schema.xml` in support of specific LucidWorks features. They can be removed if those features are disabled or not in use. In some cases, however, they will be added back to the schema if the feature is enabled in the future.

Feature	Fields
Click Scoring Relevance Framework	click click_terms click_val
ACL	acl
Spell Check	spell
Auto Completion	autocomplete
Enterprise Alerts	timestamp
SolrCloud and Near Realtime Search	_version_
De-duplication	signatureField

Crawler Fields

The crawlers included with LucidWorks create fields in `schema.xml` that begin with *attr_* and are used to store document-specific metadata during crawl processes. They are not generally used otherwise by LucidWorks (such as in search results or other computations). Due to the dynamic "*" rule, they will be added back to `schema.xml` if not in place. If not using the LucidWorks crawlers, they can be removed, but it is recommended to retain them if possible.

Other Dynamic Fields

Several other dynamic fields (all including an '*', such as *_i, *_s, *_l, etc.) are defined in `schema.xml`. These can be removed if they will not be used - the only two we recommend that you retain are the "*" rule and the attr_* fields.

Table of Fields

The table below notes whether a field will be indexed, stored, used for facets or included in results. This is default behavior, and can be modified locally. After customization, this table may not reflect the state of your `schema.xml` file.

Field Name	Type	Indexed	Stored	Used for Facets	Included in Results	Used for

version	long	X	X			Document version control, used with Near Realtime Search and SolrCloud.
acl	string	X	X			Storing Access Control List information.
attr_* (any field starting with 'attr_')	string	X	X			Created by the crawlers and used for a wide array of document-specific metadata. Not specifically declared in the schema.xml file, but dynamically created during crawls.
author	text_en	X	X		X	Raw author pulled from documents. Used by default in the built-in Search UI.
author_display	string	X		X		Used for display of authors in facets. Used by default in the built-in Search UI.

autocomplete	textSpell	X	X			Stores terms for the auto-complete index. By default, it is created by copying terms from the title, body, description and author fields.
batch_id	string	X	X			Identifies the batch that added the document.
bcc	text_en	X	X			Used in processing email messages.
belongsToContainer	text_en	X	X			Used to store the URL of the archive file (.zip, .mbox, etc.) which contains the file.
body	text_en	X	X			The body of a document (generally, the main text). Used by default for display in the built-in Search UI.

byteSize	int		X			The size of the document.
cc	text_en	X	X			Used in processing email messages.
characterSet	string		X			The character set used for the document. Only populated if it is declared in the document (most commonly with HTML files).
click	string	X	X			Used with the Click Scoring Relevance Framework and contains the boost value.

click_terms	text_ws	X	X			Used with the Click Scoring Relevance Framework and contains the top terms associated with the document.
click_val	string	X	X			Used with the Click Scoring Relevance Framework and contains a string representation for the boost value for the document. The format allows it to be used for processing function queries.
contentCreated	date	X	X			The creation date for the document, if available.
crawl_uri	string	X				A copy of the URL for the document.

creator	text_en	X	X			The creator of the document, if available.
data_source	string	X	X			The ID of the data source that crawled this document.
data_source_name	string	X	X	X		The name of the data source that crawled this document.
data_source_type	string	X	X		X	The type of data source that crawled this document.
dateCreated	date	X	X		X	The date the content was created, if available.
description	text_en	X	X		X	The description from a document, if it exists in the document. For example, Microsoft Office document properties contains a description field that can be filled in by the user.

email	text_en	X	X			Not currently used by any LucidWorks crawlers.
fileName	text_en	X	X			The name of the file.
fileSize	int	X	X			The size of the file.
from	text_en	X	X			Used in processing email messages.
fullname	text_en	X	X			Data in this field is mapped to "author".
generator	text_en	X	X			The name of the software that generated the document, if available.
id	string	X	X		X	Unique ID for the document.
id_highlight	text_en	X	X			No longer used by LucidWorks and will be removed in a later version.
incubationdate_dt	date	X	X			Used in older Solr example documents.
keywords	text_en	X	X		X	The keyword list from a Microsoft Office document.

keywords_display	comma-separated	X		X		Terms from the keyword field formatted for display to users.
lastModified	date	X	X		X	Date the content was last modified.
mimeType	string	X	X	X	X	The type of document (PDF, Microsoft Office, etc.).
name	text_en	X	X			Data in this field is mapped to "title".
otherDates	date	X	X			Dates other than dateCreated or lastModified would be mapped to this field.
pageCount	int	X	X		X	The number of pages in a Microsoft Office document such as Word or PowerPoint.
partOf	string	X	X			Typically used for an email attachment, this points to the larger document of which this document is a part.
price	float	X	X			Example field that could be used for processing e-commerce data.

retrievalDate	date	X	X			Not currently used, but could be used for the date a web document was retrieved from its server.
rootElementOf	text_en	X	X			Populated only for the root or initial document of a crawl.
signatureField	string	X	X			Used with the de-duplication feature.
spell	textSpell	X				Stores the terms to be used in creating the spell check index. Created by copying terms from the title, body, description and author fields.
text_all	text_en	X				Used to combine text fields for faster searching. Created by copying terms from the id, url, title, description, keywords, author and body fields.
text_medium	text_en	X	X			Not currently used.
text_small	text_en	X	X			Not currently used.

timestamp	date	X	X	X	X	Time the document was crawled and used for date faceting and display of activities in the LucidWorks Admin UI. Also used for Enterprise Alerts to know when the document was added to the index for alerts processing.
title	text_en	X	X			The title of the document.
to	text_en	X	X			Used in processing email messages.
type	text_en	X	X			Used by the lucid.aperture crawler to store Aperture's classification of an information object, separate from its MIME type.
url	string		X		X	The URL to access the document.
username	text_en	X	X			No longer used and may be removed in a later version.

weight	float	X	X			Example field that could be used for processing e-commerce data.

Reindexing Content

It is considered a best practice to fully design your index (i.e., define all the fields you'll need and their attributes) before indexing large amounts of content. However, the reality is that things change - you have new requirements, new content, or you'd like to give users new options for searching.

As tolerant as LucidWorks Search is to changes, there are certain changes that cannot be made without fully reindexing, by which we mean deleting content from the indexes and re-processing it from scratch. Adding a field or changing field mapping options for an existing data source, as examples, require indexing the content again to get the new field information from the document or change the way the incoming content was processed into the index.

In addition, changes to the following attributes of a field require some degree of re-index:

- Field Type value
- If it is Indexed
- If it is Stored
- If it is Multi-valued
- Short Field Boost value

Below are the options for re-indexing content.

Re-crawl the Content

All of the crawlers store information about what documents it has previously processed, and uses that information for future crawls, usually only adding documents that are new (have never been indexed before), removed from the content repository (and should be removed from the index), or changed (and should be replaced in the index with the new copy). This means that documents already in the index are not re-processed and may be skipped, which may create a mis-match between existing content and new content being indexed.

Empty the Data Source

The Admin UI includes a button to Empty a data source. This button only deletes the documents from the data source, but does not reset any of the crawl history information, which keeps track of content that were previously found and uses that information to understand if content is new, has been deleted (and should be removed from the index), or has been updated (and should be removed and replaced with the new content). The associated API is the Collection Index Delete API, which has an option to specify deleting documents from the index associated with a data source.

If changes to a collection's field list or field type list have been made, emptying the documents from the data source may not be sufficient to fully re-crawl the content to update the fields because the next time a crawl is run it will be executed incrementally, using the crawl history information that it has stored. This means that if a document has not changed it will not be re-added to the index because the crawl history registers it as unchanged.

There is, however, a REST API to delete the crawl history called Data Source Crawl Data Delete which can be used if necessary.

Delete the Data Source

Deleting the data source deletes the metadata for the data source (the configuration details for LucidWorks Search to access the content repository), and any of the content from the index and the crawl history. It can be done with either the Admin UI Delete button or the Data Sources API. This might be the easiest way to clear the content so it can be re-crawled and re-indexed with the new field attributes.

Empty the Collection

Emptying the collection stops any running data sources, deletes the entire search index for the collection, and removes all crawl history for each data source. It is a good option if you have a number of data sources that you configured during initial implementation and would like to start fresh with production data. Emptying the collection can be done with either the Empty this Collection button in the Admin or the Collection Index Delete API.

Delete the Collection

Deleting the entire collection will delete all the data sources, stop any running jobs, delete all associated content, and remove all collection-related settings for the index. It can be done with the Delete this Collection button in the Admin UI or the Collections API.

Multilingual Indexing and Search

LucidWorks Search has a number of capabilities designed to make working with multilingual data straightforward. By default, it includes support for most European languages, Japanese, Korean and Chinese. Multilingual capabilities are provided by Lucene's analysis process (see the Language Analysis section of the Solr Reference Guide for more details). Since Lucene is built on Java, which is Unicode enabled, many multilingual issues are handled automatically by LucidWorks and Solr. In fact, the main issues with multilingual search are mostly the same issues for working with any language: how to analyze content, configure fields, define search defaults, and so on.

Approaches to Multilingual Search

Besides the normal language issues, multilingual search does require decisions about whether to use a single field for each language, a field for each language or even a separate indexes for each language. Each of these three approaches has pros and cons.

Single Field Approach

Pros

- Simple to search across all languages
- Fast to search

Cons

- Requires Language Detection software, which is not included in LucidWorks, and which will slow down indexing
- Requires the query language to be specified beforehand, since language detection on queries is often inaccurate
- May return irrelevant results, since words may have same spelling but different meanings in different languages
- May skew relevancy statistics
- Hard to filter/search by language

Multiple Field Approach

Pros

- No language detection required
- Easy to search and/or filter by language
- Relevancy is clear since there is no noise from other languages with common spellings (minor)

Cons

- Many languages = many fields = more difficult to know what to search
- Slower to search across all languages

Multiple Indexes Approach

Pros

- Easy to bring one language off-line for maintenance without effecting other languages
- Can easily partition data and searches across machines by language
- Easy to search and filter by language

Cons

- More complex administration
- Slower and more difficult to search across all languages

Currently, LucidWorks supports the multiple field and multiple index approach out of the box, but the single field approach is still possible with some additional work that requires intermediate level Solr expertise.

Open Source Multilingual Capabilities

The crux of multilingual handling is applying analysis techniques to the content to be indexed. These techniques are specified in the Solr's `schema.xml` by the `<fieldType>` declarations. Out of the box, LucidWorks comes configured with numerous predefined field types designed to make indexing and searching multilingual content easy to do.

Note that most of the supported languages (especially the European languages) are designed to use Dr. Martin Porter's Snowball stemmers along with stop word filters, synonym filters and various other filters.

Multiple Languages May Require Customization

Although LucidWorks ships with default analysis and filter techniques, they may need customization for your search application. Consider the included language configurations to be good starting points for support of any given language and make adjustments as needed. For information on relevance tuning and debugging for additional tools and techniques to improve results, see Understanding and Improving Relevance.

By setting up the appropriate fields per language, it is possible to simply point LucidWorks at the given data source and have it index the content.

Adding Support for Other Languages

While there are a wide variety of languages available "out of the box", there may come a time where support for a new language is needed. There are a few possibilities:

- Try out the language with the StandardAnalyzer, since it often does the right thing as far as tokenization and basic analysis goes. Note that the analyzer doesn't do stemming or perform more advanced language translation.
- Write an Analyzer, Tokenizer or TokenFilter and the associated Solr classes as described on the Solr Wiki page at http://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters.
- Use an n-gram character-based approach that chunks characters into n-grams and indexes them. Accuracy will be limited, but it may be better than nothing.

If choosing the second option, the new capability can be brought into LucidWorks as described in the Solr wiki section on SolrPlugins.

Related Topics

- Language Analysis from the Solr Reference Guide
- AnalyzersTokenizersTokenFilters from the Apache Solr Wiki

Lucid Plural Stemming Rules

The purpose of stemming is to translate different forms of similar words to a common form so that a query for one form of a word will also match the other forms. The most common difference between word forms is singular words versus their plurals. Another variation in form is the variety of conjugations of a word. Although the administrator can select what stemming filter or options are enabled for each field type, by default all text fields will have a stemming filter that converts most plural words to singular.

Stemming is not a perfect process, so some plurals may be missed and some singular words may be mistakenly translated to some other singular or possibly even a non-word. Non-words, such as jargon, names, and acronyms can also be mistakenly stemmed. But, since stemming usually occurs at both document indexing time and at query time, improper stemming is frequently not even detectable. The default rules try to avoid removing "s" endings that are not plural (or verb conjugations), such as "alias" or "business."

If stemming proves problematic for a given application, the administrator can always turn it off or select an alternative stemming filter.

The Lucid plural stemmer is designed to focus on stemming of plural words into their singular forms. It is rule-based, so the rules can be supplemented and tuned to handle a wide range of exceptions. Individual words can be protected from stemming and can be given special-case stem words. Usually, general patterns cover wide classes of words.

The input token does not need to be lower case, but the stemming change will be lower case.

The Stemming Rules File

The default rules file is named `LucidStemRules_en.txt` and found in `$LWE_HOME/conf/solr/cores/collection/conf`. The rules file can be defined by changing the "rules" parameter in `schema.xml` for `com.lucid.analysis.LucidPluralStemFilterFactory`. These rules files are specified per text field type. It is expected that each natural language will have its own stemming rules file. This file is also specific to each collection.

If you wish to edit the stemming rules file, adhere to the following format guidelines.

- An exclamation point (!) indicates a comment or comment line to be ignored.
- White space is extraneous and ignored.
- Blank lines ignored.

Rules are evaluated in the order that they appear in the rules file, except that whole protected words and replacement words are processed before examining suffixes.

To restrict the minimum word length that is to be stemmed, simply create rules consisting of only question marks ('?') to match and protect words of those lengths. For example, to protect words of less than four characters in length, add three rules, before any other rules:

```
?     ! Protects 1-char words.
??    ! Protects 2-char words.
???   ! Protects 3-char words.
```

Types of Stemming Rules

Protected Word

Just write the word itself, it will not be changed.

- `word`

Replacement Word

Word will always be changed to a replacement word.

- `word => new-word`
- `word -> new-word`
- `word --> new-word`
- `word = new-word`

Protected Suffixes

Any matching word will be protected.

- `pattern suffix`

Pattern may start with an asterisk to indicate variable length. Use zero or more question marks to indicate that a character is required. Use a trailing slash if a consonant is required.

Examples:

- `?ass`
- `*??ass`
- `*???/ass`

Translation Suffix

The suffix of a matching word will be replaced with new suffix.

- `pattern suffix => new-suffix`

Pattern rules are the same as for protected suffixes. The pattern may be repeated before the replacement suffix for readability.

Examples:

- `*ses => se`
- `*ses -> *se`
- `*?/uses => se`

- `*???s =>`
- `*???s => *`

The latter two examples show no new suffix, meaning that the existing suffix is simply removed.

Example Stemming Rules File

Here is the default `LucidStemRules_en.txt` file that ships with LucidWorks Search, found in `$LWE_HOME/conf/solr/cores/collection/conf` (unique to each collection):

```
? ! Minimum of four characters before any stemming.
??
???
*ss ! No change : business
*'s ! No change : cat's - Handled in other filters.
*elves => *elf ! selves => self, elves, themselves, shelves
appendices => appendix
*indices => *index ! indices => index, subindices - NOT jaundices
*theses => *thesis ! hypotheses => hypothesis, parentheses, theses
*aderies => aderie ! camaraderie
*ies => *y ! countries => country, flies, fries, ponies, phonies, queries, symphonies
*hes => *h ! dishes => dish, ashes, smashes, matches, batches
*???oes => *o : potatoes => potato, avocadoes, tomatoes, zeroes
goes => go
does => do
?oes => *oe ! toes => toe, foes, hoes, joes, moes - NOT does, goes - but "does" is also
plural for "doe"
??oes => ??oe ! floes => floe
*sses => *ss ! passes => pass, bosses, classes, presses, tosses
*igases => *igase ! ligases => ligase
*gases => *gas ! outgases => outgas, gases, degases
*mases => *mas ! Christmases => Christmas, Thomases
*?vases => *vas ! canvases => canvas - NOT vases
*iases => *ias ! aliases => alias, bias, Eliases
*abuses => *abuse ! disabuses => disabuse, abuses
*cuses => *cuse ! accuses => accuse, recuses, excuses
*fuses => *fuse ! diffuses => diffuse, fuses, refuses
*/uses => *us : buses => bus, airbuses, viruses; NOT houses, mouses, causes
*xes => *x ! indexes => index, axes, taxes
*zes => *z ! buzzes => buzz
*es => *e ! spaces => space, files, planes, bases, cases, races, paces
*ras => *ra ! zebras => zebra, agoras, algebras
*us
*/s => * ! cats => cat (require consonant (not "s") or "o" before "s")
*oci => *ocus ! foci => focus
*cti => *ctus ! cacti => cactus
plusses => plus
gasses => gas
classes => class
```

```
mice => mouse
data => datum
!bases => basis
amebiases => amebiasis
atlases => atlas
Eliases => Elias
molasses
feet => foot
backhoes => backhoe
calories => calorie

! Some plurals that don't make sense as singular
sales
news
jeans
```

Choosing an Alternate Stemmer

Out of the box, the Lucid query parser comes with a basic plural stemmer that translates most plural words to their singular form. This should be sufficient for most applications. The stemming rules are all rule-based in an easy to read and write text file format that permits the addition of new rules and permits words to be protected or mapped specially. This permits flexibility for many more specialized applications.

If for some reason the administrator wishes to use an alternative stemmer, the change can be made manually in the `schema.xml` file or by using the FieldTypes API. Any stemming filter can be specified, but Lucid KStem is a typical alternative.

Information for LucidWorks Search in the Cloud Users
The instructions below refer to editing `schema.xml` to modify the stemmer used for each field type. Manual editing of the `schema.xml` file cannot be done by customers using LucidWorks Search hosted on AWS or Azure, but the same results can be achieved with the FieldTypes API.

Be sure to use the same stemmer class for both the index and query analyzers. If the stemmer classes do not match, the result can be that some queries can fail if terms were indexed according to different rules than those used by the Lucid query parser.

In general, it is best to *delete* the index and do a full re-indexing of the data collection whenever an *index* analyzer is radically changed, such as is the case when stemming filters or rules are changed. See Reindexing Content for more information about the options to reindex.

Other alternative stemming filters, such as Snowball and Porter, can be used instead of Lucid KStem if desired.

Using the FieldTypes API

The FieldTypes API is covered in depth in the section on the FieldTypes API.

The stemming rules are defined in the "analyzers" section for the field type. The analyzers section is considered an individual attribute as a whole, and it's not possible to update a single part of the analyzers rules without updating the entire section.

The `com.lucid.analysis.LucidPluralStemFilterFactory` class represents the default plural stemmer and will be shown in an API call in both the `index` and `query` section of the `analyzers` attribute. The `rules` parameter specifies the name of the text file that contains the plural stemming rules.

The `com.lucid.analysis.LucidKStemFilterFactory` class represents the Lucid KStem stemmer. To switch to this stemmer (or any other), make an API PUT call to the appropriate field type and update the `analyzers` attribute (in both the `index` and `query` sections).

For example, changing to the Lucid KStem stemmer for the `text_en` field type would require the following API call:

```
curl -X PUT -H 'Content-type: application/json'
-d '{"analyzers": {
      "index": {
        "char_filters": [ ],
        "token_filters": [
            {
                "catenateAll": "0",
                "catenateNumbers": "1",
                "catenateWords": "1",
                "class": "solr.WordDelimiterFilterFactory",
                "generateNumberParts": "1",
                "generateWordParts": "1",
                "splitOnCaseChange": "1"
            },
            {
                "class": "solr.LowerCaseFilterFactory"
            },
            {
                "class": "solr.ASCIIFoldingFilterFactory"
            },
            {
                "class": "com.lucid.analysis.LucidKStemFilterFactory"
            }
        ],
        "tokenizer": {
            "class": "solr.WhitespaceTokenizerFactory"
        }
    },
      "query": {
        "char_filters": [ ],
        "token_filters": [
            {
                "class": "solr.SynonymFilterFactory",
                "expand": "true",
                "ignoreCase": "true",
                "synonyms": "synonyms.txt"
            },
            {
                "class": "solr.StopFilterFactory",
                "ignoreCase": "true",
                "words": "stopwords.txt"
            },
            {
                "catenateAll": "0",
                "catenateNumbers": "0",
                "catenateWords": "0",
                "class": "solr.WordDelimiterFilterFactory",
                "generateNumberParts": "1",
                "generateWordParts": "1",
                "splitOnCaseChange": "1"
            },
            {
                "class": "solr.LowerCaseFilterFactory"
            },
            {
                "class": "solr.ASCIIFoldingFilterFactory"
            },
            {
                "class": "com.lucid.analysis.LucidKStemFilterFactory"
            }
        ],
        "tokenizer": {
            "class": "solr.WhitespaceTokenizerFactory"
        }
    }
}}' http://localhost:8888/api/collections/TestCollection/fieldtypes/text_en
```

Editing schema.xml

If you edit `schema.xml`, and search for the `text_en` field type, you should see that both its index and query analyzers have XML entries for the stemming filter that appear as follows:

```
<filter class="solr.ISOLatin1AccentFilterFactory"/>
<!-- <filter class="com.lucid.analysis.LucidKStemFilterFactory"/> -->
<filter class="com.lucid.analysis.LucidPluralStemFilterFactory"
rules="LucidStemRules_en.txt"/>
```

The `com.lucid.analysis.LucidPluralStemFilterFactory` class represents the default plural stemmer. The `rules` parameter specifies the name of the text file that contains the plural stemming rules.

The `com.lucid.analysis.LucidKStemFilterFactory` class represents the Lucid KStem stemmer, which is disabled by default using the standard `<!-` and `->` comment markers.

To disable the default plural stemmer and enable Lucid KStem, simply remove the comment markers from the latter and add them to the former. Do this same thing for both the index and query analyzers. The edited lines should now appear as follows:

```
<filter class="solr.ISOLatin1AccentFilterFactory"/>
<filter class="com.lucid.analysis.LucidKStemFilterFactory"/>
<!-- <filter class="com.lucid.analysis.LucidPluralStemFilterFactory"
rules="LucidStemRules_en.txt"/> -->
```

Deleting the Index

During application development, you might use sample data that is inappropriate for the production system. To remove this data, you can delete the entire index or just delete the content and crawl history for a single data source.

The easiest way to do this is to use the Admin UI to delete documents from a specific data source or an entire collection.

Another way to do this is to issue an API command using the Collections Index API. This API provides two methods to stop all running indexing tasks, clear the index, and clear any persistent crawl data (crawl history) for either the entire collection or a single data source.

This Will Delete ALL of Your Data

The following procedure to delete a collection should only be used if you are sure you want to delete **all documents** in your index. Once this command has been executed, there is **no way** to retrieve the content. If only some documents should be deleted, use the method to delete documents for a specific data source.

If you only want to clear the crawl history, the Data Source Crawl Data API provides a way to delete only the history for a data source, but not the content.

An alternative approach would be to issue a delete command directly to Solr with the following syntax. However, this will not stop running tasks nor clear persistent crawl data.

```
http://localhost:8888/solr/update?stream.body=<delete><query>id:\[\* TO
\*\]</query></delete>
```

Crawling Content

This section describes how to configure crawling with LucidWorks Search, to get the content to put in the indexes.

For the most part, crawling only requires configuring a data source with the UI or the API and starting the crawl. However, if using batch crawling, Access Control Lists, databases containing binary data, or an "external" crawler, there may be additional configuration you'll want to do.

Start with the Overview of Crawling to understand how the crawlers work. Then dive into the detailed sections as needed:

Overview of Crawling

LucidWorks Search has integrated several crawlers to make adding content to the index easier and more straightforward.

A *crawler* is a program which understands how to connect to a remote repository (or several types of repositories), find documents within the repository, and retrieve the documents for indexing by the system. A synonym in some contexts is a *connector*, but there are differences between the terms. A crawler discovers new documents on its own and makes decisions about which documents to retrieve, based on rules provided to it by its own code or by configuration. A connector is more passive - it connects to a repository and pulls all the documents, without the ability to make decisions; interpreting rules and making decisions would be up to the crawler which controls the connector.

As each repository is different, each crawler needs information to connect to a specific repository, such as the network address of the repository and any required authentication information. This information is provided to the crawler by creating a *data source*.

The data source is the central way in which you interact with the crawlers. There is one defined per repository, filesystem, website, etc. So, for example, if you want to index three websites, you'll create three Web Data Sources. Three S3 buckets, then you'll create three S3 Data Sources.

Topics covered in this section:

For the most part, we've tried to make each data source consistent in terms of the options provided, but there are differences between the crawlers and their capabilities. This leads to differences when configuring data sources of different types, and differences in performance and behavior of the crawlers themselves while retrieving documents and passing them along the indexing process.

The Crawl Process

When starting a crawl, the crawler associated with a specific data source uses the information saved in the data source configuration to connect to the repository and find documents. Most of the data sources support inclusion or exclusion parameters to define the types of documents (or paths to documents) that should be indexed. The crawlers use that information to know what pages to retrieve for eventual indexing.

The crawlers do not actually index content. A crawler retrieves the pages, and passes them to a *parser*, which prepares the documents for the indexing process. The parser handles breaking the documents into their parts, identifying fields within the documents and normalizing data so it can be more easily consumed in the index. In most cases, the crawlers use Apache Tika for parsing.

The exception to this is the Aperture crawler, which has its own parser embedded within it. In cases where the Aperture parser fails to parse a document, Tika is used as a fall-back. However, documents that were successfully parsed by the Aperture crawler do not get another pass through Tika. There is no way to change this behavior at this time.

Once documents have been retrieved and parsed, they are passed to the *UpdateController* which pushes them into the index using SolrJ, a common client used for indexing content in Solr. This process also performs *field mapping*, where the extracted fields from a document can be mapped to other fields.

Data Source Options

Logging

The crawlers log information about attempts to access documents and the results of those attempts. The log is kept in `$LWE_HOME/data/logs` in a file named `connectors.<YYYY_MM_DD>.log`.

In general, the crawlers will:

- print one line to the log with the document ID when it has successfully accessed a document, describing the status (New, Updated, Deleted, etc.). In cases where the document could not even be accessed, this may lead to the attempt not being recorded in the logs.
- not log documents of unknown type that cannot be processed as plain text.
- not log documents that fail parsing.
- not add documents that fail parsing.

Each of these behaviors can be changed in most crawlers, which would allow more information to be added to the log or more documents added to the index. With some crawlers, however, the default behaviors are the only options. More information for each data source type is available in the documentation for the Admin UI and the REST API.

Scheduling

Each data source can be scheduled to run at regular intervals. Using the Admin UI, it is only possible to schedule crawling at specific intervals (hourly, daily, weekly), but using the REST API, more complex schedules can be constructed. It is, however, only possible to have a single schedule for each data source.

Field Mapping

Field Mapping provides the ability to map fields in documents to fields or dynamic field rules already defined in LucidWorks or add fields to incoming documents. This can be done generically when an unexpected field is introduced or specifically for known incoming fields. The mapping rules can be manipulated via the Admin UI from the Data Source Details screen, or with either the Data Sources API or the Field Mapping API.

Data Source Types

LucidWorks Search currently supports 8 crawlers and 13 types of data sources. When using the Admin UI, the selection of a crawler is hidden; when using the REST API, the selection of a crawler is a required attribute.

The table summarizes the types of content repositories that can be crawled:

Crawler	Data Source Types Supported	Capabilities	Limitations (not comprehensive; see documentation for each type for full details)

Aperture	• Websites • Filesystems	• Can crawl websites and filesystems. • Stores a history of documents that have been seen before. • Indexes data contained in `<META>` tags. • Web crawling will respect robots.txt rules or can be configured to ignore them.	• The Aperture crawler is not designed for large-scale crawls of more than about 10,000 pages or files in a single crawl. • It is a single-threaded process, meaning that one data source will only use a single server process to crawl sites. This can make a long crawl take a long period of time to complete. • Multiple data sources all use the same "triple-store", which is a database inside Aperture that keeps track of web pages visited. If multiple data sources are running at the same time, the triple-store can get easily corrupted. It's highly recommended to avoid running multiple Aperture-based crawls at the same time. • Doesn't use Apache Tika for document parsing and may not be as accurate with some documents as Tika (however, if it cannot parse a document at all, it will pass that document to Tika for parsing).

JDBC	• Databases	• Allows indexing of databases. • Supports nested queries for complex data environments. • Supports delta queries to limit subsequent crawls on only new or changed table rows.	• The LucidWorks implementation is based on the DataImportHandler, which can be difficult to precisely configure in unique environments. • Requires uploading a driver before it can be used. • Converting date types can be problematic.
Google Connector Manager	• SharePoint Repositories	• Indexes all content in the SharePoint repository (files, discussion boards, calendars, contacts, sites, images, etc.). • Support SharePoint security configuration. • Can add new connectors supported by the Google Connector Manager framework.	• Must install additional Web services to work properly. • Security options can be complex to configure.

SolrXML	• SolrXML files	• Easy to understand XML structure. • Many users already have documents in this format due to prior use of Solr. • Can point it to a directory of files instead of one at a time. • Can add a unique identifier to each document as it's indexed if it doesn't have one already.	• Not a generic XML indexer; documents must structured in a very specific way.
Filesystem	• Amazon S3 buckets • SMB/Windows Shares • Hadoop Distributed Filesystems (HDFS) • Hadoop over S3 • FTP servers • Local Filesystems	• Provides access to multiple remote filesystems.	• Must allow the LucidWorks server access to the remote systems. • Hadoop crawls are throttled to prevent overloading the system.

Twitter Stream	• Twitter Stream API	• Allows filtering indexed tweets by userID, location, or keywords.	• Will continue to crawl indefinitely unless manually stopped or controlled with a parameter that's only available via the REST API. • The LucidWorks implementation is still in beta phase; may include unknown bugs.
High-Volume HDFS	• Hadoop Distributed Filesystems (HDFS)	• Allows unthrottled crawling of HDFS systems.	• Must design your LucidWorks cluster appropriately to take full advantage of the speed capabilities. • The LucidWorks implementation is still in beta phase; may include unknown bugs.
External	• Push to LucidWorks	• Can use SolrJ or another established Solr indexing process to get documents into LucidWorks. • Full access to field mapping capabilities that other crawlers use.	• The documents or processes for crawling must be prepared in advance.

Related Topics

- Data Sources in the Admin UI
- Data Sources with the REST API
- Custom Connector Guide

Supported Filetypes

LucidWorks Search crawlers can identify many different file formats (MIME types), and can extract text and metadata from the MIME types listed in the table below. Even if the crawlers cannot extract data from a file, it can often at least recognize the file type and index basic information about the file, such as the filename and its metadata. Many of the crawlers have settings that allow how to handle the situation where the MIME type is not supported.

Note that extracting data from third party proprietary file formats is often difficult and may result in irregular text being extracted and indexed. If you encounter a format that is supported, but does not get properly extracted, please share the information with Lucid Support, including the file, if possible.

Supported File Formats

Name	MIME Type(s)	Notes
HTML	text/html	
Images	image/jpeg, image/png, image/tiff	Metadata Only
Mail	message/rfc822 and message/news	Some mime based mail attachments can be extracted.
MP3 Metadata	audio/mpeg	Metadata only
Microsoft Office	Word, PowerPoint, Excel, MS Publisher, Visio	All applications are trademarks of the Microsoft Corporation
Open Office	OpenDocument and StarOffice documents	
OpenXML	Microsoft's latest Office format	
Adobe Portable Document Format	application/pdf	PDF is a trademark of Adobe
Plain Text	text/plain	
Quattro	application/x-quattropro, application/wb2	Trademark of Corel
Rich Text Format	text/rtf	
eXtensible Markup Language (XML)	text/xml	
Archives	application/zip, application/gzip, application/x-tar	

Troubleshooting Document Crawling

LucidWorks Search crawling events are logged to the `connectors.<YYYY_MM_DD>.log` file, found in the `$LWE_HOME/data/logs` directory.

Serious exceptions will be reported to the LucidWorksLogs collection, which you can search as you can any other collection through the default Search UI. In addition, the Admin UI provides some visibility into errors during crawling by showing them on the **Server Log** page, found under the Status menu. That page also allows access to browse all the log files without having to access the server.

Problems such as a document not being found or access denied will not be reported the the LucidWorksLogs collection, but will show in the Admin UI and in the Data Source Status/History APIs as "not found". This may make it difficult to find which documents were skipped, but a review of the log file may yield further information.

In general, the crawlers will:

- print one line to the log with the document ID when it has successfully accessed a document, describing the status (New, Updated, Deleted, etc.). In cases where the document could not even be accessed, this may lead to the attempt not being recorded in the logs. This can be changed by modifying the setting "Log Extra Detail" in crawlers that support it.
- not log documents of unknown type that cannot be processed as plain text. This can be changed by modifying the setting "Log warnings for unknown mime types" in crawlers that support it.
- not log documents that fail parsing. This can be changed by modifying the setting "Fail unsupported file types" in crawlers that support it.
- not add documents that fail parsing. This can be changed by modifying the setting "Add failed docs" in crawlers that support it.

Understanding Crawl Errors

Crawling is dependent on a number of factors. In order for a site to be crawl-able, several things must be aligned:

- The repository must be supported by one of the crawler and data source types.
- The repository must be accessible to the LucidWorks Search server. If authentication is required to access the repository, the data source must support the authentication type and the correct credentials supplied.
- The documents must be parseable, so the fields and content can be extracted.
- The specific data source settings must be configured to include the specific documents.

For example, if I have a file system with 100 PDF documents, each of which are OCR scans and 100Mb in size, the PDF documents: a) may not be parseable because OCR scans are images and, b) may exceed the maximum file size configured in the data source (the default is 10Mb). In this example, the files would be skipped by the crawler, which is not considered a serious exception and is generally only logged when the data source setting to "Log extra detail" is selected. Then the skipped files would be found in the log file with a format like this:

```
INFO filesystem.FileSystemCrawler - File <file-URL> exceeds the maximum size specified for
this data source. Skipping.
```

```
WARN No extractor for <file format>; Skipping: <document-URI>
```

Possible Errors

This information is provided to help you find the errors in the log file; precise troubleshooting requires information about the documents and system environment. If a document causes an error (besides being too large or the system being out of memory), it may be helpful to try to isolate it and try again to be sure it is the document causing the problem and not some other system error that may have occurred at the same time.

In each of the errors below, the document URI will be listed. For files this will be the path and filename, for websites it would be the URL, and for other data source types a base document URI will be configured based on how the data source is configured.

Exception

```
WARN Exception while crawling: <document-URI> <exception-with-stack-trace>
WARN Doc failed: <exception-with-stack-trace>
WARN Doc failed: <document-URI> - cause: <exception-cause-message>
```

PDF files are notorious for causing exceptions in their processing. These errors are not always fatal, but may cause all or part of the file to be skipped.

```
WARN util.PDFStreamEngine - java.io.IOException: Error: expected hex character and not :32
WARN util.PDFStreamEngine - java.io.IOException: Error: expected the end of a dictionary.
```

Out of memory

```
WARN File caused an Out of Memory Exception, skipping: <document-URI>
<exception-with-stack-trace>
WARN Doc failed: <exception-with-stack-trace>
WARN Doc failed: <document-URI> - cause: <OOM-exception-message>
```

SubCrawlerException

```
WARN Doc failed: <exception-with-stack-trace>
WARN Doc failed: <document-URI> - cause: <exception-message>
```

Unknown file type

```
WARN Doc failed: Could not find extractor: <document-URI>
```

In this case, this warning will be seen in the logs but will not be reported in the LucidWorksLogs collection.

I/O error

```
WARN IO Exception processing: <document-URI> <exception-with-stack-trace>
WARN Doc failed: <exception-with-stack-trace>
WARN Doc failed: <document-URI> - cause: <exception-message>
```

HTML/XML/XHTML parsing errors

```
WARN Doc failed: <exception-with-stack-trace>
WARN Doc failed: <document-URI> - cause: <exception-cause-message>
```

This is another case where a warning will be seen in the logs but will not be reported in the LucidWorksLogs collection.

Related Topics

- System Directories and Logs

Crawling Windows Shares with Access Control Lists

LucidWorks Search can crawl Windows Shares (SMB filesystems) and the Access Control Lists (ACLs) associated with shared files and directories. The ACL information can then be used to limit users' searches to the content they are permitted to access. This page describes how to configure using ACLs to control search results based on the user's permissions

As of LucidWorks Search v2.5, it's possible to configure ACL and Active Directory connections on a per-data source basis. This means that you can simply create a Windows Share data source with either the UI or the API, configure the connection to the Active Directory server, define if you want to trim results based on user authorizations, and then crawl the content.

When configuring the connection between LucidWorks Search and Active Directory, keep these requirements in mind:

- Credentials with READ and ACL READ permissions for accessing the Windows share. We recommend that you create a special user for this purpose.
- Credentials with read-only access to the Active Directory LDAP. This is used for search-time filtering, and we recommend that you create a special user for this purpose.

Permissions with Access Control Lists

The following model is implemented as a search filtering component by default:

Group READ Access	Subgroup READ Access	User READ Access	Search Result Returned?
o (permit)	o	o	o
o	× (deny)	o	×
o	o	×	×

o	×	×	x
×	o	o	×
×	×	o	×
×	o	×	×
×	×	×	×
o	- (not set)	o	o
o	o	-	o
o	-	-	o
-	o	o	o
-	-	o	o
-	o	-	o
-	-	-	×

To understand this table, read the rows left to right. For example, in the first row, we see that the user's main group, subgroup, and individual permissions all allow READ access to a shared resource, so the search result is returned. In the second row, we see that the user's main group and user's individual permissions allow READ access, but the user's subgroup's permissions do not, so no search result is returned to the user.

How SMB ACL Information Is Stored In The Index

For each file that is crawled through the SMB data source the `acl` field is populated with data that can be used at search time to filter the results so that only people that have been granted access at the user level or through group membership can see them. Two kinds of tokens are stored: Allow and Deny. The format used is as follows:

Allow:
`WINA<SID>`

Deny:
`WIND<SID>`

Where `SID` is the security identifier commonly used in Microsoft Windows systems. There are some well known SIDs that can be used in the `acl` field to make documents that are crawled through some other mechanism than by using SMB data source behave, from the `acl pow`, the same way as the crawled SMB content:

SID	Description
S-1-1-0	Everyone.

S-1-5-domain-500	A user account for the system administrator. By default, it is the only user account that is given full control over the system.
S-1-5-domain-512	Domain Admins: a global group whose members are authorized to administer the domain. By default, the Domain Admins group is a member of the Administrators group on all computers that have joined a domain.
S-1-5-domain-513	Domain Users.

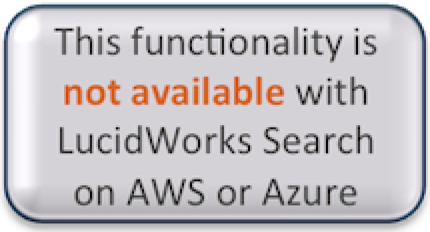
Note that some of the listed SIDs contain a `domain` token. This means that the actual SIDs differ from system to system. To find out the SIDs for particular user in particular system you can use the information provided by the Windows command line tool `whoami` by executing command `whoami /all`.

You can populate the `acl` field in your documents with these Windows SIDs to make them searchable in LucidWorks Search. For example, if you wanted to make some documents available to "Everyone" you would populate the `acl` field with the `WINAS-1-1-0` token. If you wanted to make all docs from one data source available to everybody you can use the literal definitions in the data source configuration.

Related Topics

- Filtering API
- Search Handler Components API
- LDAP Integration

Indexing Binary Data Stored in a Database


This functionality is not available with LucidWorks Search on AWS or Azure

The Database crawler in LucidWorks Search does not automatically discover and index binary data you may have stored in your database (such as PDF files). However, you can configure LucidWorks to recognize and extract the binary data correctly by modifying the data source configuration file (which does not exist until you create a JDBC data source).

For detailed information about working with JDBC data sources, see Create a New JDBC Data Source or the Data Sources API.

After you have created a Database data source, you can find the configuration file in `$LWE_HOME/conf/solr/cores/collection/conf/dataconfig_id.xml`. If you are familiar with Solr, you will recognize this file as a Data Import Handler configuration file.

Follow these steps to modify the configuration file:

1. Add a `name` attribute for the database containing your binary data to the `dataSource` entry.
2. Set the `convertType` attribute for the `dataSource` to `false`. This prevents LucidWorks from treating binary data as strings.
3. Add a `FieldStreamDataSource` to stream the binary data to the Tika entity processor.
4. Specify the `dataSource` name in the `root` entity.
5. Add an entity for your `FieldStreamDataSource` using the `TikaEntityProcessor` to take the binary data from the `FieldStreamDataSource`, parse it, and specify a field for storing the processed data.
6. Reload the Solr core to apply your configuration changes.

After you have modified the data source configuration file you should not modify the data source from the LucidWorks Admin UI because LucidWorks will automatically overwrite the `convertType` attribute, and indexing for the modified data source will fail.

Example

In this example there is a MySQL database called `test` containing a table called `documents` that contains PDF data in a column called `binary_content`. When the data source is first created, the data source configuration file (in `$LWE_HOME/conf/solr/cores/collection/conf/dataconfig_id.xml`) looks like this:

```
<dataConfig>
  <dataSource autoCommit="true" batchSize="-1" convertType="true"
driver="com.mysql.jdbc.Driver" password="admin"
   url="jdbc:mysql://localhost/test" user="root"/>
  <document name="items">
    <entity name="root" preImportDeleteQuery="data_source:9" query="SELECT * FROM
documents"
     transformer="TemplateTransformer">
      <field column="data_source" template="9"/>
      <field column="data_source_type" template="Jdbc"/>
      <field column="data_source_name" template="MySQL"/>
    </entity>
  </document>
</dataConfig>
```

To modify this data configuration file, follow these steps:

1. Add the `name` attribute to the `dataSource` and set `convertType` to `false`:

   ```
   <dataSource autoCommit="true" batchSize="-1" convertType="false"
   driver="com.mysql.jdbc.Driver" password="admin"
    url="jdbc:mysql://localhost/test" user="root" name="test"/>
   ```

   Specify another `dataSource` called `fieldReader` to handle the binary data:

   ```
   <dataSource name="fieldReader" type="FieldStreamDataSource" />
   ```

2. Specify the data source for the root entity:

   ```
   <entity name="root" preImportDeleteQuery="data_source:9" query="SELECT * FROM
   documents"
    transformer="TemplateTransformer" dataSource="test">
   ```

3. Add an entity for the `fieldReader` data source specifying the `TikaEntityProcessor` and a `dataField` for storing the processed binary data:

   ```
   <entity dataSource="fieldReader" processor="TikaEntityProcessor"
   dataField="root.binary_content" format="text">
     <field column="text" name="body" />
   </entity>
   ```

4. Restart LucidWorks Search to apply your configuration changes.

For this example, the final configuration file looks like this:

```
<dataConfig>
  <dataSource autoCommit="true" batchSize="-1" convertType="false"
driver="com.mysql.jdbc.Driver" password="admin"
   url="jdbc:mysql://localhost/test" user="root" name="test"/>
  <dataSource name="fieldReader" type="FieldStreamDataSource" />
  <document name="items">
    <entity name="root" preImportDeleteQuery="data_source:9" query="SELECT * FROM
documents"
      transformer="TemplateTransformer"
      dataSource="test">
       <field column="data_source" template="9"/>
       <field column="data_source_type" template="Jdbc"/>
       <field column="data_source_name" template="MySQL"/>
       <entity dataSource="fieldReader" processor="TikaEntityProcessor"
dataField="root.binary_content" format="text">
         <field column="text" name="body" />
       </entity>
    </entity>
  </document>
</dataConfig>
```

Related Topics

- Create a New JDBC Data Source
- Data Sources API
- Data Import Handler

Using the High Volume HDFS Crawler

The High Volume HDFS (HV-HDFS) Crawler is a Map-Reduce-enabled crawler designed to leverage the scaling qualities of Apache Hadoop while indexing content into LucidWorks Search. In conjunction with LucidWork's usage of SolrCloud, applications should be able to meet their large scale indexing and search requirements.

To achieve this, HV-HDFS consists of a series of Map-Reduce-enabled Jobs to convert raw content into documents that can be indexed into LucidWorks which in turn relies on the Behemoth project (specifically, the LWE fork/branch of Behemoth hosted on Github) for Map-Reduce-ready document conversion via Apache Tika and writing of documents to LucidWorks.

The HV-HDFS crawler is currently marked as "Early Access" and is thus subject to changes in how it works in future releases.

System Requirements

- Apache Hadoop. We've tested with Hadoop 0.20.2 and 1.0. Other versions will likely work as well, but we recommend thorough testing for compatibility.
- LucidWorks running in SolrCloud mode.

Please note, explanation of setting up Hadoop is beyond the scope of this document. We recommend reading one of the many tutorials found online or one of the books on Hadoop.

Using HV-HDFS in LucidWorks

Once Hadoop and SolrCloud are ready, configure a data source within LucidWorks, either with the High Volume HDFS data source type in the Admin UI or using the Data Sources API.

At the current time, the HV-HDFS crawler does not support field mapping. The consequence of this is that most metadata found in documents will be dropped and not indexed with the document. Only the main bocy text and the document URL are retained and indexed. We will improve this functionality in a future release.

Unlike other crawlers in LucidWorks Search, this crawler currently has no way of tracking which content is new, updated, or deleted. Thus, all content found is reported as "new" with each crawl.

How it Works

The HV-HDFS crawler consists of three stages designed to take in raw content and output results to LucidWorks. These stages are:

1. Create one or more Hadoop SequenceFiles from the raw content, where the key in the SequenceFile is the name of the file and the value is the contents of the file stored as a BehemothDocument. This process is currently done sequentially. A BehemothDocument is an intermediate form designed to allow for reuse across several systems without having to reprocess the raw content. The results are written to the Work Path area with a path name with a pattern of `inputToBehemoth_<Collection Name><Data Source Id><System Time in Milliseconds>`.
2. Run a Map-Reduce job to extract text and metadata from the raw content using Apache Tika. This is similar to the LucidWorks approach of extracting content from crawled documents, except it is done with Map-Reduce.
3. Run a Map-Reduce job to send the extracted content from HDFS to LucidWorks using the SolrJ client. This implementation works with SolrJ's CloudServer Java client which is aware of where LucidWorks is running via Zookeeper.

⚠ The processing approach is currently all or nothing when it comes to ingesting the raw content and all 3 steps must be completed each time, regardless of whether the raw content hasn't changed. Future versions may allow the crawler to restart from the SequenceFile conversion process.

Differences from Other Hadoop Crawlers in LucidWorks

While the HV-HDFS, Hadoop File System (HDFS) and Hadoop File System over S3 (S3H) crawlers all use Hadoop to access Hadoop's distributed file system, there is a big difference in how they utilize those resources. The HDFS and S3H data sources are designed to be polite and crawl through the content stored in HDFS just as if they were crawling a web site or any other file system.

The HV-HDFS crawler, on the other hand, is designed to take full advantage of the scaling abilities of Hadoop's Map-Reduce architecture. Thus, it runs jobs using all of the nodes available in the Hadoop cluster just like any other Hadoop Map-Reduce job. This has significant ramifications for performance since it is designed to move a lot of content, in parallel, as fast as possible (depending on Hadoop's capabilities), from its raw state to the LucidWorks index. Thus, you will need to design your LucidWorks SolrCloud implementation accordingly and make sure to provision the appropriate number of nodes.

Conversion to SequenceFiles

The first step of the crawl process converts the input content into a SequenceFile. In order to do this, the entire contents of that file must be read into memory so that it can be written out as a BehemothDocument in the SequenceFile. Thus, you should be careful to ensure that the system does not load into memory a file that is larger than the Java Heap size of the process. In certain cases, Behemoth can work with existing files such as SequenceFiles to convert them to Behemoth SequenceFiles. Contact Lucid Imagination for possible alternative approaches.

Example: Indexing Shakespeare with Map-Reduce

The following steps demonstrate indexing the complete works of Shakespeare using the LucidWorks Search HV-HDFS crawler.

Prepare the Content

1. Download the data into a temporary directory.
   1. `cd /tmp`
   2. `mkdir shakespeare`
   3. `wget http://www.it.usyd.edu.au/~matty/Shakespeare/shakespeare.tar.gz`
2. Unpack the archive with `tar -xf shakespeare.tar.gz -C shakespeare`
3. Load the data to Hadoop with `<PATH TO HADOOP>/bin/hadoop fs -put shakespeare ./shakespeare`

Setup LucidWorks

Start LucidWorks in SolrCloud mode and make note of where Zookeeper is running. See the section on Using SolrCloud in LucidWorks for more information on how to start in SolrCloud mode.

Setup the Data Source and Run

Create a new data source using either the Admin UI or the Data Source API, as described above. The "path" would be the location of the Hadoop NameNode, such as, `hdfs://lucidserver:54310/user/lucidworks/shakespeare`.

Once the data source is created, you can use the Hadoop UI to track the progress of the various Map-Reduce jobs. You can also inspect your specified work path to see the intermediate files using the Hadoop filesystem commands (e.g., `hadoop fs -ls`).

Related Topics

- Using SolrCloud in LucidWorks
- Apache Hadoop
- Behemoth
- Scaling Solr Indexing With SolrCloud, Hadoop and Behemoth

Suggestions for External Data Sources

In some cases, it may not be possible to use the crawlers included with LucidWorks Search to index content, such as an email archive or a Web repository that's best accessed via an API. Instead, another process may be possible, such as using SolrJ, to feed documents directly to Solr. In that situation, LucidWorks would not normally know about the documents and would not be able to include information about the data source in facets or display statistical data about the data source in the Admin UI.

Fortunately, there is a way to create an 'external' data source to add fields to the document so LucidWorks will treat the documents the same as documents found via the embedded crawlers. The data source can be created either via the Sources screen in the Admin UI or with the Data Sources API.

The 'external' data source type is different from the other data source types in that it is the only one that uses a "push" mechanism to push content into LucidWorks (and, by extension, Solr), while the other data source types use a "pull" model to go and get content for processing. Because the content is being pushed from an external process, these suggestions will ensure that they are processed consistently by LucidWorks.

Add the fm.ds Parameter to the Push Request

External data sources support field mapping in the same way all other data source types do (by specifying the mapping with the Data Sources API or via the Admin UI). The `fm.ds` parameter in the request allows LucidWorks to know which data source's field mapping rules to apply to the content as it is being processed. Without this parameter in the request, the default field mapping will be applied, even if the field mapping has been customized for the data source.

The default search UI included with LucidWorks relies on the `title` and `body` fields being populated in order to display information about results to users; the `author` and `lastModified` fields are also used for display and faceting. If your custom search UI uses these or other fields for display of results, it's recommended that the documents pushed directly to Solr include content in the those fields for a consistent user experience.

The value of the `fm.ds` parameter should match the ID of the data source, which can be found with either the Data Sources API (by reviewing the full list of data sources) or via the Admin UI (by inspecting the URL of the Edit Settings screen for the specific data source ID). If the ID supplied does not match an existing data source ID, an error will be returned and the documents will not be loaded.

It is also possible to supply '-1' for the `fm.ds` ID. In cases where there is only one external type of data source, the fields for that data source will be filled in to the documents and the mapping for that data source will be applied. If there are zero external data sources or more than one external data source, an error will be returned and the documents will not be loaded.

Add lucidworks_fields to Incoming Content

When LucidWorks crawlers acquire content, certain fields related to the data source are added to each document to help identify the documents as belonging to the data source for use in statistics, faceting, and document deletion (if necessary). This is done via an attribute called `lucidworks_fields` (which is shown as "Create LucidWorks fields" in the Edit Mapping screen of the Admin UI). The default for this attribute is "true", which means the fields will be added to all incoming documents, so usually no editing is required to add these fields as long as the `fm.ds` parameter has been added to the update request.

The fields added to each document are from the data source, but have different names. This table shows the relationship between the data source attribute name and the fields added to documents:

Data Source Attribute	Field Name (in `schema.xml`)
id	data_source
type	data_source_type
name	data_source_name

Schedule the Data Source with the callback Attribute

The external data source can be scheduled in the same ways as any other data source to periodically update, delete or add new documents. The `callback` attribute must be set in order for the schedule to work correctly. This attribute requires LucidWorks to issue an HTTP GET request to trigger the push when a job is started, either via the scheduler or manually via API or the UI. This call occurs just after LucidWorks updates the field mapping, so if the mapping is modified between schedules incoming documents get the new mapping.

Examples

Using the Data Sources API, a new data source could be created with these settings:

```
curl -X post -H 'Content-type: application/json'
http://localhost:8888/api/collections/collection1/datasources -d
'{
   "name":"Test External #1",
   "type":"external",
   "crawler":"lucid.external",
   "source_type":"Raw SolrXML",
   "source":"Solr update"
}'
```

The output of this command would be as follows:

```
{
  "commit_on_finish":true,
  "verify_access":true,
  "indexing":true,
  "source_type":"Raw SolrXML",
  "collection":"collection1",
  "type":"external",
  "crawler":"lucid.external",
  "id":3,
  "category":"External",
  "source":"Solr update",
  "name":"Test External #1",
  "parsing":true,
  "commit_within":900000,
  "caching":false,
  "max_docs":-1
}
```

Then a document such as this could be added directly to Solr:

```
curl 'http://localhost:8888/solr/collection1/update?fm.ds=3\&commit=true' -H
'Content-type: text/xml' --data-binary '
 <add>
  <doc>
   <field name="id">testdoc</field>
   <field name="body">test</field>
  </doc>
 </add>'
```

Here is an example document using SolrJ:

```
...
String dsId = "3";
SolrInputDocument doc = new SolrInputDocument();
doc.addField("id", "1234");
doc.addField("body", "test");

SolrServer server = new
CommonsHttpSolrServer("http://localhost:8888/solr/collection1");
UpdateRequest req = new UpdateRequest();
req.setParam("fm.ds", dsId);
req.add(doc);
req.process(server);
```

Related Topics

Indexing Documents Directly to Solr

Solr provides many ways to index content, and these can be used in addition to or instead of the crawlers built into LucidWorks Search. Solr can index XML (in a specific Solr format), CSV files and JSON formats. In addition, the popular SolrJ provides another way to index data that can be used with LucidWorks Search.

If you push documents directly to Solr, the LucidWorks Admin UI will be unable to display statistical information about those documents. The LucidWorks data source type "external" would allow you to integrate documents pushed directly to Solr with documents indexed from the crawlers and get statistics such as number of documents per data source in the Admin UI. In addition, the external data source also allows using LucidWorks data source field mapping functionality. For more information, see Suggestions for External Data Sources; the information contained below is still valid, but would be slightly modified.

Indexing Solr XML

One way to integrate LucidWorks with a custom data source is to dump the data from that data source into XML files formatted in this way, and index them as a Solr XML data source. LucidWorks has built-in support for indexing a directory tree of Solr XML files and scheduling periodic re-indexing. Alternatively, the XML files can easily be posted into LucidWorks and Solr externally using curl, the REST API, or other tools that can HTTP POST, like this:

```
curl http://localhost:8888/solr/collection1/update --data-binary @filename.xml
  -H 'Content-type:text/xml; charset=utf-8'
```

Solr natively digests a simple XML structure like this:

```
<add>
 <doc>
  <field name="fieldname1">field valueA</field>
  <field name="fieldname2">field valueB</field>
 </doc>
 <doc>
  <field name="fieldname3">multivalue1</field>
  <field name="fieldname3">multivalue2</field>
 </doc>
</add>
```

The `<add>` structure supports multiple `<doc>` declarations and each `<doc>` supports multiple `<field>` declarations. Fields can be multi- or single-valued, depending on the `schema.xml` configuration. The LucidWorks Search Fields screens provide a handy user interface for managing field properties, including the multivalued setting.

Solr's XML format can perform other operations including deleting documents from the index, committing pending operations, and optimizing an index (a housekeeping operation). For more information on these operations, as well as adding documents, refer to Solr's Update XML Messages.

Indexing Column (Comma) Delimited Data

The following section uses an example to illustrate how to index delimited text with LucidWorks.

1. Save the following simple comma-separated data as sample_data.text:

```
id,title,categories
1,Example Title,"category1,category2"
2,Another Record Example Title,"category2,category3"
```

2. Configure the index schema using the Fields editor in the Admin UI as follows:
   - At the bottom of the page, click **Add new field** to get a blank field form
   - Add a new field with the following settings:
     - Name: categories
     - Type: string
     - Stored: checked
     - Multi-valued: checked
     - Short Field Boost: none
     - Search by Default: checked
     - Include in Results: checked
     - Facet: checked
3. Save and apply those settings.
4. Now index the CSV data from the command-line using curl:

```
curl
"http://localhost:8888/solr/collection1/update/csv?commit=true&f.categories.split=tr

--data-binary @sample_data.txt -H 'Content-type:text/plain; charset=utf-8'
```

5. You can also make the file pipe-delimited, like this:

```
id|title|categories
3|Three|category3
4|Four|category4,category5
```

And then you can index using this command:

```
curl
"http://localhost:8888/solr/collection1/update/csv?commit=true&f.categories.split=tr

--data-binary @pipe.txt -H 'Contenttype:text/plain; charset=utf-8'
```

For a full description of all CSV options, see the Solr UpdateCSV documentation.

Related Topics

- Suggestions for External Data Sources

From our Apache Solr Reference Guide:

- Indexing and Basic Data Operations
- Using SolrJ

Integrating Nutch

LucidWorks Search includes support for "external" data sources (also known as "push crawlers"). While the built-in LucidWorks crawlers use the "pull" model (meaning that LucidWorks initiates the crawl and actively discovers new or updated resources), push crawlers are external processes that manage the discovery and sending of new and updated documents for indexing outside of the LucidWorks crawler framework.

Apache Nutch is a framework for building and running large-scale Web crawling using Hadoop map-reduce clusters (see http://nutch.apache.org/ for more information). Recent releases of Nutch rely on Solr for indexing and searching. From the point of view of LucidWorks, Nutch can be integrated as an "external" or "push" crawler.

The following sections describe step-by-step how to integrate a Nutch 1.4 crawler (or Nutch trunk) with LucidWorks.

Solr indexer

Nutch comes with a tool for map-reduce indexing to Solr called `SolrIndexer`. From the command-line, this tool is invoked like this:

```
nutch solrindex http://localhost:8983/solr/collection1 db -linkdb linkdb [-params
k1=v1,k2=v2] segment1 segment2 [...]
```

Support for the `-params` option exists in Nutch trunk, post 1.4 release, or if you apply the patch found in NUTCH-1212).

Field mapping in Nutch

Nutch uses indexing plugins to construct the outgoing documents, and these plugins add various fields with various names. These field names do not necessarily match the default LucidWorks `schema.xml` for a collection. Nutch provides a limited facility to adjust these names (see `$nutch_home/conf/solrindex-mapping.xml`). This field mapping facility is often enough in simple cases to re-map field names so that they match the LucidWorks schema.

However, this solution has some drawbacks:

- This mapping is static for all indexing jobs that use the same job file (or the same `conf` directory in the case of a non-distributed Nutch installation) and changing it requires rebuilding of the job file, which can be cumbersome.
- There is no easy way to add fields that are useful for managing documents in LucidWorks (such as `data_source_type`, `data_source_name` or `data_source`), short of implementing a new Nutch indexing plugin.
- the field mapping in `solrindex-mapping.xml` cannot be managed from the LucidWorks Admin UI.

Fortunately, there is a better solution to this problem which is to use the field mapping functionality in LucidWorks, defined as part of the External data source type definition, in combination with the `-params` option for `SolrIndexer`.

Field mapping in LucidWorks

External processes that submit documents to LucidWorks can be integrated using the External data source type. When you define a new data source in LucidWorks, one of its properties is `field_mapping`. With the Data Sources API, the JSON serialization looks similar to this:

```
...
"mapping": {
   "datasource_field": "data_source",
   "default_field": null,
   "dynamic_field": "attr",
   "literals": {},
   "lucidworks_fields": true,
   "mappings": {
   "acl": "acl",
   "author": "author",
   "batch_id": "batch_id",
   "content": "body",
   "content-encoding": "characterSet",
   "content-length": "fileSize",
   "content-type": "mimeType",
   "contentcreated": "dateCreated",
   "contentlastmodified": "lastModified",
   ...
   },
"multi_val": {
   "acl": true,
   "author": true,
   "body": false,
   "dateCreated": false,
   "description": false,
   "fileSize": false,
   "mimeType": false,
   "title": false
   },
"types": {
   "date": "DATE",
   "datecreated": "DATE",
   "filesize": "LONG",
   "lastmodified": "DATE"
   },
"unique_key": "url",
"verify_schema": true
},
...
```

The LucidWorks Admin UI includes a page for each data source to edit field mapping for that data source which is where you can define, for example, that "content" should be mapped to "body", or that you allow only a single value for "title", etc.

In particular, you can define what is the name of the "uniqueKey" field in the incoming documents. If Nutch produces documents that use "url" as their unique identifier, then you would specify `"uniqueKey":"url"`. If "verify_schema" is set to "true" then LucidWorks will automatically define a mapping from "url" to whatever the current "uniqueKey" field is in the Solr schema for the target collection.

Once the external data source is defined (or updated) LucidWorks sends the serialized field mapping to the FieldMappingUpdateProcessor, which is a part of the "lucid-update-chain". This update processor receives the field mapping definition, and stores it in memory under a specified data source id. This field mapping is then updated each time a user makes some modifications to the data source definition, either via the Admin UI or using the REST API.

From this point, whenever an update request is received from an external process and it goes through this update chain, the update processor looks for a Solr parameter "fm.ds", which indicates the data source ID. If this parameter is present, and matches an existing defined mapping, then the documents in the update request are put through the FieldMappingUpdateProcessor, which re-maps field names, adjusts field multiplicity and adds LucidWorks-specific field names and values (which, among others, help to manage documents using the LucidWorks Admin UI).

Putting it all together

Now that we know how the field mapping is configured and processed in LucidWorks we can make sure that Nutch SolrIndexer uses the correct parameters, so that the correct field mapping is applied in LucidWorks to documents arriving from Nutch. Let's say that our external data source in LucidWorks has a data source id "4", we want to add the documents to "collection1" and our LucidWorks instance is running on a host "lucidworks.io:8888". Then the command-line parameters to SolrIndexer would look like this:

```
nutch solrindex http://lucidworks.io:8888/solr/collection1 db -linkdb linkdb -params
'update.chain=lucid-update-chain&fm.ds=4' segment1 segment2 [...]
```

As you can see, we are using the target collection's URL, and we specify "fm.ds=4" parameter that determines what field mapping needs to be applied to the incoming documents. Just in case, we explicitly set the update chain in case "lucid-update-chain" is not the default one (which it is in an out-of-the-box installation of LucidWorks). Please note that the `-params` option uses a URL-like syntax for passing Solr parameters, and since ampersand is usually a special shell character we had to enclose the `-params` string in single quotes to prevent the shell from interpreting it.

Summary

Nutch and LucidWorks form a powerful combination. Nutch is a robust crawling platform that can easily crawl thousands of pages per second while LucidWorks offers a scalable and robust indexing and search platform.

The way to use the two together is simply to:

- Define an "external" data source in LucidWorks, and adjust its field mapping to properly map the default Nutch field names to the ones that make sense in the current LucidWorks schema (e.g., "uniqueKey":"url", "content":"body", etc.). An external data source can be created by choosing the "External" type in the Sources page of the Admin UI or with the Data Sources API, specifying "lucid.external" for the `crawler` and "external" for the `type`.

- Start the Nutch SolrIndexer job with the additional -params option that specifies the data source id of the "external" data source defined in LucidWorks.

Related Topics

- Suggestions for External Data Sources
- Apach Nutch homepage

Integrating Google Connectors

LucidWorks ships with a connector for SharePoint repositories that uses the Google Connection Manager (GCM), which has been integrated with LucidWorks Search. SharePoint is only one of several available connectors, however, and others can be integrated by following the process defined below. Other repositories that can be crawled using GCM connectors include Documentum, IBM FileNet, LDAP, and Lotus Notes.

The process to integrate a new connector is simpler than developing a crawler from scratch, but requires understanding many of the same concepts. Before proceeding, please review the Introduction to Lucid Connector Framework. With this procedure we'll create a DataSourceSpec and a DataSource and then register it with LucidWorks. Once created, the new connector will be a type of data source for the `lucid.gcm` crawler (this would only be important if you use the Data Sources API - if you use the Admin UI only, you won't notice a difference between the other data source types).

It may also be helpful to review the Connector Developer's Guide from Google. The latest GCM version available is 2.8, although the Google documentation still refers to v2.4. LucidWorks is using Google Connector Manager version r3051 from the 2.8.x branch.

Covered in this section:

Overview of the Development Process

There are several steps described below, summarized here:

1. Prepare the development environment by checking the requirements.
2. Download and install LucidWorks. Do not start it during installation, or stop it if it is already running.

3. Download compatible Google Connector. Links to available connectors are available at https://code.google.com/p/google-enterprise-connector-manager/. Extract the files and copy the `lib` directory to `LWE_HOME/app/webapps/lwe-core/connector-manager/WEB-INF/lib/`. If there are duplicate .jars in the target directory, you may need to resolve any conflicts.
4. Start LucidWorks and verify there are no errors in the logs (see System Directories and Logs for more information on logs - in this case, pay attention to the `core-<date>.log`).
5. Extract the sample project that will be used during development of the DataSource and DataSourceSpec.
6. Use the sample project to extract the connector configuration form and use that information to create an appropriate DataSourceSpec.
7. Implement a DataSourceSpec class.
8. Implement an extension class that registers the DataSourceSpec
9. Write a configuration file for the java service loader.
10. Deploy the `.jar` file.
11. Start LucidWorks.

Preparing for development

The included sample project requires the following on the system where development will be completed:

- Ant 1.8+ is installed
- Java 1.6+ is installed
- LucidWorks is installed

To properly prepare the development environment, perform these steps:

1. Download and install LucidWorks. More information on installing LucidWorks is available in the section on Installation. If it is running after installation, stop it.
2. Download a compatible Google connector. Links to available connectors are available at https://code.google.com/p/google-enterprise-connector-manager/. Review the release notes for the candidate connectors to make sure they work with GCM v2.8.
3. Extract the connector files from the downloaded Connector archive and copy the `lib` directory to `LWE_HOME/app/webapps/lwe-core/connector-manager/WEB-INF/lib/`. If there are duplicate .jars in the target directory, you may need to resolve any conflicts.
4. Start LucidWorks with following command:
   `app/bin/start.sh -lwe_connectors_java_opts "-DlucidworksGCMPort=10000"`
5. Download and extract the sample project. The package contains an ant buildable project that can be used as a starting point when implementing new integrations.
6. Localize the sample project.
   1. Open `build.xml` and edit the `lwe-install-dir` property to point to the root directory of the LucidWorks installation.
   2. The `gcm-url` property must be changed to the correct port that was specifies when starting LUcidWorks.
7. Run `ant dist` to verify that the environment is succesfully configured. If the build does not succeed, correct any errors in `build.xml`.

Here is the beginning of a `build.xml` file that has been localized for development:

```
<project name="lwe-google-connector" default="dist" basedir=".">
        <description>
    simple example for adding new google connector to LWE
  </description>
        <!-- set global properties for this build -->
        <property name="lwe-install-dir"
location="/Users/cassandra4work/LucidWorks/LucidWorksSearch"/>
        <property name="lwe-lib-dir"
location="${lwe-install-dir}/app/webapps/lwe-core/lwe-core/WEB-INF/lib/" />
        <property name="connectors-lib-dir"
location="${lwe-install-dir}/app/webapps/connectors/lib/" />
        <property name="gcmcc-lib"
location="${lwe-install-dir}/app/crawlers/gcm-crawler.jar" />
        <property name="src" location="src/main/java" />
        <property name="resources" location="src/main/resources" />
        <property name="build" location="build" />
        <property name="dist" location="dist" />
        <property name="name" value="gcm-ldap-glue" />
        <property name="gcm-url" value="http://127.0.0.1:10000" />
...
```

Writing the DataSourceSpec

When integrating Google connectors into LucidWorks, the majority of the work is to make sure the new data source type can be used with Admin UI. By itself, a Google Connector provides a UI through a HTML form. In LucidWorks, however, the way to provide the configuration UI is based on DataSourceSpec-s. A DataSourceSpec is a class that provides sufficient information for the UI so that it can render the configuration screen (data source configuration screens in LucidWorks are dynamically generated based on information provided to it by the data source). Sometimes making the Admin UI work with the new connector is a straightforward task, but sometimes it requires some additional effort.

The configuration screen should provide the user with information about which fields are required and if possible, provide some guidance about what format to enter information. Unfortunately, at this point the process of figuring out what fields a Google Connector requires a little bit of reverse engineering (figure aout required parameters and their format from the HTML form) and some trial and error.

There are several steps to writing the DataSourceSpec for the connector, explained in each of the sections below:

1. Extracting Fields from the Connector HTML Form
2. Writing the DataSourceSpec

Extract Fields from the Connector HTML Form

Once the environment is properly configured and the sample project is set up for development, you can use the ant target `ant list-connectors` to show the GCM connectors installed in LucidWorks. You'll need the name of the connector in order to retrieve the HTML form. The output of this command looks like this (the WARN messages are fine):

```
Buildfile: /Users/cassandra4work/Downloads/google-connector-example/build.xml

init:

compile:
    [javac] /Users/cassandra4work/Downloads/google-connector-example/build.xml:33:
warning: 'includeantruntime' was not set,
            defaulting to build.sysclasspath=last; set to false for repeatable builds

list-connectors:
     [java] Using url:http://127.0.0.1:8888/connector-manager/
     [java] log4j:WARN No appenders could be found for logger
(org.apache.commons.httpclient.params.DefaultHttpParams).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for
more info.
     [java] Available connectors:
     [java]
     [java] - EMC_Documentum_Content_Server
     [java] - LDAPConnectorType
     [java] - sharepoint-connector

BUILD SUCCESSFUL
Total time: 3 seconds
```

The last few lines show the GCM connectors installed on the system.

Next, retrieve the connector configuration HTML form by executing `ant get-connector-form -Dconnector=<name of the connector>` (for example, `ant get-connector-form -Dconnector=LDAPConnectorType`). The output of this command is a file called `connector-form.html`, which is found in the example project main directory. You will need to examine this file for the required fields and their valid formats. Then you can use that information to write the DataSourceSpec for the connector.

For example, the HTML form for LDAP connector looks like this:

```
<tr><td><br/><b>LDAP Connector Configuration<span
style="color:#FF0000"><sup>Preview</sup></span></b></td></tr>
<tr> <td valign="top"><label for="hostname">LDAP Directory Server Host</label></td>
   <td><input name="hostname" id="hostname" type="text" ></input></td></tr>
<tr> <td valign="top"><label for="port">Port number</label></td>
   <td><input name="port" id="port" type="text"  value="389"></input></td></tr>
<tr><td valign="top"><label for="authtype">Authentication Type</label></td>
   <td><select name="authtype" id="authtype">
   <option value="ANONYMOUS" selected="selected">Anonymous</option>
   <option value="SIMPLE">Simple</option>
   </select></td></tr>
<tr> <td valign="top"><label for="username">LDAP Binding Distinguished Name
(DN)</label></td>
   <td><input name="username" id="username" type="text" ></input></td></tr>
<tr> <td valign="top"><label for="password">LDAP Binding Password</label></td>
   <td><input name="password" id="password" type="password" ></input></td></tr>
<tr><td valign="top"><label for="method">Connection Method</label></td>
   <td><select name="method" id="method">
   <option value="STANDARD" selected="selected">Standard</option>
   <option value="SSL">SSL</option>
   </select></td></tr>
<tr> <td valign="top"><label for="basedn">LDAP Search Base</label></td>
   <td><input name="basedn" id="basedn" type="text" ></input></td></tr>
<tr> <td valign="top"><label for="filter">User Search Filter (only these users will be
indexed)</label></td>
   <td><input name="filter" id="filter" type="text" ></input></td></tr>

<tr style='display: none'><td>
<input type = 'hidden' id = 'schemavalue' name = 'schemavalue' value = '[]' /><script
type="text/javascript">
function getIndexOf(arr, value) {for(var i=0; i< arr.length; i++){if(arr[i] ==
value)return i;}return -1;}var schemaList = new Array();
function appendToSchema(chkbox) {if (schemaList.length == 0) {schemaList =
JSON.parse(document.getElementById('schemavalue').value);}
   if (chkbox.checked)
{schemaList.push(chkbox.value);document.getElementById('schemavalue').value =
JSON.stringify(schemaList);}
   else {if(getIndexOf(schemaList,chkbox.value) >= 0)

{schemaList.splice(getIndexOf(schemaList,chkbox.value),1);document.getElementById('schemav
= JSON.stringify(schemaList);}}}
</script>
</td></tr>
```

From that it is possible to see that there are 8 parameters: `hostname`, `port`, `authtype`, `username`, `password`, `based`, `filter` and `schemavalue` statically defined in the HTML form. Closer inspection shows that the `schemavalue` field is dynamically build with javascript (it is actually based on the content of the LDAP server) and the format for that field is `["<field1>", "<field2>"]`.

Writing the DataSourceSpec

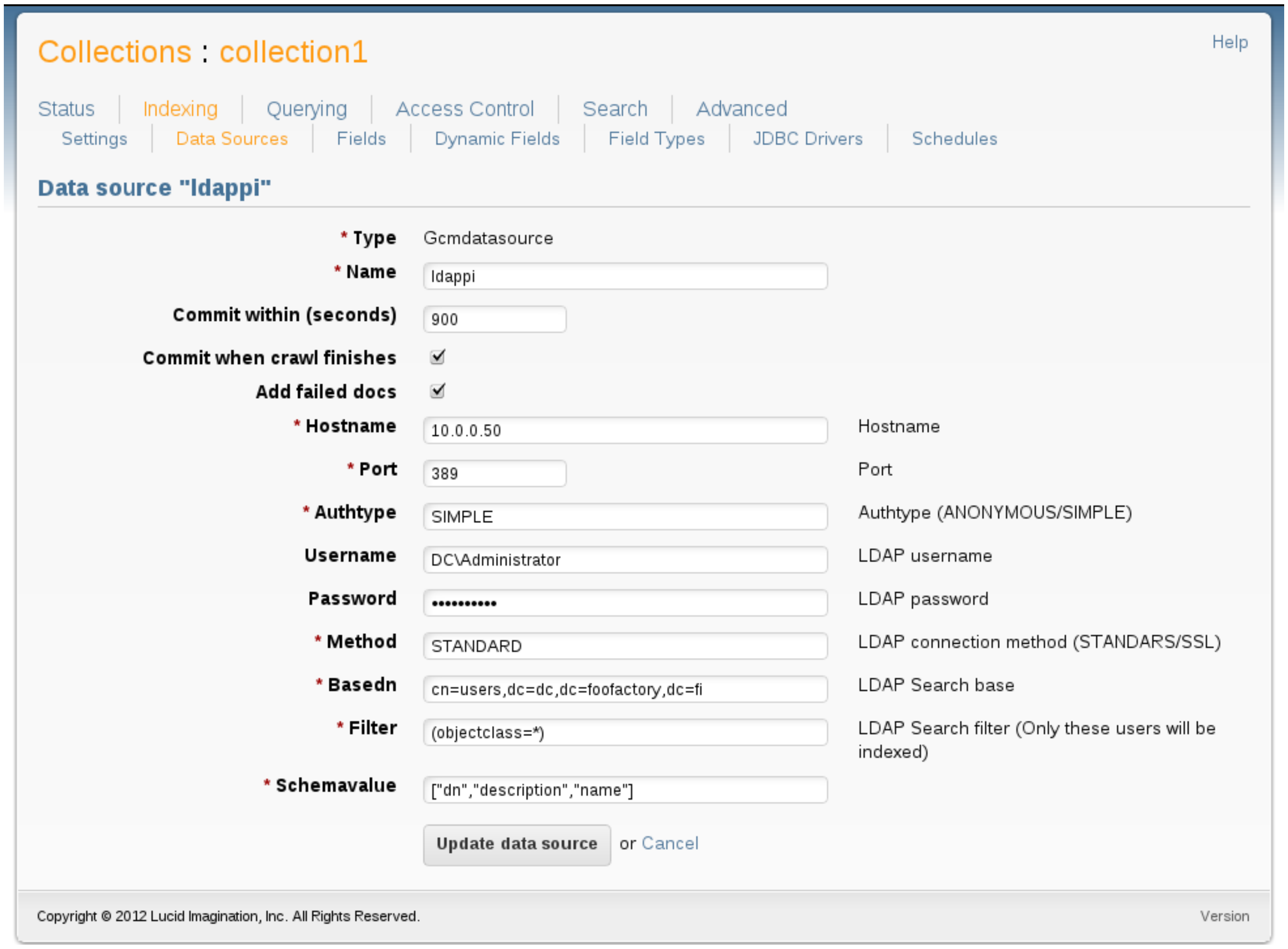
The next step is to write a `DataSourceSpec` definition based on this information. Specs provide information for the crawler to know which attributes are required, how the attributes should be defined, and how the Admin UI should create a form to allow GUI configuration of the data source.

For all Google Connector-based data sources there is a base class called `GCMSpec` that can be extended to the needs of the new Google Connector.

From the example project the class implementing the UI spec for the LDAP connector is called `GCMLDAPSpec` and it looks like this:

```
package com.lucid.gcm.connector;

import com.lucid.crawl.datasource.FieldMapping;
import com.lucid.crawl.gcm.GCMSpec;
import com.lucid.crawl.gcm.LWEGCMAdaptor;
import com.lucid.spec.SpecProperty;
import com.lucid.spec.Validator;

public class GCMLDAPSpec extends GCMSpec {

  public static final String NAME = GCMLDAPSpec.class.getSimpleName().toLowerCase();

  public static String HOSTNAME = "hostname";
  public static String PORT = "port";
  public static String AUTHTYPE = "authtype";
  public static String USERNAME = "username";
  public static String PASSWORD = "password";
  public static String METHOD = "method";
  public static String BASEDN = "basedn";
  public static String FILTER = "filter";
  public static String SCHEMAVALUE = "schemavalue";

  public GCMLDAPSpec(LWEGCMAdaptor adaptor) {
    super(adaptor);
  }

  @Override
  protected void addCrawlerSupportedProperties() {
    super.addCrawlerSupportedProperties();
    addSpecProperty(new SpecProperty(HOSTNAME, "Hostname",
        String.class, "", Validator.NOT_BLANK_VALIDATOR, true));
    addSpecProperty(new SpecProperty(PORT, "Port",
        Integer.class, "", Validator.NOT_BLANK_VALIDATOR, true));
    addSpecProperty(new SpecProperty(AUTHTYPE,
        "Authtype (ANONYMOUS/SIMPLE)", String.class, "",
        Validator.NOT_BLANK_VALIDATOR, true));
    addSpecProperty(new SpecProperty(USERNAME,
        "LDAP username", String.class, "", Validator.NOT_NULL_VALIDATOR, false));
    addSpecProperty(new SpecProperty(PASSWORD,
        "LDAP password", String.class, "", Validator.NOT_NULL_VALIDATOR, false));
    addSpecProperty(new SpecProperty(METHOD,
        "LDAP connection method (STANDARS/SSL)", String.class, "",
        Validator.NOT_BLANK_VALIDATOR, true));
    addSpecProperty(new SpecProperty(BASEDN,
        "LDAP Search base", String.class, "", Validator.NOT_BLANK_VALIDATOR,
        true));
    addSpecProperty(new SpecProperty(FILTER,
        "LDAP Search filter (Only these entries will be indexed)", String.class,
        "", Validator.NOT_BLANK_VALIDATOR, true));
    addSpecProperty(new SpecProperty(SCHEMAVALUE, "[ ]",
        String.class, "", Validator.NOT_BLANK_VALIDATOR, true));
  }


  @Override
  public FieldMapping getDefaultFieldMapping() {
    FieldMapping fieldMap = super.getDefaultFieldMapping();
    //define the default field mapping for the datasource
    //fieldMap.defineMapping("GCM_xxx", "xxx");
    return fieldMap;
  }

}
```

When deployed to LucidWorks, the data source will look like this in the Admin UI:


Collections : collection1
Help
Status | Indexing | Querying | Access Control | Search | Advanced
Settings | Data Sources | Fields | Dynamic Fields | Field Types | JDBC Drivers | Schedules

Data source "ldappi"

* Type	Gcmdatasource	
* Name	ldappi	
Commit within (seconds)	900	
Commit when crawl finishes	✓	
Add failed docs	✓	
* Hostname	10.0.0.50	Hostname
* Port	389	Port
* Authtype	SIMPLE	Authtype (ANONYMOUS/SIMPLE)
Username	DC\Administrator	LDAP username
Password	••••••••••	LDAP password
* Method	STANDARD	LDAP connection method (STANDARS/SSL)
* Basedn	cn=users,dc=dc,dc=foofactory,dc=fi	LDAP Search base
* Filter	(objectclass=*)	LDAP Search filter (Only these users will be indexed)
* Schemavalue	["dn","description","name"]	

Update data source or Cancel
Copyright © 2012 Lucid Imagination, Inc. All Rights Reserved.
Version

Registering the Extension

Finally the implemented classes needs to be registered into LWE so that the new Google Connector is recongnized in LWE. This is done with a specific class that implements `GCMExtension`:

```
package com.lucid.gcm.connector;

import java.util.HashMap;
import java.util.Map;

import com.lucid.crawl.gcm.GCMExtension;
import com.lucid.crawl.gcm.GCMSpec;
import com.lucid.crawl.gcm.LWEGCMAdaptor;

public class GCMLDAPExtension extends LWEGCMAdaptor implements GCMExtension {

 @Override
  protected void customizeProperties(Map<String, Object> dsProperties,
      HashMap<String, String> gcmProperties) {
   gcmProperties.put(GCMSpec.CONNECTOR_TYPE, "LDAPConnectorType");
  }

  @Override
  public void register(
      Map<String,com.lucid.crawl.datasource.DataSourceSpec> types) {
    LWEGCMAdaptor.register("ldap", this);
    types.put("ldap", new GCMLDAPSpec(this));

  }
}
```

There are two methods in this class. The first is `register`, which registers the new connector. In the other method, `customizeProperties` allows modifications to the parameters sent from the Admin UI so they match what GCM expects (this may just be desirable, or it may be mandatory).

Some examples of transformations that might be needed are:

- Set values that are required by the connector or GCM but not exposed in the Admin UI (for example, the `CONNECTOR_TYPE`)
- Build (sometimes cryptic) strings that are normally built with javascript by the connector HTML form from static set of fields specified in the Spec

One final step is required, because the extensions are implemented by using the Java Service Loader. A file in the example project called `src/main/resources/META-INF/services/com.lucid.crawl.gcm.GCMExtension` lists the available GCMExtensions. You need to add the name of the class implementing `GCMExtension` in that file.

Deployment

During the development phase the ant target "ant deploy" can be used. When everything is working as expected the jar for the integration glue can be created by running `ant dist` that will create a `.jar` file in the `dist` directory. This jar file can then be added inside `gcm-crawler.jar` in directory `$LWE_INSTALL_DIR/app/crawlers/` (the deploy target does this automatically).

Once the connector has been deployed, restart LucidWorks and the new connector should be available to use via the Admin UI or with the Data Sources API.

During development it might be necessary to cleanup the GCM configuration and state in `$LWE_INSTALL_DIR/conf/gcm/connectors/<connector-name>` by running "rm -rf" inside that directory.

Processing Documents in Batches

By default, LucidWorks Search will crawl as much content as it can (within limits set on the data source), parse the documents to extract fields, and finally index the documents in one seamless step. However, there may be times when you would like to do some processing on the documents before indexing them, perhaps to add metadata or to modify data in specific fields. In that case, it is possible to only crawl the content and save it in a batch for later parsing and/or indexing. This is called Batch Processing and allows you to separate the fetching data from the process of parsing the rich formats (such as PDFs, Microsoft Office documents, and so on), as well as the process of indexing the parsed content in Solr.

How a Batch is Constructed

Batches consist of the following two parts:

- a container with raw documents, and the protocol-level metadata per document
- a container with parsed documents, ready to be indexed.

The exact format of this storage is specific to a crawler controller implementation. Currently a simple file-based store is used, with a binary format for the raw content part and a JSON format for the parsed documents. The first container is created during the fetching phase, and the second container is created during the parsing phase. A new round of fetching creates a new batch if one or more of the parameters described above requires it.

Steps to Configure Batch Crawling

It's not possible to configure Batch Crawling with the LucidWorks Search Admin UI. To work with batches and batch jobs, use the Batch Operations API. The basic workflow is as follows:

1. Create a data source using the Admin UI or Data Sources API. Don't start crawling yet.
2. Configure the data source to be saved as a batch by setting the `indexing` parameter to `false` using the Data Sources API. You can also set the `caching` and `indexing` parameters as described below.
3. Start the crawl and let it finish.
4. Get the `batch_id` for the data source using the Batch Operations API call: `GET http://localhost:8888/api/collections/collection1/batches`.
5. Using the Batch Operations API, start the batch job for your data source using the `batch_id` obtained in the previous step: `PUT http://localhost:8888/api/collections/collection1/batches/crawler/job/batch_id`.

More about the Data Source Settings

To instruct LucidWorks Search not to parse or index the crawled documents, set the `indexing` parameter of a data source to `false` using the Data Sources API. You can also set the `parsing` and `caching` parameters to true or false, depending on your needs. Batch crawling attributes for data sources are as follows:

Key	Type	Default	Description
parsing	boolean	true	If true, the raw content fetched from remote repositories is immediately parsed in order to extract the plain text and metadata. If false, the content is not parsed: it is stored in a new batch with its protocol-level metadata. New batches are created during each crawl run as needed.
caching	boolean	false	If true, the raw content is stored in a batch even if immediate parsing and/or indexing is requested. You can use this to preserve the intermediate data in case of crawling or indexing failure, or in cases where full re-indexing is needed and you would like to avoid fetching the raw content again.
indexing	boolean	true	If true, the parsed content is sent to Solr for indexing. If false, the parsed document is not indexed: it is stored in a batch (either a newly created one, or the one where the corresponding raw content was stored). Set this attribute to `false` to enable batch crawling.

When you configure a data source to process documents as a batch, information about crawl attempts will display in the Admin UI for that data source (even though you cannot configure the batch parameters via the UI). So, you can use the Data Sources API to enabled `caching` and/or disable `indexing`, and initiate the crawl through the Admin UI. The UI will show the number of documents found, updated, deleted, etc.

Not all crawler controllers support all batch processing operations. For example, the Aperture crawler (`lucid.aperture`) does not support raw content storage: it behaves as if the "parsing" parameter is always `true` and caching is always `false`.

You can also use the Batch Operations to get the status of or stop running batch jobs, delete batches and batch jobs, and so on.

Related Topics

- Batch Operations
- Data Sources

Query and Search Configuration

Once your content is in the index, you and your users will want to query the index to find the documents they need. This section covers the options and settings to optimize the search experience for users.

First, there's an overview of how searching works in the section Managing Queries.

A few features make it easy for users to find documents: Enterprise Alerts allow them to get email notifications when new documents are added to the index; Spell Check corrects errors in terms they've entered; Auto-Complete of User Queries makes suggestions for valid terms while they type, and Synonyms and Stop Words allows use of similar terms and very common words to improve the search experience.

While LucidWorks Search includes a Search UI, it's meant to be used during development and not for a production application. The section Getting Search Results describes in detail how to query the LucidWorks Search index, and what responses look like, for use while designing your own search application customized for your needs.

You may have need to improve the results your users see. The Click Scoring Relevance Framework provides a way to boost documents that other users have already clicked on for the same query, with the theory that if other users found it useful, you might too.

If you have serious business needs for including very specific rules in response to certain queries (or all queries), the section Business Rules Integration describes how to plug in those rules with LucidWorks Search.

Managing Queries

Understanding Query Processing

The goal for any search application is to return the correct document while allowing a user to enter a query however they want. The query may be in the form of keywords, a natural language question, or snippets of documents. Advanced queries may be (or may include) date ranges, Boolean operations, searches on specific document fields, or proximity information to define how close (or how far apart) terms should be to each other.

Search engines take the query entered and transform them to find the best results. Synonyms of the terms entered may be applied to expand the number of possible document matches (such as looking for "attorney" when a user enters "lawyer"). If terms are stripped of punctuation and capital letters during indexing, a similar process should also be applied to the user query to ensure matches in the index.

Relevance Ranking

The system then tries to match the user's transformed terms to terms in documents in the index. Once it finds documents, it puts the list of matching documents into some order. They might be ordered by date, by entry to the index, or, most commonly, by relevance. Relevance ranking is one of the most complex components of a search engine, since most queries are very short (one to three words) and that is usually not enough information to know the user's full intention. However, some general methods to return the best results are common across many good search engines.

When matches for a user's query are found, the number of times each term (word, date, and so on) occurs in the document is calculated. Documents with a higher frequency of a term are considered more likely to be relevant than documents with a lower frequency. This "term frequency" is combined with another method called "inverse document frequency", which means that words appearing in the fewest documents tend to be the more valuable terms to use in calculating relevance. In other words, the more rare words (or dates, numbers, etc.) are in your query, the more important they are for finding the best document matches.

For example, in a search for "the new bookshelf", simply adding up the term frequencies to get the total of how many times they occur would not get the best document, because each word is not of equal information value. Documents with lots of occurrences of "bookshelf", the rarest of the three words, would likely be more relevant than documents with lots of occurrences of either "the" or "new", which are very common words and likely occur in many documents that have nothing to do with bookshelves.

Some search engines will strip very small and common words (a, the, of, and so on), known as stop words, from the user's query. This is less common than it used to be, when disk space was very expensive and it was critical for indexes to be as small as possible: removing these words could reduce an index as much as 50%. Since these words are so common they generally do not add much to most user's searches, but in some cases they could be critical to finding the right document (a movie search for "To Have and Have Not", for example, or a company called "On Technology") so removing them was limiting for users. Today, with less expensive disks available and better compression techniques they are nearly always included in the index and used when processing a user's query. Special care should be taken to ensure the high frequency of these small words does not distort query results, which the Lucid query parser was designed to do.

Other techniques used in relevance ranking include considering the date of the item (documents that are more recent may be considered more relevant to some users) or where the term matches occur (words in the title of the document may be more relevant than words at the end). LucidWorks includes the option to use Click Scoring, which uses information about what documents prior users have clicked when calculating relevance as another method that can be employed.

One factor that can improve the ranking of results is to provide user's with tools to expand their queries without providing additional search terms. A "find similar" option allows users to request documents that are similar to one that may be almost right for their search. Explicit or automatic feedback allows users to resubmit their search with terms pulled from documents that are considered near matches, in hopes of getting more or better matches. In LucidWorks, unsupervised feedback can be enabled, which automatically takes the top documents from the preceding results and pulls important terms from them to use with the user's original query.

Search Results

Once the system has compiled the list of matching documents, they need to be presented to the user with enough information to help them decide which documents are best. First, the documents should be sorted in some way: the most common is by how well the documents match the query (relevance), but date may also be preferred, or another field such as author or manufacturer. Some snippet of the document should be used to help users figure out if the document is a match, such as title, author and date. The first few sentences, or a few sentences around the highlighted occurrence of the user's search term, are also helpful to give the user some context for why each document was selected as a match.

Document clustering, also called faceting, can help users select from a large list of results. Facets are documents grouped together by some common element such as author, type, or subject and are usually displayed with the number of results that can be found in each group. Providing facets allows users to "drill down" or further restrict their results and find the documents they are looking for.

Result lists may need to be limited to only documents that a user has access to view. Lucid Works Enterprise has several options for doing this.

Enterprise Alerts

The alerts feature of LucidWorks Search allows a user to save a search and receive notifications when new results are available.

A *passive alert* acts like a smart saved search. It is smart in the sense that it keeps track of the last time the user checked for new results to their search and provides only new results the next time the alert is checked. As the name implies, a passive alert provides no notification when new documents are indexed. It waits for a request before it checks for new query results.

An *active alert* is checked periodically at a user-defined interval (currently every hour, day or week is available). When new results to the query are discovered, an active alert sends a notification via email to the email address defined in the alert. At the current time, only email notifications are possible.

How Alerts Work

1. The user does a search, and clicks the link under the search box to "Create new alert".
    1. The user configures the alert and notification settings, including how often to run the alert (`period`) and an email address to send alert notifications.
    2. LucidWorks Search automatically saves the timestamp of when the alert was created (`checked_at`).
2. Every 60 seconds, a scheduled process within the UI checks to see if it is time to run any alerts.
3. When the alert is run, the query is executed as entered by the user, on the collection that the query was initially run on, and the timestamp of the most recent document is compared to the timestamps of documents in the result set.

4. If there are new results for the user, a notification is sent, assuming the mail server has been configured in the Settings page of the UI.

Parameter names in parentheses above refer to the attributes used with the Alerts API. Alerts can be set up with the default Search UI, but while designing your own search application, you will likely need to use the Alerts API to integrate the functionality.

Enabling Alerts

In order for alert notifications to work with LucidWorks Search, the email server must be configured via the System Settings page in the Admin UI.

In addition, the LucidWorks Search schema must define a `timestamp` date field. Both active and passive alerts require that the index define a `timestamp` date field that is indexed, defaulted to NOW, and used to indicate the time of document indexing. By default, LucidWorks Search schema already defines this field. However, if modifying the LucidWorks Search default field set (the "schema"), you must retain this field for alerts to work properly.

Spell Check

Spell check, also known as "Did You Mean?", is the ability of the search application to make alternate suggestions for queries based on words that are similar to the terms entered by the user.

Integrated query spell checking is bundled with LucidWorks Search, with the option to integrate third-party enhanced spell checking capabilities. It is index-driven, meaning all suggestions are derived from the actual content in an indexed collection and not from a predefined dictionary of words. In practical terms, this helps solve problems with messy data written by a variety of authors of varying quality where one author may spell a word one way, while another author spells it a different way and the user spells it a third way. An index-derived spell checker provides suggestions based on the (sometimes incorrect) words in the dictionary, ensuring that end users still find relevant documents even if they contain misspellings.

To enable spell checking for specific fields, three steps must be taken:

1. Enable spell checking by accessing the Querying - Settings tab of the Admin UI and check the box next to "Spell-check". Alternatively, the Settings API can be used.
2. Ensure there are fields configured for spell checking by accessing the Indexing - Fields tab and choosing "Index for Spell Checking". The Fields API could also be used to modify field settings. Be sure to select fields that contain ample text-based content that end users are going to search against using word-based queries. For example, the title and body fields are good candidates, while a "price" field likely isn't.
3. Crawl your content.
4. Perform queries.

Spell Check Settings are Per Collection

The indexes created for spell checking are unique to each collection, and based on the documents indexed for a particular collection. In a multi-collection environment, the steps to enable spell checking must be done in each collection.

When indexing content, LucidWorks will automatically create an index of terms to be used for term suggestions. By default, LucidWorks will create this index from content in the `author`, `body`, `description`, and `title` fields.

In prior versions of LucidWorks, a separate task needed to be scheduled to build the spell check index of terms. Starting with v2.0 of LucidWorks Search, that requirement has been removed and the `spell` index will be created automatically during regular indexing.

Auto-Complete of User Queries

Query auto-complete shows users suggestions for their queries as they type the words. In LucidWorks Search, this is an index-driven feature, meaning all suggestions are derived from the actual content in an indexed collection and not from a predefined dictionary of words. For users, this means they will see suggestions for actual terms in documents, not for terms that may not exist in the content.

Auto-Complete Settings are Per Collection

The indexes created for auto-complete are unique to each collection, and based on the documents indexed for a particular collection. In a multi-collection environment, the steps to enable auto-complete must be done in each collection.

To enable auto-complete of user queries, three steps must be taken:

1. Enable auto-complete by accessing the Query Settings screen of the Admin UI and check the box next to "Auto complete". Alternatively, the Settings API can be used.
2. Ensure there are fields configured for auto-complete by accessing the Indexing Fields screen and choosing "Index for autocomplete". The Fields API can be used instead if you prefer. A good auto-complete field is a field that contains ample text-based content that end users are going to search against using word-based queries. For example, the title and body fields are good candidates, while a "price" field probably isn't.
3. After crawling some content, create the "autocomplete" index by accessing the Index Settings page and scheduling a time for the "Generate autocomplete index" job to run. The Activities API can be used instead if preferred. This **must** be done before automatic query completion will occur for users.

LucidWorks Search does not create the auto-complete index by default. Auto-Complete indexing jobs must be scheduled using the Indexing - Settings tab of the Admin UI (or via the Activities API) before query suggestions will appear for users.

If you enable auto-complete but don't see any suggestions, you may want to modify the `threshold` parameter, which defines the minimum fraction of documents a term should appear in before being added to the `autocomplete` index. The default is "0.05", and a lower number will include more terms in the index.

The only way to modify this parameter is to edit `solrconfig.xml` for each collection (in `$LWE_HOME/conf/solr/cores/collection/conf`). In the section:

```
<searchComponent class="solr.SpellCheckComponent" name="autocomplete">
```

Find the parameter `<float name="threshold">.005</float>` and change it to the desired value. After saving `solrconfig.xml`, restart LucidWorks.

Automatic Creation of Auto-Complete Indexes

This functionality is **not available** with LucidWorks Search on AWS or Azure

By default, LucidWorks does not build the indexes for auto-complete each time documents are added to the index because doing so may have performance implications in a production environment with a large index. However, LucidWorks can be configured to do this automatically by changing the `buildOnCommit` setting in `solrconfig.xml` to `true`. Usually, it's a better idea to schedule index builds so that they run on a regular interval rather than doing it on every commit using this method.

If, however, you would like this to happen automatically, find the following section in the `solrconfig.xml` file for each collection:

```
<!-- Auto-Complete component -->
 <searchComponent name="autocomplete" class="solr.SpellCheckComponent">
  <lst name="spellchecker">
   <str name="name">autocomplete</str>
   <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
   <str name="lookupImpl">org.apache.solr.spelling.suggest.tst.TSTLookup</str>
   <str name="field">autocomplete</str>
   <str name="storeDir">autocomplete</str>
   <str name="buildOnCommit">false</str>
   <float name="threshold">.005</float>
   <!-- <str name="sourceLocation">american-english</str> -->
  </lst>
 </searchComponent>
```

In the section, `str name="buildOnCommit">false</str>`, change "false" to "true", and save the file. Restart LucidWorks for the changes to take effect. Repeat this for each collection that should build the auto-complete index each time documents are added to the index.

Synonyms and Stop Words

Synonyms are words that are similar in meaning to each other, such as "hat" and "cap". In the context of a search application, they are another tool for improving results for users because they provide the opportunity to substitute words and expand the terms matched in the index.

Stop Words, on the other hand, are used to restrict the results of a search, by removing very small and very common words (such as "the" and "and") that often have little bearing on whether a document is a good match or not.

Synonym Expansion

LucidWorks Search manages synonyms with the use of a `synonyms.txt` file found in the `$LWE_HOME/conf/solr/cores/collection/conf` directory (unique for each collection). Synonyms can be edited in that file, via the Admin UI, or with the Settings API.

Synonyms can be either single terms or multi-term phrases. There are two ways to express synonyms:

- A comma-separated list of words (i.e., "lawyer, attorney" or "i-pod, i pod, ipod"). When the term entered by the user matches a term in the list, all terms are substituted for the term the user entered, including the matching term. If "lawyer, attorney" appears in the synonym list, when the user enters "lawyer", the system will search for documents that include both "lawyer" and "attorney".
- A mapping of one or more terms to another (i.e., "i-pod => ipod"). When entered as a mapping, the terms on the left of the "=>" symbol will be replaced by the terms on the right side of the symbol, which means that the user's query may not appear in the documents returned for the query. If "i-pod => ipod" appears in the synonym list, when the user enters "i-pod", the system will search for documents that contain the term "ipod" only.

There can be an unlimited number of terms and phrases which are defined as synonyms. However, it's usually not a good idea to add an entire thesaurus as a synonym file because not all terms are necessarily interchangeable (in some contexts, yes, but not always). For example, a doctor looking "myocardial infarction" is likely looking for documents that use the clinical term for the condition (and are thus more advanced) instead of documents written for a layman which likely uses the phrase "heart attack".

When considering synonyms, you should also consider which fields should be used for synonym expansion. In LucidWorks Search, the `body`, `description`, `title` and `text_all` fields are used for synonym expansion by default, meaning that those are the fields that will be used for the expanded or modified query.

If creating a synonym file manually, make sure to format the file properly. Lines starting with pound (#) are comments. Explicit mappings are indicated with terms separated by "=>", where a comma-separated list of terms on the left side will be replaced with the list of terms on the right side. Equivalent synonyms may be separated with commas and will give no explicit mapping (that is, the listed terms are equivalent). This allows the same synonym file to be used in different synonym handling strategies. For example:

```
lawyer, attorney
one, 1
two, 2
three, 3
ten, 10
hundred, 100
thousand, 1000
tv, television

#multiple synonym mapping entries are merged.
foo => foo bar
foo => baz
#is equivalent to
foo => foo bar, baz
```

If familiar with Solr, the file is formatted the same as the Solr synonyms file.

Stop Words

LucidWorks Search stores stop words in a file called `stopwords.txt`, found in the `$LWE_HOME/conf/solr/cores/collection/conf` directory (unique for each collection). The stop words can be edited in that file, via the Admin UI, or with the Settings API.

The stop word file is just a list of terms, one per line.

Many common prepositions, pronouns, and adjectives offer little benefit for matching documents, but can add some value when ranking results. Although it is possible to remove stop words when documents are indexed, more relevant results will be achieved by indexing all terms, querying only non-stop words, and then boosting the results by including the stop words with non-stop words. There is the special case where a query consists only of stop words (such as the classic, "To be or not to be"). In that case, all words are included in the query.

All words within quoted phrases are used for the query, even if they are stop words. The user can also force a stop word to be included in the search by either preceding it with a plus sign ("+") or enclosing it within double quotation marks. For example,

User Input	Query Interpretation
`at a conference`	"at" and "a" are stop words, so they will not be included with the query
`+at a conference`	"at" will be included in the query, but "a" will not
`"at" a conference`	Same
`"at a conference"`	All three words will participate in the query
`this is it`	There is no need to override because all three words are stop words, so all three will be included in the query

If creating the stop words file manually, the format is one term per line, as in:

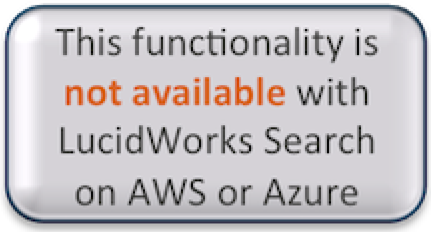
```
a
an
and
are
as
at
```

This is the same format as the Solr stopwords format.

Related Topics

- Suppressing Stop Word Indexing
- Settings API
- Synonyms in the Admin UI
- Stop words in the Admin UI

Suppressing Stop Word Indexing



This functionality is **not available** with LucidWorks Search on AWS or Azure

By default, LucidWorks Search indexes all stop words. Modern data storage is very cheap and even the simplest of stop words provide additional context that boosts relevancy and enables more precise queries. By default, the Lucid query parser eliminates stop words from basic queries, including them only when they are used in quoted phrases, or when the query term list consists only of stop words. In addition, the Lucid query parser uses query stop words to construct relevancy boosting phrase terms (bigram and trigram phrases) to supplement the basic query. Still, there may be applications and environments where the choice is to suppress the indexing of stop words.

TODO - update this for Field Types in the UI and API

Disabling Stop Word Indexing

Solr field types in the schema XML file control whether stop words will be indexed for particular fields. A stop word filter may be placed in the tokenizer chain for the index analyzer for a field type to filter out stop words and assure that they will not be stored in the index.

Filters are specified at the field *type* level, not the field level. For example, you may have `title` and `body` fields, both with the `text_en` field type. A stop word filter may be specified for the `text_en` field type and will apply to all fields of that same type, in this case `title` and `body`. If you really need to have a separate filter for a subset of the fields of a given type, you must create a separate field type to use for that subset of fields.

The standard stop word filter is named `StopFilter` and is generated by the `StopFilterFactory` Java class. LucidWorks ships with a schema XML file (`schema.xml`) with the `text_en` field type with a commented out entry for this standard stop word filter. To enable it, simply remove the XML comment markers around that one filter entry.

Schemas are Collection Specific

The `schema.xml` file is specific to each collection and can be found under `$LWE_HOME/conf/solr/cores/collection/conf`. If using multiple collections, be sure to locate the correct `schema.xml` file for the collection to be updated. After editing the schema.xml file, LucidWorks should be restarted. On some Windows machines, LucidWorks may need to be stopped before editing the file.

So, starting with the following in `schema.xml`:

```
<fieldType class="solr.TextField" name="text_en" positionIncrementGap="100">
 <analyzer type="index">
  <tokenizer class="solr.WhitespaceTokenizerFactory"/>
  <!-- in this example, we will only use synonyms at query time
  <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt"
          ignoreCase="true" expand="false"/>
  -->
  <!--
  <filter class="solr.StopFilterFactory" ignoreCase="true"
          words="stopwords.txt"/>
  -->
  <filter class="solr.WordDelimiterFilterFactory"
          generateNumberParts="1" generateWordParts="1"
          catenateAll="0" catenateNumbers="1" catenateWords="1"
          splitOnCaseChange="0"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ISOLatin1AccentFilterFactory"/>
  <filter class="com.lucid.analysis.LucidPluralStemFilterFactory"
          rules="LucidStemRules_en.txt"/>
 </analyzer>
 ...
```

Edit the stop filter factory entry that is commented out:

```
<!--
<filter class="solr.StopFilterFactory" ignoreCase="true"
        words="stopwords.txt"/>
-->
```

And remove the XML comment markers to get:

```
<filter class="solr.StopFilterFactory" ignoreCase="true"
        words="stopwords.txt"/>
```

Which results in the following analyzer description:

```
<fieldType class="solr.TextField" name="text_en" positionIncrementGap="100">
 <analyzer type="index">
  <tokenizer class="solr.WhitespaceTokenizerFactory"/>
  <!-- in this example, we will only use synonyms at query time
  <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt"
          ignoreCase="true" expand="false"/>
  -->
  <filter class="solr.StopFilterFactory" ignoreCase="true"
          words="stopwords.txt"/>
  <filter class="solr.WordDelimiterFilterFactory"
          generateNumberParts="1" generateWordParts="1"
          catenateAll="0" catenateNumbers="1" catenateWords="1"
          splitOnCaseChange="0"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ISOLatin1AccentFilterFactory"/>
  <filter class="com.lucid.analysis.LucidPluralStemFilterFactory"
          rules="LucidStemRules_en.txt"/>
 </analyzer>
 ...
```

After such a change, be sure to re-index all documents.

Also, make sure that the query analyzer for that field type references the same stop words file:

```
<analyzer type="query">
 <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
```

Do not change or comment out the query analyzer when making this index change.

This example only changes the `text_en` field type. If other field types are being used, or should be changed, find the section of the `schema.xml` for that field type and

Position Increment Mode

There are two modes for suppressing stop word indexing:

1. **Skip mode**: Completely ignore or skip them, as if they were not present. This is the default when no other option is selected. When skip mode is selected, the query parser will ignore or skip stop words in quoted phrases.
2. **Position increment mode**: Do not store them in the index, but increment the position counter so as to leave a blank at the position of each stop word. When position increment mode is selected, the query parser will also skip each stop word, but will increment the position of the next term in the phrase so as to allow any term to match between the previous term and the next term after the stop word. This will allow for more precise query matching than the first mode where stop words are simply discarded.

For example, given these documents:

- Doc #1: Buy the time for the test.
- Doc #2: Buy more time for the test.
- Doc #3: Buy time for test.

A query of `Buy the time` regardless of the stop word indexing mode will be equivalent to `Buy AND time` and match all three documents.

A query of `"buy the time"` in normal indexing mode will match exactly that phrase and match only the first document. In skip mode it is equivalent to `"buy time"` and will match the first and third documents. In position increment mode the query is equivalent to `"buy * time"` which is not a valid query format but indicates that `"time"` will match the second word after `"buy"` regardless of the intervening word. This will match the first and second documents, but not the third document.

To enable position increment mode, edit the `StopFilterFactory` entry of the index analyzer (which was un-commented above) in `schema.xml` to add `enablePositionIncrements="true"`. The section will appear as follows:

```
<filter class="solr.StopFilterFactory" ignoreCase="true"
        words="stopwords.txt" enablePositionIncrements="true"/>
```

Only the index analyzer should be changed. The query analyzer should not be changed regardless of the indexing mode. The query parser has internal logic that decides whether and when to call the query stop word filter.

After this change, be sure to re-index all documents.

Getting Search Results

LucidWorks Search includes a default search interface that is designed to be used during development to evaluate and test the performance of crawler and index configuration. At this time, there are no options to customize the default Search UI because we expect that you will prefer your own designs and options specifically tailored to your audience.

This page serves as an introduction to Solr Searching. You should also look at these sections:

- Constructing Solr Queries
- Solr Query Responses

What follows is some information about how to start working with search results in LucidWorks Search.

LucidWorks is built upon Solr and supports it natively. While LucidWorks includes a REST API for many administrative functions (like creating data sources, updating fields, etc.), there is no LucidWorks-specific API for search results. In order to get results from LucidWorks, you'll need to learn a little Solr syntax. To help you with this, you may find it helpful to review LucidWorks' free Apache Solr Reference Guide, particularly the section on Searching.

Basics of Searching

Search in LucidWorks is making a direct connection to Solr, which processes queries with a **request handler**. The request handler defines the logic to be used for processing the query. Solr supports several different request handlers, and LucidWorks includes a special Solr search request handler called `/lucid`. This is pre-selected as the default, but could be changed to another request handler by editing `solrconfig.xml` for the collection.

Request Handlers

Each request handler has several settings pre-configured, but these can be overridden for an individual query by the client application. In some cases, this may adversely affect the expected search results, so care should be taken when overriding some parameters.

To process a query, a request handler calls a **query parser**, which interprets the terms and parameters of a query. The query parser understands the terms the user entered (the actual words), any parameters entered for fine-tuning the query (such as instructions to search a specific field for the terms, to boost terms found in specific fields to rank them higher in results, and to interpret the syntax for advanced queries including ranges or boolean operators, etc.), and any parameters for controlling the presentation of the response (such as the order of results or the fields of a document to be returned). LucidWorks has created its own query parser that is used by default, but any other Solr query parser could also be used (the two most popular are DisMax and ExtendedDisMax).

The request handler also likely has defined many parameters for faceting, spell check, autocomplete, highlighting, security settings and so on. The `/lucid` request handler has enabled and defined each of those components by default; with other request handlers those may need to be defined in `solrconfig.xml` or defined with each search request. Each of these will either help fine-tune the query or control the presentation of results.

Query Parsers

During query processing, Solr queries specific fields for matches to the user query. The fields may be a default set configured in advance or specifically defined in the query request. Each field has a type, and each field type has defined rules for how to index content of that type, and how to process queries of that content. In general, rules applied during indexing should be applied during queries to be confident of expected results. For example, if all fields are modified to lower-case during indexing, queries should be modified to lower-case to be sure they match as many terms as possible. These are defined in the field **analyzer** definitions, which include **tokenizers** and **filters** to be applied to indexing and queries. The tokenizers and filters will in many cases modify the original query from the user, perhaps by converting the user's input to lower-case or stripping extra characters like hyphens or other punctuation. There are several dozen options for tokenizers and filters and links at the end of this section will take you to more information about them. You can see the defined field analyzers by looking in the `schema.xml` file for the collection, or in the Admin UI screens for Field Type.

While all of this may seem quite complicated, LucidWorks can be used out of the box with pre-set defaults. If the defaults do not match your desired behavior, however, learning a bit more about how Solr processes content during indexing and handles query requests may be required.

Related Topics

- Apache Solr Reference Guide
- Tokenizers
- Filter Descriptions
- CharFilterFactories
- Language Analysis

Constructing Solr Queries

In basic terms, searches are done with an HTTP GET that specifies the parameters to use for the search. As noted above, the `/lucid` request handler includes several components by default, which means they do not have to be added to the query. If using the `/select` request handler, however, items such as faceting and spell check suggestions would need to be specifically requested.

To search using the `/lucid` request handler, simply point your HTTP client or browser to http://localhost:8888/solr/collection1/lucid?q=some+query. LucidWorks returns XML by default. If you would rather have serialized PHP returned instead of XML, modify the URL to http://localhost:8888/solr/collection1/lucid?q=some+query&wt=phps and the response will be formatted in PHP.

Topics covered in this section:

- Solr Query Parameters
- Query Parsers
- Related Topics

Any request sent to Solr must include the collection name. In the above example URLs, `collection1` refers to the default LucidWorks collection. If you have configured multiple collections, replace "collection1" with the appropriate collection name.

Solr Query Parameters

Solr has a tremendous amount of flexibility for controlling how queries are handled and how results are returned, all of which can be defined as parameters of the query. Some basic parameters to know, however are discussed below.

Parameter	Name	Uses	Example	Defau Lucid\

`q`	query	The main search request and keyword terms for the query.	`q=solr`	No spe default param is defir , which all resu is used a quer is supp the use
`sort`	sort	The field to sort the results by. Must also specify `asc` or `desc` to define the order. Multiple values can be used, separated by a comma. Multi-valued fields cannot be used for sorting.	`sort=dateCreated+asc`	`score`
`fl`	fields	The fields to return with the response.	`fl=id,title`	`id`, `url` `data_s` , `lastM` `mimeTy` `pageCo`
`start`	start	The number of results to skip when returning the results. Can be used with `rows` to provide pagination.	`start=20`	None c LucidW default which i employ instead
`rows`	rows	The number of results to return. Can be used with `start` to provide pagination.	`rows=15`	None c LucidW default which i employ instead
`wt`	writer	The response writer that Solr should use, which defines the format of the results.	`wt=json`	Solr's XML.

`qt`	query handler	The request handler to use to process the query. This can be used instead of a syntax like http://localhost:8888/solr/collection1/lucid? or http://localhost:8888/solr/collection1/select? shown in the examples above, or in conjunction with them to override the default request handler if one is defined.	`wt=/lucid`	`/lucid` default handle
`debug`	debug	Detailed information about the query and results, for debugging purposes. There are four options for this parameter: • `true`: all of the debug information • `query`: information about the query only • `results`: information about the documents returned and how they scored • `timing`: information about how long each component took to complete their tasks	`debug=timing`	In the LucidW Search "explai inform (detail docum scored shown.

There are many other parameters that can be employed, but these are the basic ones that let you submit a query and see some responses. For more detailed information on Solr's query capabilities (some of which depend on the query parser used), see the section of the Apache Solr Reference Guide on Query Syntax and Parsing.

✓ To ensure that your query is indexed and shown in the activity graphs in the LucidWorks Search Admin UI, include the `req_type=main` parameter in your query URL.

Query Parsers

All of query parsers included with Solr are available for use, in addition to the enhanced parser included with LucidWorks. This table shows what are considered the "main" query parsers that are designed for general use. There are also parsers that can be used for specific purposes, listed below.

Name	**ID in LucidWorks**	**Description**

Lucene or Solr	lucene	The Lucene Query Parser, with some Solr enhancements. In the Apache Solr Reference Guide, the section The Standard Query Parser has more details about the options for this parser.
DisMax	dismax	Search across multiple fields, allow +, -, and phrase queries while escaping most other Lucene syntax to avoid syntax errors. More information is available in the Apache Solr Reference Guide in the section The DisMax Query Parser.
Extended DisMax	edismax	A version of the Extended DisMax parser developed by LucidWorks and donated to the Apache Software Foundation for inclusion in Solr. More information is available in the Apache Solr Reference Guide in the section The Extended DisMax Query Parser.
Lucid	lucid	Allows Lucene syntax, enhanced proximity boosting, and query time synonym expansion. Tolerant of syntax errors. More information available in this guide in the section on the Lucid Query Parser.

There are also a number of query parsers which can be used on an ad hoc basis. Each of these are documented in full in the Apache Solr Reference Guide, in the section Other Query Parsers. A few highlights include:

Name	**Description**
Boost	Generates a BoostedQuery which boosts a Query by a FunctionQuery.
Function	Parses a FunctionQuery which calculates a function over field values.
Field	Generates a query on a single field.
Nested	Delegates to another query parser, which can be used to override the default parser for a specific purpose.
Prefix Query Parser	Generates a prefix query on a single field.
Raw	Generates a raw unanalyzed term query.
Spatial Filter	Generates a query which filters results by a defined distance from a point in space.

Other query parsers are also available.

Related Topics

- Query Syntax and Parsing, with several sub-pages for query parsers and local parameters

Solr Query Responses

Structure of the Response

All Solr responses have at least two sections, the `responseHeader` and the `response`.

The responseHeader Section

The `responseHeader` includes the status of the search (`status`), the processing time (`QTime`), and the parameters (`params`) that were used to process the query.

The response Section

The `response` includes the documents that matched the query, in `doc` sub-sections. The fields return depend on the parameters of the query (and the defaults of the request handler used). The number of results is also included in this section.

The highlighting Section

The `highlighting` section will show, for each document in the response, the sections of text in the document that should be highlighted. If using the `/lucid` request handler, they will be shown as snippets of text, with HTML `<span>` tags around them. Your client can consume those and you can format them by specifying the `highlight` class in your CSS however you'd like.

If using another request handler, such as `/select`, that does not have predefined configuration options for highlighting, you may need to set the parameters in your request. There are quite a few Solr parameters to control highlighting and the output in the response. For more details, see the section of the Apache Solr Reference Guide for Highlighting.

The facet_counts Section

The `facet_counts` shows the facets that have been constructed for the result list, including the facet fields and facet values (with counts) to populate each field.

The spellcheck Section

The `spellcheck` will include suggestions for possible spelling errors in the user's query.

The debug Section

The `debug` section will contain the detailed information about how the query was processed. This section will only be returned if the `debug` parameter was used with the query.

There are many sub-section of this section, including:

- `explain`: Information about how each document scored according to the in relevancy ranking algorithm.
- `timing`: Information of how long each component took.
- `parsedquery`: The query string as submitted to the query parser.

Calculating the debug info, particularly the scores, is expensive in terms of processing power, so it should only be used when needed to debug query results.

Ack! What Do Those Scores Mean?

The `explain` sub-section of `debug` is the section that gives you information about the relevancy scores of each document returned in the query. It's the section you'll want to look at if you want to know why one document is ranked higher than another. But it's pretty complex.

The `explain` section shows you each factor that went into the final score and how it was weighted. There may be specific boosts defined (LucidWorks for example boosts a document when the query terms are found in the title, among others), the frequency of the term in the document may be high relative to the frequency of the term in all documents (a relationship called the "term frequency-inverse document frequency", or TF-IDF), or the term may have matched a field that is smaller than others (such as "author" instead of "body").

Some make the mistake of focusing on the score of a document in absolute terms instead of looking at a document's score relative to the other documents returned. This is an error because scoring of a single document is always relative to other documents in the index, and your index changes over time. The point of looking at scoring should be instead to understand why a document is ranked higher or lower than other document.

More information on `explain` can be found in the section describing the Explain Info of the LucidWorks Search UI.

Format of Results

The default format for search results in LucidWorks Search is XML. There are other options available - such as JSON, PHP, and CSV, among others - and you request the results in that format when sending the query. This is defined with the `wt` parameter.

The data is returned as a standard Solr search data structure, formatted either as XML, Ruby, Python, PHP, PHPS, and even server-side XSL. For more information, see the section in the Apache Solr Reference Guide on Response Writers.

Related Topics

- Understanding and Improving Relevance
- Explain Info

- Response Writers

Query and Response Examples

LucidWorks Search includes a simple Search UI, but if you are going to build your own user interface, or your own application to access the data stored in LucidWorks, you will need to access the underlying engine directly.

LucidWorks is built on Apache Solr, so the techniques necessary for performing a search against it are the same as those for performing a search against Solr. In other words, an HTTP call to a URL of:

```
http://127.0.0.1:8888/solr/collection1/select/?q=NickChase
```

Would return a result such as this:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
   <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">99</int>
      <lst name="params">
         <str name="q">NickChase</str>
      </lst>
   </lst>
   <result name="response" numFound="151" start="0">
      <doc>
         <str name="geo">none</str>
         <str name="id">29059644164939776</str>
         <int name="retweetCount">0</int>
         <str name="source">web</str>
         <str name="text">Working on a Twitter app; anybody got a preferred Java
Twitter library?</str>
         <arr name="text_medium">
            <str>NickChase</str>
            <str>en</str>
            <str/>
            <str>web</str>
            <str>Working on a Twitter app; anybody got a preferred Java Twitter
library?</str>
            <str>2011-01-23T06:15:33.000Z</str>
            <str>0</str>
         </arr>
         <date name="timestamp">2011-02-13T14:06:53.191Z</date>
         <arr name="userId">
            <str>99999999</str>
         </arr>
         <str name="userLang">en</str>
         <str name="userName">Nicholas Chase</str>
         <str name="userScreenName">NickChase</str>
      </doc>
      ...
   </result>
</response>
```

You can then consume that XML from within your application.

While XML is the default output format, LucidWorks supports multiple formats, including JSON, CSV, and even object formats such as PHP, Java, and Python.

In general, to change the output format, use the `wt` parameter, as in:

```
http://127.0.0.1:8888/solr/collection1/select/?q=NickChase&wt=json
```

This provides a response of

```
{
   "responseHeader":{
      "status":0,
      "QTime":1,
      "params":{
         "wt":"json",
         "q":"NickChase"
      }
   },
   "response":{
      "numFound":151,
      "start":0,
      "docs":[
         {
             "id":"29059644164939776",
             "userName":"Nicholas Chase",
             "userScreenName":"NickChase",
             "userLang":"en",
             "source":"web",
             "text":"Working on a Twitter app; anybody got a preferred Java Twitter
library?",
             "retweetCount":0,
             "timestamp":"2011-02-13T14:06:53.191Z",
             "geo":"none",
             "text_medium":["NickChase","en","","web","Working on a Twitter app;
anybody got a preferred Java Twitter library?",
  "2011-01-23T06:15:33.000Z","0"],
             "userId":["99999999"]
         }
      ...
      ]
   }
}
```

The structure of the results depends on the options you choose in the request string. For example, you can specify faceting and highlighting;

```
http://127.0.0.1:8888/solr/collection1/select/?q=twitter&facet=on&facet.field=userScreenNa
```

Which gives a result such as this:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
   <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">359</int>
      <lst name="params">
         <str name="facet">on</str>
         <str name="facet.field">userScreenName</str>
         <str name="hl.fl">text</str>
         <str name="hl">true</str>
         <str name="q">twitter</str>
      </lst>
   </lst>
   <result name="response" numFound="2190" start="0">
      <doc>
         <str name="geo">none</str>
         <str name="id">38402455221829632</str>
         <arr name="objectType">
            <str>twStatus</str>
         </arr>
         <int name="retweetCount">0</int>
         <str name="source">&lt;a href="http://twitter.com/" rel="nofollow"&gt;Twitter
for iPhone&lt;/a&gt;</str>
         <str name="text">RT @Onventive: Really useful Twitter Android code RT @enbake
Developing an android twitter
                           client using twitter4j http://is.gd/1YUFyY #a ...</str>
         <arr name="text_medium">
            <str>t4j_news</str>
            <str>en</str>
            <str/>
            <str>&lt;a href="http://twitter.com/" rel="nofollow"&gt;Twitter for
iPhone&lt;/a&gt;</str>
            <str>RT @Onventive: Really useful Twitter Android code RT @enbake
Developing an android twitter
                           client using twitter4j http://is.gd/1YUFyY #a ...</str>
            <str>2011-02-18T01:00:33.000Z</str>
            <str>0</str>
         </arr>
         <date name="timestamp">2011-02-18T01:45:05.52Z</date>
         <arr name="userId">
            <str>88888888</str>
         </arr>
         <str name="userLang">en</str>
         <str name="userName">t4j_news</str>
         <str name="userScreenName">t4j_news</str>
      </doc>
      ...
   </result>
   <lst name="facet_counts">
      <lst name="facet_queries"/>
      <lst name="facet_fields">
         <lst name="userScreenName">
            <int name="beaker">189</int>
            <int name="cloudexpo">35</int>
            <int name="randybias">35</int>
            <int name="getjavajob">26</int>
            ...
         </lst>
      </lst>
      <lst name="facet_dates"/>
      <lst name="facet_ranges"/>
   </lst>
   <lst name="highlighting">
      <lst name="38402455221829632">
         <arr name="text">
            <str>RT @Onventive: Really useful &lt;span
class="highlight"&gt;Twitter&lt;/span&gt; Android code RT
                    @enbake Developing an android &lt;span
class="highlight"&gt;twitter&lt;/span&gt; client</str>
         </arr>
      </lst>
   ...
</response>
```

Notice the structure of the search response: it starts with the `responseHeader` block, which provides information such as the query, whether you have specified highlighting, and so on.

Next is the `result` block, which shows the actual documents returned by the search, along with the `numFound` and `start` attributes, which specify the total number of results and the starting position for the results returned in this response. For each document, LucidWorks Search returns all fields that are marked as `stored=true` in the field definition.

If you have specified faceting, next you will see facet counts for each field specified. You can then use that information to build links to your narrowed search. For example, we started with the query:

```
http://127.0.0.1:8888/solr/collection1/select/?q=twitter&facet=on&facet.field=userScreenNa
```

If you then wanted to build a link to results narrowed on the `userScreenName cloudExpo`, it would look like this:

```
http://127.0.0.1:8888/solr/collection1/select/?q=twitter&facet=on&hl=true&hl.fl=text&fq=us
```

This way you have the same set of results, with the additional filter query of `userScreenName:cloudExpo`, which selects only the documents with a `userScreenName` field of `cloudExpo`.

After the facet information comes the `highlighting` block. Highlighting consists of snippets with the relevant information marked up appropriately. (By default, terms are marked up as a `span` with the class `highlight`, so you can use CSS to style them however you like.) Each snippet is contained in a block that refers back to the `id` value of the original document. So in this case, the `name` attribute of `38402455221829632` refers back to `doc` with an `id` of `38402455221829632`. You can then use this information to build your web application.

As far as how to actually use these responses, you can either work with them directly, or use the Solr API as provided for your programming language. For example, a SolrJ request looks something like this:

```
SolrServer server = new
CommonsHttpSolrServer("http://localhost:8888/solr/collection1");

SolrQuery query = new SolrQuery();
query.setQuery( "twitter" );
query.addSortField( "timestamp", SolrQuery.ORDER.desc );

QueryResponse rsp = server.query( query );
SolrDocumentList docs = rsp.getResults();
for (SolrDocument doc : docs){
 System.out.println((String)doc.getFieldValue("id")+": ");
 System.out.println((String)doc.getFieldValue("userScreenName")+" --
"+(String)doc.getFieldValue("text"));

}
```

Here you are creating a connection to the server, then creating and executing the request. From there, you can manipulate documents as you see fit.

APIs exist for most programming languages. You can find a list of bindings on the Solr Wiki.

Related Topics

- Searching chapter from Lucid Imagination's Solr Reference Guide

Understanding and Improving Relevance

Relevance is one of the most complex components of a search engine implementation. Two users performing the same query will likely have differing opinions about what the best match for their query is because judging relevance inherently has a level of subjectivity to it. However, there are some ways to assess relevance and adjust how documents are scored to improve ranking. This section discusses the various approaches to analyzing a problem with relevance (real or perceived) and possible solutions.

For more background on how LucidWorks approaches relevance, see the discussion in the section on Managing Queries.

Topics discussed in this section:

Judging Relevance

Relevance should always be judged in the context of a specific index and a set of queries for that index. Often the best way to achieve this is through query log analysis. In a typical query log analysis, the top 50 queries (give or take) are extracted from the logs, plus ten to twenty random queries. Next, one to three users enter each query into the system and then judge the top ten (or five) results. Judgments may be done using values of relevant, somewhat relevant, not relevant and embarrassing. The goal of relevancy tuning is to maximize the number of relevant documents while minimizing the number of irrelevant ones.

By recording these values and repeating the test over time, it becomes possible to see if relevancy is getting better or worse for the particular system in question.

An alternative method for judging relevance is to use what is commonly referred to as A/B testing. In this approach, some set of users are shown results using one version of the index while another set of users is shown the results from a different version. To judge the success of a particular approach, user clicks are tracked and analyzed to determine which approach provides better results.

Click Scoring Relevance Framework

One important aspect of LucidWorks relevance scoring functionality is the ability to boost documents that prior users have selected. This functionality is the Click Scoring Relevance Framework and can be enabled through the Administrative User Interface.

Indexing Techniques

Several techniques can be employed during indexing to alter default relevance ranking. While these techniques almost always have to be mirrored on the query side, they will be considered here as they originate during indexing.

Document and Field Boosting

When indexing using the Solr APIs it is possible to mark one document or field as being more important than other documents or fields by setting a boost value during indexing. These boost factors are then multiplied into the scoring weight during search, thus potentially boosting the result higher up in the result set. This type of boosting is usually done when knowledge about a document's importance is known beforehand. However, index time boosting only provides 255 distinct values of granularity and if a change is needed to the boost value, the document must be re-indexed.

LucidWorks Search also includes a way to boost fields in a document based on their length. In theory, if a term that the user has searched for appears in a field that is significantly shorter than other fields (such as the title), it should be boosted more than if the term appears in a longer field (such as the body). The short field boost factor provides three approaches: "none", which provides no boost; "moderate", which uses the LucidSimilarityFactory to provide a smaller boost than the standard Lucene calculations; and "high", which uses the Lucene DefaultSimilarityFactory to calculate the boosts. This functionality is used during indexing - during query time, the standard Lucene calculations are used.

Stemming and Lemmatization

Stemming is the process of reducing a word to a base or root form. For example, removing plurals, gerunds ("ing" endings) or "ed" endings. Lemmatization is a variation of stemming that leaves a whole word in place, while stemming need not do that. There are many stemming theories and techniques. Some are quite aggressive, stripping words down to very small roots, while others (called light stemmers) are less aggressive.

LucidWorks includes many options for stemming but it is also possible to plug in a custom analyzer or use other Solr or Lucene analyzers not included. As a general rule of thumb, it is usually best to start with a light stemming approach that removes plurals and other basics techniques and then progress to more aggressive stemming only after performing some relevance testing as described in Judging Relevance.

Default stemming in LucidWorks uses the Lucid Plural Stemmer for the default English text analysis Field Type which simply stems plural words into their singular form, although rules can be added to a rules file to protect and specially translate words or even add or modify stemming rules as needed (see the section Lucid Plural Stemming Rules.) The Lucid KStem stemmer is available, which is based on the KStemmer light stemming library. More aggressive stemmers are also available, like Dr. Martin Porter's Snowball stemmers (choose the "text (English Snowball)" Field Type).

To experiment with different stemmers, there is a well-defined mechanism in the embedded Solr for plugging in stemmers via the Analysis Process. There is also an easy to use Admin interface for testing the analysis process located in the Solr Admin screens.

Alternate Indexing Fields

When indexing, it is often useful to apply several different analysis techniques to the same content. For example, providing a default case-insensitive search is often the best choice for general users, but expert users will often want to do exact match searches which additionally require a case-sensitive field. In Solr, this can be accomplished by using the `<copyField>` mechanism, as described in the Apache Solr Reference Guide section on Copying Fields. This currently can be set up by editing the `schema.xml` file and restarting LucidWorks.

Other examples of alternate fields include different stemming approaches, using character-based and word-based n-grams, and stripping punctuation, accents and other marks. As with any change, though, some time should be taken to evaluate whether it is an improvement.

Stop words

Removing stop words (such as a, the, of, etc.) from the index and stripping them from queries is a common technique for reducing the size of an index and improving search results, despite the fact that it throws away information. While LucidWorks can remove stopwords at indexing time, it does not by default. Instead, LucidWorks excludes stopwords at query times, except in certain types of queries where they are used to better clarify the user's intent (such as in phrases). Both the Extended Dismax Query Parser and the Lucid Query Parser can take advantage of stopwords to improve results by using them in an n-gram approach.

Search Techniques

On the search side, there are many techniques for improving relevance, the most important of which is user education. While the techniques described below can make things much easier for users, educating users on how to use the proper query syntax, when to use it, and how to refine queries can be instrumental in enhancing the relevance of search results. Obviously, not all users will read manuals or take the time to learn new query syntax, so the following techniques can be used to achieve better results in many situations.

Query Term Boosting

Similar to Document/Field boosting, terms in a query can be boosted. Boosting a query term implies that the term in question is somehow more important than the other terms in the query. One advantage of query time boosting is an expanded level of granularity is available for expressing the boost value. Additionally, the boost value is not "baked in" to the index, so it is easier to change.

Synonym Expansion

Synonym expansion is a common technique that looks up each token in the original query and expands it with synonyms; strictly speaking, synonym expansion mostly improves the ability to get more relevant documents (also called recall) rather than relevance ranking or the exclusion of irrelevant documents. For instance, a user query containing "USA" could be expanded to look like "(USA OR "United States" OR "United States of America")", which will likely bring back results that the user intended to retrieve, but did not fully specify. In LucidWorks Search, it is easy to specify a list of synonyms that can be used for expansion. Synonym lists are best created by analyzing query logs and then looking up synonyms for common query terms and then testing the results. Generic synonym lists (like those obtained from WordNet) can be useful, but care must be taken as too many synonyms can be problematic for users, especially if they are not appropriate for the genre of the index. It is, however, quite common to produce synonym lists contain common abbreviations, numbers (for example, 1 -> one, 2 -> two, and so on) and acronyms.

Unsupervised Feedback

Unsupervised feedback is a relevancy tuning technique that executes the user's query, takes the top five or ten documents from the result, extracts "important" terms from each of the documents and then uses those terms to create a new query which it then executes and whose results are returned to the user. This is all done automatically in the background with no interaction required by the end user. As an example, if the user searches for the word "dog" and the top three documents are (for the sake of example):

1. Great big brown dogs run through the woods.
2. Dogs don't like cats.
3. A poodle is a type of dog.

The feedback query might look something like `(dog) OR (great OR big OR brown OR dog OR run OR woods OR cat OR poodle)`.

Since these terms co-occur with the word "dog" in high ranking documents, the theory goes that they are terms that can help better specify the user's short query. Unsupervised feedback is often viewed as a helper, but it does rely on the assumption that the top few documents are highly relevant to the search. If they are not, then the results incorporating feedback will likely be worse than those without feedback.

Unsupervised feedback is optional in LucidWorks and is *disabled* by default. It may be enabled by checking the **Enable Unsupervised Feedback** check box in the query settings panel of the Administration User Interface.

Supervised Feedback

Supervised feedback is similar to unsupervised feedback except that users explicitly pick which results are relevant, usually by clicking the result or checking a box indicating it is relevant. The LucidWorks feedback component does not currently support supervised feedback.

Boosting Specific Results with Query Elevation Component

The Query Elevation Component is a Solr SearchComponent that enables editorial control of results by allowing specific results to be placed in specific positions for a given query. For example, if a particular FAQ answer is buried in the result set for a query, then it can be "promoted" to occur as the first result by setting making it so in the Query Elevation Component. It can be configured by editing `solrconfig.xml` manually, using the Settings REST API, or with the default Search UI of LucidWorks.

Debugging Relevance Issues

It is often the case that particular queries cause users to question the relevance of the documents returned. There are several key things to keep in mind when debugging these problems. First and foremost, determine how important the query is. Is it a common query or does it only occur once in a great while? If it is a relatively rare query, it may not be worth the effort to try to "fix" it. Second, do not overtune. Fixing one query may break ten other queries. Unless there is an obvious fix, it is recommended that relevance judgments be established first so that any breakages can be quickly caught. After the need to fix a problem is established, there are some techniques to do just that.

In the default LucidWorks Search UI, links will appear under each search result for "Explain"; clicking that will show the scoring of each document for the query.

Using Luke

Another useful tool for evaluating relevance scores applied to documents is Luke, which is an easy to use GUI that provides valuable information about the underlying Lucene index. Its features include document browsing, query testing, term browsing (including high frequency terms) and statistics about the collection as a whole. To use Luke with LucidWorks Search, launch it using the script located in the `$LWE_HOME/app/luke` directory.

Once Luke is launched, point it at the LucidWorks Search index directory (such as `$LWE_HOME/data/solr/cores/collection1_0/data/index`, replacing "collection1_0" with the actual collection path you want to look at) and open the index. From there, the most useful actions are to view the high frequency terms, and also particular documents (under the Documents tab) using the "Browse by term" and "Browse by document number" options. Key items to look for are missing documents and fields, terms or words that are not tokenized "correctly". Correctly, in this case, does not necessarily mean the analysis process was wrong, it may mean that the output is not what a user would expect. For instance, the word may be stemmed in an unexpected way.

Luke in LucidWorks Search

LucidWorks Search packages a version of Luke, which is provided 'as is'. It can be found at `$LWE_HOME/app/luke` and launched by running the `luke.sh` script for Linux/Mac or the `luke.bat` script for Windows.

Correcting Issues

Once issues have been discovered and understood, then it is best to develop a strategy for fixing or working around the issue. This can often mean changing analysis steps. To help better visualize the analysis process, Solr ships with an analysis tool that effectively shows the outcome of each analysis step on both the indexing side and the query side. To use this tool, point a browser at http://localhost:8888/solr/collection1/admin/analysis.jsp? and enter the text to be analyzed. By trying out the text with different analysis capabilities (by selecting different Fields or Field Types), it is possible to better understand why matches may or may not occur.

Improving relevance with external boost data

The standard mechanism in Solr for adding external field data (which may affect ranking) is through the use of `ExternalFileField` type. This mechanism is sufficient when adding simple string or numeric values to be processed by function queries, but it's not sufficient to express more complex scoring mechanisms, based on other regular query types.

Related Topics

- Relevance chapter from Lucid Imagination's Solr Reference Guide
- Debugging Search Application Relevance Issues, by Grant Ingersoll, hosted at SearchHub.org.

Click Scoring Relevance Framework

It is often desirable to adjust the scoring of results based on user feedback, whether that feedback is explicit or implicit. Query logs provide a wealth of information that indicates what users were searching for and which results they found relevant to the query.

If certain documents are often selected as answers to queries, then it makes sense to increase their ranking based on their popularity with users.

LucidWorks includes a component that enables administrators to add this type of information to the index. This component is referred to as the Click Scoring Relevance Framework (or Click Scoring, for short). The framework includes tools for query log collection, processing of logs and robust calculation of the data used to boost certain documents. It is possible to supply boost data prepared without Click Scoring tools included with LucidWorks, however the data must be available in a predefined location and follow a specified text format. More details about how Click Scoring works and information about other advanced configuration parameters are described in Using Click Scoring Tools.

This component can be enabled in the Query Settings section of the Admin UI or with the `click`-related parameters of the Settings API. Once enabled, a job must be scheduled to process the click logs and create the data for boosting documents based on prior clicks.

Topics covered in this section:

Using Click Scoring with NearRealTime Search

Enabling Solr's Near RealTime (NRT) search by configuring the `update_handler_autosoftcommit_*` parameters with the Settings API or the Auto-soft-commit* settings in the Admin UI has some impacts on how user clicks are processed by LucidWorks.

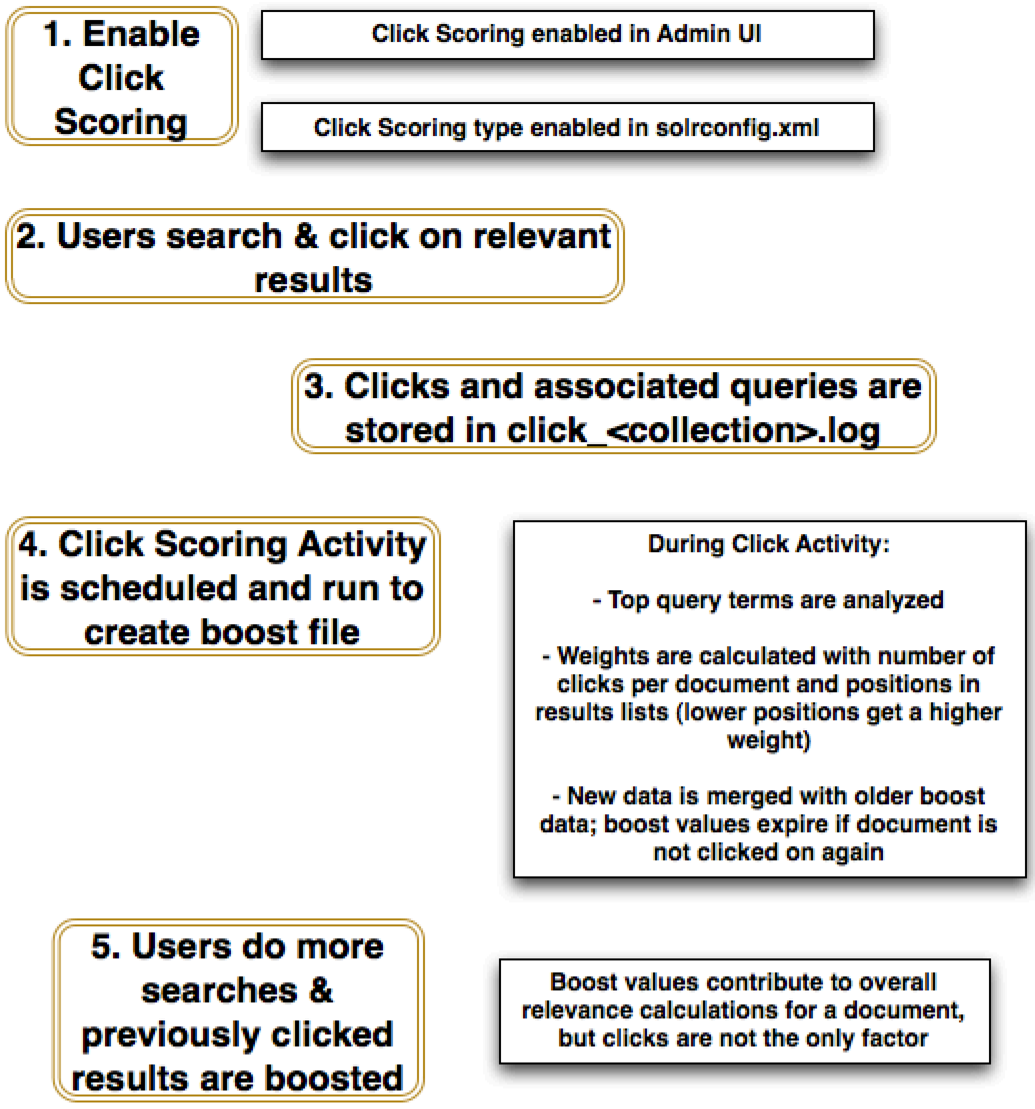
In order to avoid performance issues with NRT search when Click Scoring is enabled, documents added between the last "hard" commit and the current "soft" commit are **not** augmented with click-through data.

Deletions since the last hard commit are processed as usual (i.e., documents deleted are not visible), but their term statistics are still included in the global term statistics (which includes the fields added by Click). Added documents since the last hard commit will not get any click-related fields until the next hard commit, even if a document with the same unique key was deleted and replaced by a new, updated, version of the document.

Functionality of Click Scoring

When users select a particular document for viewing from a list of search results, we can interpret this as implicit feedback that the document is more relevant to the query than other documents in the results list. This further indicates a strong association between the terms of the query and the selected document, because apparently users believe the selected document matches their query better than other returned documents.

This graphic gives an overview of how Click Scoring works:


1. Enable Click Scoring
Click Scoring enabled in Admin UI
Click Scoring type enabled in solrconfig.xml
2. Users search & click on relevant results
3. Clicks and associated queries are stored in click_<collection>.log
4. Click Scoring Activity is scheduled and run to create boost file
During Click Activity:
- Top query terms are analyzed
- Weights are calculated with number of clicks per document and positions in results lists (lower positions get a higher weight)
- New data is merged with older boost data; boost values expire if document is not clicked on again
5. Users do more searches & previously clicked results are boosted
Boost values contribute to overall relevance calculations for a document, but clicks are not the only factor

The reinforcement of ranking and terms is counterbalanced by the "expiration" of the past history of click-through events, to avoid situations when documents that are selected many times start to permanently dominate the list of results. Without expiration of old history, these results may become selected even more often at the expense of other perhaps more relevant documents that did not enjoy such popularity over time.

Click Scoring implements several major areas of functionality related to the processing of click-through events:

- collection of query logs and click-through logs
- maintenance of historical click data to control the expiration of past click-through events
- aggregation of log data, calculation of click-induced weights and association of query terms with target documents
- integration of boost data with the main index data

These areas of functionality are described in the following sections.

Collection of Query and Click-Through Logs

This part of the functionality is implemented in the LucidWorks Search UI. If LucidWorks is used with a custom search user interface that directly uses the LucidWorks REST API and the underlying Solr API, a Click Scoring API can be used to record user clicks and query events. The final boost data file follows a simple text format, so this can be also supplied by an external process if desired. See Using Click Scoring Tools for more details about such custom integration.

Both the queries and the click-through events are logged to the same log file. The default location of this file is in `$LWE_HOME/data/logs/click-collection.log`, where `collection` is the name of the collection (for example, `click-collection1.log` the default LucidWorks collection, collection1).

Collection of User Clicks

If Click Scoring is not enabled, LucidWorks Search does not gather information about user document clicks.

When using Index Replication this log data is not replicated to slave nodes. Since the Click Scoring API points to the LucidWorks Search Core component, which is only used on a single node, and not directly to the indexes, it is not required to replicate the log files across shards. The latest version of the calculated boost data (after the logs have been processed) is replicated together with the main index files, so that the slave nodes can perform click-based scoring the same way as the master node that calculated the boost data. In addition, Click Scoring is not available in SolrCloud mode.

Maintenance of Historical Click Data

Each time the Click Scoring logs are processed, the system stores a copy of the current `click-collection.log` (by default, this is in `$LWE_HOME/data/solr/cores/collection /data/click-data/`). Other data produced during Click processing is also stored in that location.

Over time the amount of data collected could be significant. LucidWorks does not delete this data automatically, because query and click-through logs are a valuable resource and can be used for other data mining tasks. If the size of this data becomes a concern, all subdirectories in that location can be removed except for `current/` and `previous/` directories that preserve the current and previous boost data.

Processing of click-through logs

Raw logs need to be processed before they can be used for scoring, and historical data needs to be appropriately accumulated (taking into account the gradual expiration of older data mentioned above). This processing step can be started with the Activities API, or scheduled to run periodically using the Admin UI by setting a recurring activity in the Index Settings screen of the Admin UI.

This process results in the creation of calculated click boost data, which is by default located in `$LWE_HOME/data/solr/cores/collection/data/click-data/current`.

Integration of click boost data with main index

If Click Scoring is enabled and boost data exists, the boost data is integrated on the fly with the main index when new documents are indexed, an index optimization is run, or a full re-index is executed. Most frequent query terms are added as a field to respective documents, and weights of these documents are adjusted.

The field names added by Click Scoring are configurable, but assuming their prefix is set to the default value of `click` the following fields will be created from boost data and automatically populated:

- `click`: an indexed, not-stored field with a single term "1", whose purpose is only to convey a floating-point field boost value. Field boost values have limited resolution, which means that small differences in boost value may yield the same value after rounding.
- `click_terms`: an indexed, stored, and analyzed field that contains a list of top terms associated with the document (presumably obtained through analysis of click-through data). This field's Lucene boost is also set to the boost value for this document obtained from the boost data file.
- `click_val`: an indexed, stored field that contains a single term: a string representation of the boost value for this document. This format is suitable for processing in function queries.

Using Click Scoring information

There are several ways that the added Click Scoring information can affect ranking of results. By default, LucidWorks Search is configured to use Click Scoring data as an additional field in a query parsed by the Lucid Query Parser. Other methods can be configured manually, and may involve using `click_val` field as an input to a function query. This section describes the `lucid` query parser method, which is the default.

When Click Scoring is enabled via the Admin UI, a field with a boost `click_terms^5.0` is automatically added to the list of fields for the search handler (which uses a `lucid` query parser).

This means that query terms will be matched also with the `click_terms` field using the relative weight of 5.0 (this weight can be changed only via Settings API or by manual editing of `solrconfig.xml`). The score contribution of this match will be related to this weight, the term frequency/inverse document frequency scoring formula for this field, and the usual `lucid` (extended `dismax`) scoring rules.

The end result of this query processing is that documents that contain in their `click_terms` field terms from the query will have a higher ranking, proportionally higher to the popularity of the document (the number of click-throughs) and the overlap of query terms with `click_terms`.

Using Click Scoring Tools

This functionality is **not available** with LucidWorks Search on AWS or Azure

The Click Scoring Tools package is a set of tools for analyzing query and click-through logs in order to obtain relevance-boosting data. This boost data can then be used by other Click Scoring components such as `ClickIndexReaderFactory` and the `lucid` query parser to adjust document ranking based on the click-through rate and common query terms.

File Formats

The Click Scoring Tools package reads and generates files that follow specific formats, which are summarized below. All files are plain text files with tab-separated records, one record per line.

Query and Click-through Log Format

Click Scoring tools expect this file to be located in `$LWE_HOME/data/logs/click-<collectionName>.log` .

```
Q TAB queryTimestamp TAB query TAB requestID TAB numberOfHits
C TAB clickTimestamp TAB requestID TAB documentID TAB position
```

The fields are:

Field	Description
`Q` or `C`	Identifies the type of the record, either a query log record or a click-through log record
`queryTimestamp`	A long integer representing the time when the query was executed
`query`	The user query, after basic escaping (removal of TAB and new-line characters)
`requestID`	A unique request identifier related to query and timestamp
`numberOfHits`	The total number of results matching the query
`clickTimestamp`	A long integer representing the time of the click-through event
`requestID`	The same value as above for the `Q` record
`documentID`	The `uniqueKey` of the document that was selected
`position`	The 0-based position of the selected document on the list of results

Boost File Format

This file is usually generated as a result of the Click Scoring processing of log files, but it could be also supplied by some other external process. Click Scoring expects this file to be located in `$LWE_HOME/data/solr/cores/collection/data/click-data/current`.

```
documentID TAB list(topTerms) TAB list(boost) TAB list(updateTimestamp)
```

The fields are:

Field	Description
`documentID`	The `uniqueKey` of the document
`list(topTerms)`	A comma-separated list of pairs in the format phrase:weight
`list(updateTimestamp`	A comma-separated list of long integer timestamps, which affect how the current boost data will be aggregated with the next version of boost data. This element is optional and it's for internal use by Click Scoring Tools

Click-induced Boost Calculation

When Click Scoring tools are run (using the `ClickAnalysisRequestHandler`) old boost data (if present) is merged with the new boost data, processed by a `BoostProcessor` to produce the new numeric boost value per documentID, and a new list of top-N shingles per documentID. Previous values of the floating-point boost are preserved in a boost history field, so that they may be considered during the next round of calculations.

The default configuration uses a `BoostProcessor` that discounts historical boost values depending on the passed time by applying an exponential half-life decay formula. Such discounted historical values are then aggregated with the current values. This method of aggregation reflects both past history of click-throughs and also reacts closely to recent click-through events.

ClickAnalysisRequestHandler

The `ClickAnalysisRequestHandler` initiates and monitors the click-through analysis. The tools for Click Scoring processing are available via `com.lucid.handler.ClickAnalysisRequestHandler`, which can be activated from the `solrconfig.xml` configuration file the same way as any other request handler.

The configuration that ships with LucidWorks Search already contains a section that activates this handler, under the relative path `/click`.

This handler accepts a `request` parameter, which can take one of the following values:

- STATUS - return the status of the ongoing analysis, if any. Example request:

```
curl "http://localhost:8888/solr/collection1/click?request=STATUS"
```

Example response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <lst name="responseHeader">
  <int name="status">0&lt;/int>
  <int name="QTime">205&lt;/int>
 </lst&gt;
 <str name="logDir">java.io.File:.../logs</str>
 <str name="prepDir">java.io.File:.../click-prepare</str>
 <str name="boostDir">java.io.File:.../click-data</str>
 <null name="dictDir"/>
 <str name="processing">Idle.</str>
</response>
```

- `PROCESS` - start the clickthrough processing. If the processing is already running, an error message will be returned and this request will be ignored.
  Example request:

```
curl "http://localhost:8888/solr/collection1/click?request=PROCESS"
```

  Example response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">136</int>
 </lst>
 <str name="result">Clickthrough analysis started.</str>
</response>
```

  Subsequently, the status returned after all processing is finished will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">1</int>
 </lst>
 <str name="logDir">java.io.File:./logs</str>
 <str name="prepDir">java.io.File:./click-prepare</str>
 <str name="boostDir">java.io.File:./click-data</str>
 <null name="dictDir"/>
 <str name="processing">Stopped: Stage 3/3: prepare=finished, ok
aggregate=finished, ok boost_calc=finished, ok</str>
</response>
```

- `STOP` - stop the currently ongoing analysis, if any is running.
  Example request:

```
curl "http://localhost:8888/solr/collection1/click?request=STOP"
```

  Example response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">0</int>
 </lst>
 <str name="result">There is no running analysis to stop - ignored.</str>
</response>
```

When processing is finished, new versions of boost files will be placed in the `current` directory, and previous boost data will be moved to the `previous` directory. At this point in order to read the new boost values SolrCore needs to be reloaded (for example, by issuing a `<commit/>` update request).

In addition to the `request` parameter this handler supports also the following parameters:

- `commit` (default to false) if set to true, then after the processing is finished the handler will automatically execute a commit operation to reopen the IndexReader and to load the newly calculated boost data. Please note that Solr supports only a single global commit, which means that all other open transactions (such as ongoing indexing) will also be committed at this time.
- `sync` (default to false) if set to true, then the processing will be executed synchronously, blocking the caller and returning only when all processing is finished. Default is to run the processing in a separate background thread.

Click Scoring Tools and Index Replication

When LucidWorks Search is configured to use Index Replication the boost data files (by default, in `$LWE_HOME/data/solr/cores/collection/data/click-data`*) will also be automatically replicated. Due to the internal limitations of Solr's* `ReplicationHandler` *the boost data file will be located _inside* the main index directory on the slave nodes, but it will be properly recognized by the Click Scoring components on the slave nodes.

Click Scoring does not currently work with the SolrCloud functionality available with Solr 4.

For the replication of `boost.data` to work the `solconfig.xml` must contain the following line in the `<mainIndex>` section:

```
<mainIndex>
   ...
   <deletionPolicy class="com.lucid.solr.click.ClickDeletionPolicy"/>
   ...
  </mainIndex>
```

Business Rules Integration

This functionality is **not available** with LucidWorks Search on AWS or Azure

A Rules Engine is designed to allow business users to write rules that can effect the processing of search results. For instance, an ecommerce company may wish to alter the search results to boost particular documents based on a sale or the HR department of a company may wish to make sure the document covering 401K benefits is always at the top of a search for 401K. In essence, a rules engine integrated with a search engine allows business to dynamically effect relevance based on business needs without having to write extensive, low-level client-server code. Instead, they can express rules in a declarative programming language that are much simpler to understand without the complexity of logic that goes into writing code in a programming language like Java or Ruby. In a rules engine, users express rules to be matched along with the steps to do if a rule is matched, using simple if-then statements and the engine figures out which rules should be fired given the facts present in the system. For example, a set of rules may look like:

```
if owner.hasDog then recommend dog food
if owner.hasCat then recommend cat food
if owner.gender is female and store is "sporting goods" then discount golf clubs 20%
```

The important thing to note in this example is we didn't have to do any complex logic to tie these rules together. We simply express the conditions and the things that should happen if a condition is true. The engine is responsible for figuring out which rules should fire based on the information (facts) it has to work with when evaluating the rules. It is also important to note that at any given execution of the engine some, all, or none of the conditions may be met depending on the facts in the system, thus implying that all of the "then" clauses will be executed.

When should I use business rules?

There is a time and place for the use of business rules. Generally speaking, they are most effectively used in situations where non-developers are expected to apply changes to the search results based on business conditions. They are not a replacement for code that integrates search into an application, but instead should be thought of as a way for companies to fine tune user interactions with a system without the need to go through extensive (and expensive) development cycles. It also is not a substitute for general relevance tuning across a broad set of queries nor is it appropriate for ranking modifications that are best done at a lower level in the search engine.

⚠ Business rules integration is enabled by default. However, if not using business rules in your implementation, it makes sense to disable it to prevent unnecessary processing. See the section on Disabling Business Rules for details on how to turn rules off.

How to Implement Business Rules

There are two main areas to cover for implementing business rules with LucidWorks Search:

First, determine how the rules will be implemented. There are a variety of methods, each described in the section on LucidWorks Rules Integration.

Second, define the rules themselves. LucidWorks Search has integrated Drools, and you'll want to look at the section on Drools Integration for information on how to construct a rules file.

There are Example Rules and Recipes. If you're not using rules at all, you can disable business rules.

Integrating with your Rules Engine

If you already have a rules engine (such as ILOG's JRules or Fair Isaac's Blaze Advisor) you can hook them into LucidWorks by implementing a RulesEngine class that talks to your rules engine. Naturally, you can also implement your own SearchComponent, DocTransformer, UpdateRequestProcessor, etc., if the ones shipped with LucidWorks do not meet your needs.

LucidWorks Rules Integration

LucidWorks Search includes several integration points to plug in business rules engines as well as a default implementation of a rules engine based on the open source Drools from Red Hat. LucidWorks implemented these integration points using Drools 5.3.0. The integration should allow engines to alter a variety of things, such as search results, facets, sorting and filtering criteria, and the documents themselves, both at index time and query time.

Topics covered in this section:

Many users will not need to know all the details of the topics covered in this section, but they will show up in their configuration and it may be helpful to be able to customize how the rules work on queries or documents as they are indexed.

More information on the LucidWorks integration with Drools is available in the section on Drools Integration, which provides information on writing rules files themselves.

Search Integration

Search integration consists of two main functionalities:

1. `RulesComponent`: The ability to modify incoming parameters for searching, faceting, sorting, etc., as well as to modify the intermediate results.

2. `LandingPageComponent`: The ability to bypass the rest of the processing and return a landing page based on some matching criteria.

During processing, the following objects are added as Facts:

- The Schema
- The Query
- Any Filters
- The results
- The sorting specification
- The grouping specification
- Any facets

See the `FactCollector.java` class for more details on facts (in the section on Drools Integration).

Applying Rules at Query Time (RulesComponent)

The primary mechanism for applying rules at query time (i.e., any request to the system that uses a `SearchHandler` and is not a document indexing request) is via a Solr `SearchComponent` called `RulesComponent`. The `RulesComponent` can be configured to occur anywhere in the SearchComponent, but it is typically best to configure it to be the first item in the chain after the filter by role component, since it is often the case that you want rules to make decisions based on the applications input parameters (such as the query, sort, etc.) and you want the rules to make changes before they get processed by the other components. For instance, you may have a rule that fires when the user query is equal to "title:dogs" and you want the rule to change the query to be "title:dogs AND category:pets". By configuring the component first in the chain, you will be able to change the query before it is parsed, thus saving extra rule writing involving re-arranging complex Query objects.

Configuration

`SearchComponents` are configured in the `solrconfig.xml` file for each collection (found in `$LWE_HOME/conf/solr/cores/collection/conf/` where `collection` is the name of the specific collection). The default configuration for the `RulesComponent` is:

```
<searchComponent class="com.lucid.rules.RulesComponent" name="firstRulesComp">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">first</str>
    <!-- The handle can be used to turn on or off explicit rules components in the
     case when you have multiple rules at different stages of the component ordering-->
    <str name="handle">first</str>
  </searchComponent>
  <searchComponent class="com.lucid.rules.RulesComponent" name="lastRulesComp">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">last</str>
    <str name="handle">last</str>
  </searchComponent>
```

This can then be hooked into the `RequestHandler` using the `first-components`, `components` or `last-components` capability.

Input Parameters

There is a fair amount of control around exactly when rules will be fired.

Parameter	Type	Description	Default	Example
rules	boolean	Turn on or off the RulesComponent	false	`&rules=false`
rules.<handle name>	boolean	Turn on or off a specific RulesComponent instance using the handle name	true	`&rules.first=false`
rules.prepare	boolean	Turn off rule processing as part of the prepare phase	true	`&rules.prepare=false`
rules.process	boolean	Turn off rule processing as part of the process phase	true	`&rules.process=false`
rules.finishStage	boolean	Turn off rule processing as part of the finishStage phase	true	`&rules.finishStage=false`

The system does not currently allow you to turn off individual phases of an instance (unless it is the only instance that is configured). In other words, if two `RulesComponent`-s are configured, it is not possible to turn off the process phase of only one.

Landing Pages (LandingPageComponent)

The `LandingPageComponent` is very similar the `RulesComponent`, with the main difference being that it only implements the `prepare()` method on the `SearchComponent` and it invokes the `RulesEngine.findLandingPage()` method. The `LandingPageComponent` does not turn off other components in the chain, but it should be possible for the rules engine to do so. Placing the landing page in the output is also the responsibility of the rule writer. In essence, all the component does is guarantee that it is called as part of the `prepare()` phase and that the rules used can be configured separately from the other rules used.

Configuration

The `LandingPageComponent` is also configured in the `solrconfig.xml` file for each collection. The default LucidWorks integration includes this section:

```
<searchComponent class="com.lucid.rules.LandingPageComponent" name="landingPage">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">first</str>
    <!-- The handle can be used to turn on or off explicit rules components in the
     case when you have multiple rules at different stages of the component ordering
     -->
    <str name="handle">landing</str>
  </searchComponent>
```

Input Parameters

Like the `RulesComponent`, the `LandingPageComponent` has several parameters. One thing to note is that the `LandingPageComponent` is only executed in the prepare phase of rules execution so the other available parameters will likely not be required for your implementation.

Parameter	Type	Description	Default	Example
landing	boolean	Turn on or off the LandingPageComponent	false	`&landing=false`
landing.<handle name>	boolean	Turn on or off a specific LandingPageComponent instance using the handle name	true	`&landing.first=false`
landing.prepare	boolean	Turn off rule processing as part of the prepare phase	true	`&landing.prepare` =false
landing.process	boolean	Turn off rule processing as part of the process phase	true	`&landing.process` =false
landing.finishStage	boolean	Turn off rule processing as part of the finishStage phase	true	`&landing.finishStage` =false

Altering Documents

Just as it is possible to alter things like queries and sort parameters, it is also possible to alter documents, both at index time and at query time, detailed in the following sections.

Index time (RulesUpdateProcessor)

The `RulesUpdateProcessor(Factory)` is a Solr `UpdateRequestProcessorFactory` that can be configured to be a part of the update processor chain and used to alter documents as they are about to be indexed. Like any `UpdateRequestProcessorFactory`, it has access to the `AddUpdateCommand` and the associated docs, which can then be exposed to the `RulesEngine` via the `prepareDocument()` method.

By default, the `RulesUpdateProcessor` is configured in the `lucid-update-chain` and can be enabled/disabled by passing in the name of the handle, prefixed by `rules.`. For instance, if the Processor has a handle of `docProc`, then `&rules.docProc=false` would disable the processor and processing would continue down the chain. Rule processing is on by default.

⚠ In v2.1, rules processing may introduce performance lags which may be negligible or significant, depending on your implementation. While rules processing is enabled by default, if you don't intend to use rules with your LucidWorks implementation, it should be disabled. See the section on Disabling Business Rules for more information.

During processing, the following objects are added as Facts:

- The AddUpdateCommand
- The Schema
- The SolrInputDocument

See the `FactCollector.java` class for more details on facts (in the section on Drools Integration).

Configuration
Here is the default configuration for the `lucid-update-chain` in the `solrconfig.xml` file for each collection:

```
<updateRequestProcessorChain name="lucid-update-chain">
      <processor class="com.lucid.update.CommitWithinUpdateProcessorFactory" />
      <processor class="com.lucid.update.FieldMappingUpdateProcessorFactory" />
             <processor class="com.lucid.rules.RulesUpdateProcessorFactory">
                  <str name="requestHandler">/rulesMgr</str>
                  <!-- we re-use the engine, but we could have an independent one-->
                  <str name="engine">docs</str>
                  <!-- Each one should have it's own handle, as you can have multiple
in the chain -->
                  <str name="handle">docProc</str>
             </processor>
      <processor class="com.lucid.update.DistributedUpdateProcessorFactory">
    <!-- example configuration... "shards should be in the *same* order for
   every server in a cluster. Only "self" should change to represent what server
   *this* is. <str name="self">localhost:8983/solr</str> <arr name="shards">
   <str>localhost:8983/solr</str> <str>localhost:7574/solr</str> </arr> -->
      </processor>
      <processor class="solr.LogUpdateProcessorFactory">
  <int name="maxNumToLog">10</int>
      </processor>
      <processor class="solr.DistributedUpdateProcessorFactory" />
      <processor class="solr.RunUpdateProcessorFactory" />
 </updateRequestProcessorChain>
```

To disable rules processing, you can either remove or comment out the section that defines the `com.lucid.rules.RulesUpdateProcessorFactory` parameters.

Query time (RulesDocTransformer)

In Solr, a `DocTransformer` is invoked as part of response writing to allow applications to inject and/or modify the fields on a document before being returned to the client. For instance, it can be used to include a new value for a field calculated by a rule or to do things like alter a price for a specific user.

During processing, the following objects are added as Facts:

- The Schema
- The SolrDocument
- The Lucene internal Doc Id

See the `FactCollector.java` class for more details on facts (in the section on Drools Integration)..

Altering a field will not cause an item to be resorted

If, for example, you are sorting by price and you change one of the document's prices, this will not cause a resort. If you want to do that, we suggest you use Solr's Sort by Function capability.

The RulesEngineManagerHandler

The `RulesEngineManagerHandler` is the Solr `RequestHandler` that holds onto references to the various RuleEngine instances specified in the solr configuration. The manager maintains a map of engines to their names. Most components are setup to take in the name of this `RequestHandler` and then go ask it for the engine by name.

Sample Configuration

```
<requestHandler class="com.lucid.rules.RulesEngineManagerHandler" name="/rulesMgr">
    <!--
     Engines can be shared, but they don't have to be.  A SearchComponent or other
consumer can
     specify the engine they want by name.
     -->
    <lst name="engines">
      <lst name="engine">
        <str name="name">first</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultFirst.drl</str>
        </lst>
        <!-- The fact collector defines what facts get injected into the rules engines
working memory -->
        <!--<lst name="factCollector">
          <str name="class">com.lucid.rules.drools.FactCollector</str>
        </lst>-->
      </lst>
      <!-- Engine is using rules that are designed to be called after all other
components -->
      <lst name="engine">
        <str name="name">last</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultLast.drl</str>
        </lst>
      </lst>
      <lst name="engine">
        <str name="name">landing</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultLanding.drl</str>
        </lst>
      </lst>

      <lst name="engine">
        <str name="name">docs</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultDocs.drl</str>
        </lst>
        <lst name="factCollector">
          <str name="class">com.lucid.rules.drools.FactCollector</str>
        </lst>
      </lst>
    </lst>
  </requestHandler>
```

Drools Integration

Drools is an open source, Apache licensed, business rules engine primarily maintained by Red Hat. The LucidWorks Search integration uses Drools v5.3.0.

Drools comes with a Rule Editor (named `guvnor`) as well as libraries for business process management, complex event processing and, of course, the rules engine itself. It's rules language is Java-like, but it can be extended by implementing a Domain Specific Language (DSL). Drools provides two versions of its Rules Engine: a stateless engine and a state-maintaining engine. The state-maintaining engine is expected to live for longer periods of time while the stateless engine is best used on

a per request basis. LucidWorks Search currently only
uses the stateless engine.

An example rule in our Drools implementation looks like:

```
import org.apache.solr.handler.component.ResponseBuilder;
rule "HelloWorld"
  no-loop
  when
    $rb : ResponseBuilder();
  then
    $rb.rsp.add("hello", "world");
end
```

This rule says when there is a `ResponseBuilder` object in the working memory (the instance of this is called a fact in the working memory) then add the key/value pair "hello", "world" to the Solr response. We'll show more examples later.

StatelessDroolsRulesEngine

The `StatelessDroolsRulesEngine` is the only current implementation of the `RulesEngine` class. As the name implies, it is stateless, meaning that rules processing is handled on a per-request basis and no state is kept between the firing of rules. Drools does offer a state-based version, but we have not implemented it at this time. Note, the RulesEngineManagerRequestHandler is set up to instantiate engines on core reloads. Engine instantiation is when the rules files are loaded, which means that if you make changes to your rules file, then you need to reload the core. In LucidWorks, this is done by restarting the system.

Configuration

Configuration consists of specifying a few key elements: the `FactCollector`, one or more rules files (named `*.drl`) and the name of the engine. For example, recalling the default LucidWorks configuration from the previous section, the `solrconfig.xml` for each collection could have a definition like this:

```
<lst name="engine">
        <str name="name">first</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultFirstDocs.drl</str>
        </lst>
        <!-- The fact collector defines what facts get injected into the rules engines
working memory -->
        <!--<lst name="factCollector">
          <str name="class">com.lucid.rules.drools.FactCollector</str>
        </lst>-->
      </lst>
```

The "rules" attribute consists of one or more rules files. These files must be accessible via the `SolrResourceLoader.openResource()` method (i.e., via the classpath). Likewise, the factCollector must also be available via the SolrResourceLoader. In the default example, the files are available in the 'rules' directory, which is found under `$LWE_HOME/conf/solr/cores/collection/conf/rules`, where `collection` is the particular collection to which the rules will be applied.

When using the `RulesComponent`, the `LandingPageComponent` or the `RulesDocTransformerFactory`, rule writers may rely on the fact that the name of the "handler" (specified in the configuration) will be available as part of the Request context (`request.getContext().get("rulesHandle")`) along with the phase the component is in (landing, prepare or process – `getContext().get("rulesPhase")`) such that rules can be written that target a specific handler and/or a specific phase.

Fact Collection

Fact collection is the process of taking "facts" from Solr's request, state, etc., and injecting them into Drools' working memory. By inserting a fact into the working memory, the rules engine algorithm can determine whether or not a rule should fire. Thus, fact collection is a key part of the rules evaluation process. Fact collection is specified by the `FactCollector` class or a derivative. LucidWorks comes with two `FactCollector` implementations, each described below. Implementers may wish to provide their own if they wish to inject other facts into the working memory or present existing facts in different ways.

FactCollector

The default `FactCollector` injects a number of Solr objects into the working memory for consumption by the rules engine. The `FactCollector` class provides three callback mechanisms to the RulesEngine where facts can be added:

- `public void addFacts(ResponseBuilder rb, Collection facts)`: Called at query time (`RulesComponent`) and injects a number of `ResponseBuilder` items into the working memory to make rule writing easier.
- `public void addFacts(AddUpdateCommand cmd, Collection facts)`: Called when adding a document at indexing time (`RulesUpdateProcessor`)
- `public void addFacts(SolrDocument doc, int docId, Collection facts)`: Called when transforming a document at query time (`RulesDocTransformer`)

StatsFactCollector

The `StatsFactCollector` extends the `FactCollector` and injects Solr and/or LucidWorks MBeans into the working memory, thus making it possible to write rules based on the state of the system. For instance, one could potentially only add new facet parameters if the system load was below some threshold. See Integrating Monitoring Services for more information about available MBeans for Solr and LucidWorks.

DroolsHelper Class

The `DroolsHelper` class contains a number of methods that can be invoked by rules writers to help with common tasks and simplify the "then" part of the rule. For instance, there is a method that can take in a query and a boost and set the boost value. There are also methods for helping merge separate facet requests together (such as a field facet with a facet query). For instance, it has methods that evaluate what phase the engine is in and returns true or false if it matches an expected value. This can be useful if you want rules to fire only during certain phases of the `SearchComponent` process (i.e. prepare, process, etc.). To see this in action, notice the use of the `hasPhaseMatch()` method in the example rules section.

The `DroolsHelper.class` file may not be in your distribution of LucidWorks. For that reason, we've provided the text of the code below.

```
package com.lucid.rules.drools;

public class DroolsHelper extends java.lang.Object
{
    /* Fields */
    private static transient org.slf4j.Logger log;
    public final static java.lang.String RULES_PHASE = "rulesPhase";
    public final static java.lang.String RULES_HANDLE = "rulesHandle";

    /* Constructors */
    public DroolsHelper() {
    }

    /* Methods */
    public static boolean
hasPhaseMatch(org.apache.solr.handler.component.ResponseBuilder, java.lang.String) {
    }

    public static boolean
hasPhaseMatch(org.apache.solr.handler.component.ResponseBuilder, java.lang.String,
java.lang.String) {
    }

    public static boolean
hasHandlerNameMatch(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String) {
    }

    public static void boostQuery(org.apache.lucene.search.Query, float) {
    }

    public static void addToResponse(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String, java.lang.Object) {
    }

    public static void addToResponse(org.apache.solr.common.util.NamedList,
java.lang.String, java.lang.Object) {
    }

    public static void mergeFacets(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String, int, java.lang.String[]) {
    }

    public static void addFacet(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String, java.lang.String, int, int) {
    }

    public static void modRequest(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String, java.lang.String[]) {
    }

    public static void modRequest(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String, int) {
    }

    public static void modRequest(org.apache.solr.handler.component.ResponseBuilder,
java.lang.String, boolean) {
    }

    public static boolean contains(java.lang.String, java.lang.String) {
    }

    public static java.util.Collection analyze(org.apache.solr.schema.IndexSchema,
java.lang.String, java.lang.String) throws java.io.IOException {
    }

}
```

Limitations

Since the implementation is stateless, there is obviously no way to write rules that go across requests without implementing your own RulesEngine.

Example Rules and Recipes

The following examples and recipes are meant to help you implement Drools business rules processing.

Example Rules

LucidWorks internal tests uses the following rules for testing purposes, which you may find instructive:

```
package rules;

import org.apache.solr.handler.component.ResponseBuilder;
import function com.lucid.rules.drools.DroolsHelper.*;
import org.apache.solr.common.SolrDocument;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.TermQuery;
import org.apache.solr.search.DocListAndSet;
import org.apache.solr.common.params.SolrParams;
import org.apache.solr.common.params.ModifiableSolrParams;
import org.apache.solr.common.util.NamedList;

import org.slf4j.Logger;

global org.slf4j.Logger logger;

rule "HelloWorld"
no-loop
when
$rb : ResponseBuilder();
then
$rb.rsp.add("hello", "world");
end

#when a user searches for title:"cool foxes", change the query to be title:foxes
cat_s:cool
rule "cool foxes"
no-loop
when
$rb: ResponseBuilder($qStr : req.params.get("q") == "title:\"cool foxes\"");
then
#$rb.rsp.add("origQuery", $qStr);
addToResponse($rb, "origQuery", $qStr);
#$rb.rsp.add("modQuery", "title:foxes +cat_s:cool");
addToResponse($rb, "modQuery", "title:foxes +cat_s:cool");
#((ModifiableSolrParams)$rb.req.getParams()).set("q", );
modRequest($rb, "q", "title:foxes +cat_s:cool");
end

# Part one of two steps to add/modify facets
rule "Add Facet Pre"
no-loop
when
$rb: ResponseBuilder($facet : req.params.get("facet.field") == "cat_s");
then
addToResponse($rb, "modFacet", "cat_s");
#if we want the same number of results back, then we need to alter the limit
modRequest($rb, "facet.limit", $rb.req.getParams().getInt("facet.limit", 10) - 1);
modRequest($rb, "facet.query", "val_s:AAA");
```

```
end

rule "Change Sort"
no-loop
when
$rb: ResponseBuilder($facet : req.params.get("sort") == "sort_i");
then
modRequest($rb, "sort", "sort_f desc");
end

rule "Landing Page"
no-loop
when
$rb: ResponseBuilder($qStr : req.params.get("q") == "title:foxes");
then
addToResponse((NamedList)$rb.rsp.getValues().get("responseHeader"), "landingPage",
"http://www.fox.com");
end

rule "FirstPrepare"
no-loop
when
$rb : ResponseBuilder();
eval(hasPhaseMatch($rb, "prepare", "first"));
then
$rb.rsp.add("first", "prepare");
end

rule "FirstProcess"
no-loop
when
$rb : ResponseBuilder();
#Only fire this rule in the "first" engine's process phase
eval(hasPhaseMatch($rb, "process", "first"));
then
$rb.rsp.add("first", "process");
end

rule "FinishStage"
no-loop
when
$rb : ResponseBuilder();
eval(hasPhaseMatch($rb, "finishStage", "first"));
then
$rb.rsp.add("first", "finish");
end
```

Recipes

Below are a number of common recipes that may be useful for Drools rules writers based on common scenarios where a rules engine may be needed.

Altering a Query

A common rules scenario is to somehow modify or extend a user's input query with other needs and/or capabilities. For instance, one might modify a query based on some information in the user's profile, such as gender, previous purchases, or location.

As a simple example, consider the case where one has a title field and a category field in their content and you want to filter by category when certain keywords are encountered. This could be written as:

```
rule "cool foxes"
  no-loop
  when
    $rb: ResponseBuilder($qStr : req.params.get("q") == "title:\"cool foxes\"");
  then
    addToResponse($rb, "origQuery", $qStr);
    addToResponse($rb, "modQuery", "title:foxes +cat_s:cool");
    modRequest($rb, "q", "title:foxes +cat_s:cool");
end
```

In this example, if the user's query is title:"cool foxes" we do three things:

1. Add the original query in to the response
2. Add the new modified query into the response
3. Actually change the query (the "q" param) to now be "title:foxes +cat_s:cool" where cat_s is the name of our category field.

Facet Modification

In some cases, there is a need to make sure a specific facet, regardless of sorting criteria (like count or alphabetical order) appears in a set of facet results. For instance, if Nike is a preferred brand on a shoe website, we may want the `manufacturer:Nike` facet to show up for all manufacturer facet requests when they have at least 1 item. For this case, the best way to handle this is to add a facet by query option to the request as part of the prepare phase and then merge it into the facets after the facets have been calculated. In order to do this, we need to have two RulesComponents configured, one that runs before the facet component and one that runs after it.

Our RequestHandler configuration could look like:

```
<arr name="components">
 <str>filterbyrole</str>
 <str>landingPage</str>
 <str>firstRulesComp</str>
 <str>query</str>
 <str>mlt</str>
 <str>stats</str>
 <str>feedback</str>
 <!-- Note: highlight needs to be after feedback -->
 <str>highlight</str>
 <!-- Note: facet also needs to be after feedback -->
 <str>facet</str>
 <str>spellcheck</str>
 <str>lastRulesComp</str>
 <str>debug</str>
</arr>
```

In this case, the `firstRulesComp` and the `lastRulesComp` are the pertinent RulesComponents. Then, we need to have two rules, one in the rules files for the first component and one for the last:

First

```
rule "Add Facet Pre"
  no-loop
  when
    $rb: ResponseBuilder($facet : req.params.get("facet.field") == "cat_s");
  then
    addToResponse($rb, "modFacet", "cat_s");
    #if we want the same number of results back, then we need to alter the limit
    modRequest($rb, "facet.limit", $rb.req.getParams().getInt("facet.limit", 10) - 1);
    modRequest($rb, "facet.query", "val_s:AAA");
end
```

Here we detect that we are faceting against the category field, so we alter the facet limit to save room for the new facet and then we also inject a specific facet query. Then, in the last component, we do the merge:

```
rule "Add Facet Post"
  no-loop
  when
    $rb: ResponseBuilder($facet : req.params.get("facet.field") == "cat_s");
    ResponseBuilder(rsp.values.get("facet_counts") != null);
  then
    addToResponse($rb, "facetMerged", "cat_s");
    mergeFacets($rb, "cat_s", 0, "val_s:AAA");
end
```

The `mergeFacets` call is a DroolsHelper invocation that does the appropriate list manipulation, inserting the counts from the facet query created in the first step into position 0 of the category facet. Note, in this case, we aren't checking that the count is at least one, but that could be added.

Disabling Business Rules

As noted earlier, business rules are enabled by default. However, that may introduce unnecessary performance lags or create errors with traditional index replication. To disable business rules, you must remove or comment out references to rules in the `solrconfig.xml` file for each collection.

It would be possible to remove these rules parameters from the default `solrconfig.xml` file and create a template for future collection creation. To learn more about this, see the section on Collection Templates.

When removing business rules from the `solrconfig.xml` file, LucidWorks will need to be either stopped while making the changes, or restarted once the changes are made.

These are the steps to disabling business rules:

- Remove Rules from Update Chain
- Remove Rules from the Standard Request Handler
- Remove the Rules Request Handler
- Remove Rules Search Components
- Remove the RulesDocTransformer
- Remove Rules From the Replication Handler

Remove Rules from Update Chain

Comment out the section that defines the Rules Update Processor (`<processor class="com.lucid.rules.RulesUpdateProcessorFactory">` until the closing `</processor>` tag).

In most cases, this is sufficient to disable business rules. However, the next sections will assist you in fully removing business rules from your implementation.

```
<updateRequestProcessorChain name="lucid-update-chain">
  <processor class="com.lucid.update.CommitWithinUpdateProcessorFactory"/>
        <processor class="com.lucid.rules.RulesUpdateProcessorFactory">
          <str name="requestHandler">/rulesMgr</str>
          <!-- we re-use the engine, but we could have an independent one-->
          <str name="engine">docs</str>
          <!-- Each one should have it's own handle, as you can have multiple in the
chain -->
          <str name="handle">docProc</str>
          </processor>
  <processor class="com.lucid.update.DistributedUpdateProcessorFactory">
   <!-- example configuration... "shards should be in the *same* order for
    every server in a cluster. Only "self" should change to represent what server
    *this* is. <str name="self">localhost:8983/solr</str> <arr name="shards">
    <str>localhost:8983/solr</str> <str>localhost:7574/solr</str> </arr> -->
  </processor>
  <processor class="solr.LogUpdateProcessorFactory">
   <int name="maxNumToLog">10</int>
  </processor>
  <processor class="solr.DistributedUpdateProcessorFactory"/>
  <processor class="com.lucid.update.FieldMappingUpdateProcessorFactory"/>
  <processor class="solr.RunUpdateProcessorFactory"/>
 </updateRequestProcessorChain>
```

The specific section to remove is:

```
<processor class="com.lucid.rules.RulesUpdateProcessorFactory">
          <str name="requestHandler">/rulesMgr</str>
          <!-- we re-use the engine, but we could have an independent one-->
          <str name="engine">docs</str>
          <!-- Each one should have it's own handle, as you can have multiple in the
chain -->
          <str name="handle">docProc</str>
          </processor>
```

Remove Rules from the Standard Request Handler

Find the section as below that defines the `/lucid` request handler, and remove the lines for `landingPage` and `firstRulesComp` and `lastRulesComp`.

```
<requestHandler class="solr.StandardRequestHandler" name="/lucid">

  <arr name="components">
   <str>filterbyrole</str>
                        <str>landingPage</str>
                        <str>firstRulesComp</str>
   <str>query</str>
   <str>mlt</str>
   <str>stats</str>
   <str>feedback</str>
   <!-- Note: highlight needs to be after feedback -->
   <str>highlight</str>
   <!-- Note: facet also needs to be after feedback -->
   <str>facet</str>
   <str>spellcheck</str>
                         <str>lastRulesComp</str>
   <str>debug</str>
  </arr>
```

Remove the Rules Request Handler

The rules request handler defines the `RuleEngine` instances and the rules files. The entire section copied below can be removed or commented out.

```
<requestHandler class="com.lucid.rules.RulesEngineManagerHandler" name="/rulesMgr">
    <!--
     Engines can be shared, but they don't have to be.  A SearchComponent or other
consumer can
     specify the engine they want by name.
     -->
    <lst name="engines">
      <lst name="engine">
        <str name="name">first</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultFirst.drl</str>
        </lst>
        <!-- The fact collector defines what facts get injected into the rules engines
working memory -->
        <!--<lst name="factCollector">
          <str name="class">com.lucid.rules.drools.FactCollector</str>
        </lst>-->
      </lst>

      <lst name="engine">
        <str name="name">landing</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultLanding.drl</str>
        </lst>
      </lst>

      <!-- Engine is using rules that are designed to be called after all other
components -->
      <lst name="engine">
        <str name="name">last</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultLast.drl</str>
        </lst>
      </lst>
      <lst name="engine">
        <str name="name">docs</str>
        <str
name="class">com.lucid.rules.drools.stateless.StatelessDroolsRulesEngine</str>
        <lst name="rules">
            <str name="file">rules/defaultDocs.drl</str>
        </lst>
      </lst>
    </lst>
  </requestHandler>
```

Remove Rules Search Components

The search components allow the rules to make changes to queries, based on the rules defined. The entire sections shown below can be removed or commented out.

```
<searchComponent class="com.lucid.rules.LandingPageComponent" name="landingPage">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">landing</str>
    <!-- The handle can be used to turn on or off explicit rules components in the
     case when you have multiple rules at different stages of the component ordering
     -->
    <str name="handle">landing</str>
  </searchComponent>
  <searchComponent class="com.lucid.rules.RulesComponent" name="firstRulesComp">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">first</str>
    <!-- The handle can be used to turn on or off explicit rules components in the
     case when you have multiple rules at different stages of the component ordering-->
    <str name="handle">first</str>
  </searchComponent>
  <searchComponent class="com.lucid.rules.RulesComponent" name="lastRulesComp">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">last</str>
    <str name="handle">last</str>
  </searchComponent>
```

Remove the RulesDocTransformer

The `RulesDocTransformer` allows business rules to inject or modify fields in a document before returning them to a client.

```
<transformer class="com.lucid.rules.RulesDocTransformerFactory" name="rules">
    <str name="requestHandler">/rulesMgr</str>
    <str name="engine">first</str>
  </transformer>
```

Remove Rules From the Replication Handler

If using Index Replication, remove the rules-related files from the list of `conf` files to replicate between servers. In this section:

```
<requestHandler class="solr.ReplicationHandler" name="/replication">
  <lst name="master">
  <str name="replicateAfter">commit</str>
  <str
name="confFiles">admin-extra.html,admin-extra.menu-bottom.html,admin-extra.menu-top.html,e
</lst>
 </requestHandler>
```

Specifically, remove `rules/defaultFirst.drl`, `rules/defaultLast.drl`, `rules/defaultLanding.drl`, and `rules/defaultDocs.drl`.

Security and User Management

Generally, enterprise-level application designers must take into account four main security considerations for any search application:

- Network access to the various components of the service
- Authentication of users
- Authorization to use various parts of the user interface
- Authorization to view certain documents

LucidWorks Search implements security for each of these as follows:

Network access: Because the components of LucidWorks (LWE-Core and LWE-UI) run on different ports, an administrator can easily secure individual components at the network level by restricting access to the port in question. For example, if only the Admin and Search UI services need to be accessible outside the production network, an administrator can leave those ports open while blocking LWE-Core. The chapter Securing LucidWorks describes this process in more detail. Note that if you are using the LucidWorks Search document authorization features this step is particularly important, as direct access to the underlying Solr application can circumvent these measures.

In addition, you may want to ensure that the components use SSL for communication or that users access the Admin UI via HTTPS. The chapter Enabling SSL describes how to do that in more detail.

User authentication: LucidWorks supports LDAP binding for user authentication, so an administrator can create roles or groups on an external LDAP server, then use them to control access to UI functionality or sets of documents. The chapter LDAP Integration describes how to configure LDAP for LucidWorks.

UI authorization: LucidWorks controls access to the Admin UI and the Search UI. The chapter LDAP Integration discusses how to configure these access levels in order to give different LDAP users and groups authorization to use these different functions.

Document authorization: LucidWorks allows the administrator to configure document filters for different roles. These document filters then limit what documents appear in search results for users in those roles. For example, the administrator can create a filter that enables users in the *finance* role to see only documents that satisfy a query of *department:finance*. You can create these filters with the Search Filters screen of the Admin UI. LucidWorks also enables the creation of document-based filtering, in which only the owner (or owners) of a document are able to see it. The section Restricting Access to Content describes how to set up your documents to support this functionality.

Securing LucidWorks

Restricting Access

LucidWorks Search consists of three components: LWE-Core, LWE-UI, and LWE-Connectors.

Because it provides access to the REST API, direct access to the LWE-Core component provides access to all of Solr's capabilities, including adding and removing documents, retrieving stored field values for all documents, and additional LucidWorks-enhanced capabilities such as job scheduling and system status. The LWE-Core component should only be directly HTTP accessible to other components that need access to Solr or REST API interfaces. If you are using a single server installation and don't want to expose Solr or REST API interfaces via the network then you might want to restrict access to LWE-Core to localhost only. You can do that by adding the socket connector's `host` attribute for the Jetty container.

You can also restrict direct access to LucidWorks components by IP address, or by fronting it with an authenticating firewall. For a production implementation, consider restricting access to the component HTTP ports to only those required by the application, just as one would do with a typical relational database. If you are using the built-in search filters or document-level authentication, you **must** prevent access to LucidWorks by any process other than your application in order to prevent circumvention of these features.

Implementing SSL

Some of the components can be implemented with SSL. See the chapter Enabling SSL for more details.

Restricting Access to LucidWorks Search User Interfaces

LucidWorks has two built-in authorizations to control user access:

- ADMIN allows users to access any part of the LucidWorks UI.
- SEARCH limits users to only the built-in end user search interface.

You can restrict a user's access to specific parts of the application by mapping manually created or LDAP-supplied usernames and/or LDAP-supplied groups to appropriate authorization on the User screen in the Admin UI.

Hiding Documents by Restricting Access

The privileges of the LucidWorks process and the rights that process has to access documents for indexing are crucial to its proper operation. Generally, you want LucidWorks Search to be able to access all documents within a particular folder or from a particular web site. The built-in LucidWorks Search crawlers will index any specified document, as long as the LucidWorks process has permissions to do so. After a document has been indexed, all stored fields are accessible through the Solr interface.

That said, documents can be excluded from indexing by leveraging operating system, file, and web-level security capabilities; if the process doesn't have access, it will not index the content.

Some data sources, such as those configured to crawl content in a database or in SMB, SharePoint, S3 or S3H servers, credentials need to be supplied for the crawler to access the system. Those credentials determine what documents the crawler has access to.

Enabling SSL

This functionality is **not available** with LucidWorks Search on AWS or Azure

Secure Socket Layer (SSL) encryption can be easily enabled in the LucidWorks Search system.

- Steps to Enable SSL
- Certificate Management
- Configuring Mutually Authenticated SSL
- Debugging SSL Configuration
- Related Topics

Steps to Enable SSL

In the steps below, note that LucidWorks Search components run under Jetty, but have separate configuration files. Each component needs to be enabled separately, although the process for each component is the same. For more information about configuring Jetty to use SSL, see also the Jetty documentation on Configuring SSL.

Step 1: Modify master.conf

If you have already installed LucidWorks Search, you can set these values by modifying the `master.conf` file found in `$LWE_HOME/conf/`:

```
# COMPONENT LWE-Core - LWE-Solr + LWE REST API.
# -----
lwecore.enabled=true
lwecore.address=http://127.0.0.1:8888
  ...
# COMPONENT LWE-Connectors.
# -----
lweconnectors.enabled=true
lweconnectors.address=http://127.0.0.1:8765
  ...
# COMPONENT LWE-UI - Admin and Search UI as well as Alerts
# -----
lweui.enabled=true
lweui.address=http://127.0.0.1:8989
```

Alternately, each component could be set to `https://` and the desired port during the installation process.

Step 2: Modify jetty.xml for LWE-Core Component

The `jetty.xml` file found in `$LWE_HOME/conf/jetty/lwe-core/etc` needs to be modified to comment out the non-SSL connector. In the file, find the following section and add comment markers at the beginning and at the end (`<!--` and `-->`, respectively):

```
<Call name="addConnector">
      <Arg>
          <New class="org.eclipse.jetty.server.bio.SocketConnector">
            <Set name="port"><SystemProperty name="jetty.port" default="8888"/></Set>
            <Set name="maxIdleTime">50000</Set>
            <Set name="lowResourceMaxIdleTime">1500</Set>
          </New>
      </Arg>
    </Call>
```

Step 3: Modify jetty-ssl.xml for LWE-Core Component

In the directory `$LWE_HOME/conf/jetty/lwe-core/etc` the file `jetty-ssl.xml` should be edited to activate the sample configuration. The configuration is currently commented out, but the comment tags should be removed and the `port`, `keyStore`, `keyStorePassword`, `keyManagerPassword`, `trustStore` and `trustStorePassword` properties should be configured.

```
<Configure id="Server" class="org.eclipse.jetty.server.Server">
<!--
    <Call name="addConnector">
     <Arg>
       <New class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
         <Arg>
           <New class="org.eclipse.jetty.http.ssl.SslContextFactory">
             <Set name="keyStore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-ui/etc/keystore</Set>
             <Set name="keyStorePassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
             <Set name="keyManagerPassword">OBF:1u2u1wml1z7s1z7a1wnl1u2g</Set>
             <Set name="trustStore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-ui/etc/truststore</Set>
             <Set name="trustStorePassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
           </New>
         </Arg>
         <Set name="port">8443</Set>
         <Set name="maxIdleTime">30000</Set>
       </New>
     </Arg>
   </Call>
-->
</Configure>
```

The `keyStore` and `trustStore` files must be located in the locations specified so they can be found on Jetty startup. They can be located anywhere on the server, as long as the correct locations are defined in the `jetty-ssl.xml` file.

SSL Ports

The sample `jetty-ssl.xml` files for the LWE-Core and LWE-UI components use the same example port (8443). Note, however, that these ports must be different from each other for LucidWorks to operate properly.

Step 4: Modify jetty.xml for LWE-UI Component

The `jetty.xml` file found in `$LWE_HOME/conf/jetty/lwe-ui/etc` needs to be modified to comment out the non-SSL connector. In the file, find the following section and add comment markers at the beginning and at the end (`<!--` and `-->`, respectively):

```
<Call name="addConnector">
      <Arg>
          <New class="org.eclipse.jetty.server.bio.SocketConnector">
            <Set name="port"><SystemProperty name="jetty.port" default="8888"/></Set>
            <Set name="maxIdleTime">50000</Set>
            <Set name="lowResourceMaxIdleTime">1500</Set>
          </New>
      </Arg>
    </Call>
```

Step 5: Modify jetty-ssl.xml for LWE-Core Component

In the directory `$LWE_HOME/conf/jetty/lwe-ui/etc` the file `jetty-ssl.xml` should be edited to activate the sample configuration. The configuration is currently commented out, but the comment tags should be removed and the `port`, `keyStore`, `keyStorePassword`, `keyManagerPassword`, `trustStore` and `trustStorePassword` parameters should be configured.

```
<Configure id="Server" class="org.eclipse.jetty.server.Server">
<!--
    <Call name="addConnector">
     <Arg>
       <New class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
         <Arg>
           <New class="org.eclipse.jetty.http.ssl.SslContextFactory">
             <Set name="keyStore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-ui/etc/keystore</Set>
             <Set name="keyStorePassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
             <Set name="keyManagerPassword">OBF:1u2u1wml1z7s1z7a1wnl1u2g</Set>
             <Set name="trustStore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-ui/etc/truststore</Set>
             <Set name="trustStorePassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
           </New>
         </Arg>
         <Set name="port">8443</Set>
         <Set name="maxIdleTime">30000</Set>
       </New>
     </Arg>
   </Call>
-->
</Configure>
```

The `keyStore` and `trustStore` files must be located in the locations specified so they can be found on Jetty startup. They can be located anywhere on the server, as long as the correct locations are defined in the `jetty-ssl.xml` file.

SSL Ports

The sample `jetty-ssl.xml` files for the LWE-Core and LWE-UI components use the same example port (8443). Note, however, that these ports must be different from each other for LucidWorks to operate properly.

Step 6: Restart LucidWorks

After verifying that the `keyStore` and `trustStore` files are in the locations specified in the `jetty-ssl.xml` files, LucidWorks Search must be restarted for the changes to take effect.

Certificate Management

LucidWorks uses standard java jks format in keystores and truststores. Those stores can be managed using the standard Java keytool.

Currently all certificates are managed outside of LucidWorks. There are no certificate management tools or admin displays for configuring SSL certificate related settings. All configuration tasks need to be made manually after installing LucidWorks and potentially repeated on all nodes where LucidWorks is running.

Configuring Mutually Authenticated SSL

LucidWorks supports securing communications to the core APIs with Mutual SSL authentication. This means the REST API and Solr API can be protected so that only clients that you trust can access these APIs. The system can also use mutually authenticated SSL internally to communicate to each Solr node when using distributed search.

The LucidWorks portions of SSL functionality can be configured by using the SSL Configuration API.

When configuring LucidWorks to use mutually authenticated SSL the container must also be configured to require certificates for authentication. In Jetty this is done by using `<property name="needClientAuth" value="true" />` in the related SSLConnector.

Debugging SSL Configuration

Reviewing logging events from the LucidWorks log files (either `core.YYYY_MM_DD.log` or `ui.YYYY_MM_DD.log`) may provide some hints about what is going on if SSL is not working as expected.

Common SSL Problems

Symptom: `javax.net.ssl.SSLHandshakeException: null cert chain`

Cause: Client is not sending client certificate. Reconfigure client so that it sends a client certificate with the request.

Symptom: `javax.net.ssl.SSLException: Unrecognized SSL message, plaintext connection?`

Cause: Client is connecting to SSL endpoint without using SSL.

The cURL command line tool can be used to verify the SSL configuration. For example,:

```
curl --cacert <ca.crt> --key <host.key> --cert <client.crt>
https://localhost:8443/dashboard
```

The link in this example is to the main LucidWorks Admin UI dashboard. Since this requires authentication, you should see the HTML indicating you will be redirected to the login page. If that's what you see, then SSL is properly set up.

Related Topics

- Jetty doc on configuring SSL
- Java keytool

Restricting Access to Content

LucidWorks Search provides three ways to restrict access to content through based on user identity:

- Search Filters
- Access Control Lists
- Document-based Authorization

Information for LucidWorks Search in the Cloud Users
Some sections following refer to editing the `solrconfig.xml` file, which is not possible for LucidWorks Search customers hosted in AWS or Azure.

Search Filters

Search filters provide the ability to limit the visibility of content only to specific users or user groups. For example, users in the finance role might be limited only to documents that satisfy the query `department:finance`. The LucidWorks Search Admin UI allows the creation of search filters that can be appended to all user queries. Usernames (manually created or supplied by the LDAP system) and/or groups (supplied by the LDAP system) can be mapped to search filters with the Search Filters page. You can also configure manual or LDAP search filters using the Roles API.

By default, LucidWorks comes configured with a default filter called "DEFAULT" that allows users to see all results for any query. This filter is defined in `solrconfig.xml`, and could be modified if needed:

```
<searchComponent class="com.lucid.handler.RoleBasedFilterComponent"
name="filterbyrole">
   <!-- Solr filter query that will be applied for users without group/role info -->
   <str name="default.filter">-*:*</str>
   <!-- Solr filter queries for roles, one role may have multiple filter queries.
 name is the role, value is the part of the filterquery that is to be formed. -->
   <lst name="filters">
      <str name="DEFAULT">*:*</str>
   </lst>
</searchComponent>
```

Note that this has defined that the default filter is `-*:*`. What this means is that someone without the DEFAULT role should see no results. However, since queries in LucidWorks Search are handled by the `/lucid` request handler, we have configured that handler to process searches for users without a role as though they had the DEFAULT role. This is in a later section of `solrconfig.xml`, where defaults are defined for the `/lucid` request handler (the below is truncated):

```
<lst name="defaults">
...
      <str name="role">DEFAULT</str>
...
    </lst>
```

Access Control Lists

LucidWorks also supports access control lists (ACL) on Windows Share (SMB) and SharePoint data sources. ACL uses Windows Active Directory to control document access on a per-user basis. ACL filtering is configured for each data source, allowing you to have different authorizations depending on the definitions in each repository. To use this functionality, set up a Windows Share or SharePoint data source and configure the requisite fields.

If you do not need to configure ACL filtering on a per-data source basis, you can use the Filtering API to configure a Search Handler to perform the same functionality. Note that this is only supported for a Windows Share data source type. The Filtering API will configure the search handler when configured in `solrconfig.xml` will look like this:

```
<searchComponent class="com.lucid.security.AclBasedFilterComponent" name="adfiltering">
  <str name="provider.class">com.lucid.security.ad.ADACLTagProvider</str>
  <str name="filterer.class">com.lucid.security.WindowsACLQueryFilterer</str>
  <lst name="provider.config">
    <str name="java.naming.provider.url">ldap://127.0.0.1</str>
    <str name="java.naming.security.principal">admin</str>
    <str name="java.naming.security.credentials">admin</str>
  </lst>
  <lst name="filterer.config">
    <str name="should_clause">*:* -data_source_type:smb</str>
  </lst>
</searchComponent>
```

In certain circumstances, you may need to add a `userFilter` or `groupFilter` parameter to the search component to properly implement your ACL filter.

Once created, the search component must be added to the `/lucid` request handler with the Search Handler Components API.

Document-based Authorization

An application can enforce document visibility controls in front of LucidWorks simply by adding fields to each document that represent usernames, group membership, or other types of flags that help match a user with the content they are allowed to see in results. Generally these types of fields would be of type "string", possibly multi-valued. This technique is best suited to content extracted from a database or custom data source. The file and web crawling capabilities in LucidWorks do not index any security related attributes (though the file path itself may be useful for application-level restrictions).

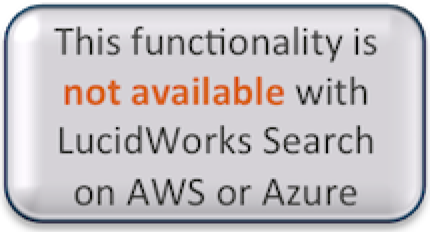
For example, documents could be indexed with an "owner" field. Here's a Solr XML file for this example:

```
<add>
  <doc>
    <field name="id">1</field>
    <field name="text">Bob's Document - For his eyes only\!</field>
    <field name="owner">bob</field>
  </doc>
  <doc>
    <field name="id">2</field>
    <field name="text">Jill's Document - Only she should find this</field>
    <field name="owner">jill</field>
  </doc>
</add>
```

Related Topics

- Windows Shares Data Sources
- SharePoint Data Sources
- Filtering Results

LDAP Integration


This functionality is **not available** with LucidWorks Search on AWS or Azure

LucidWorks Search supports integrating user authentication with an existing LDAP system.

Two LDAP features are currently supported:

1. Authentication and Authorization of users (prerequisite for any other LDAP functionality)
2. User-to-group mapping (optional)

LDAP and built-in (API-based) user authentication are mutually exclusive. If LDAP is enabled, built-in authentication is not, and the reverse.

For standard LDAP integration, the LDAP administrative user only needs permissions to query the LDAP server for users and groups. We recommend that you create an LDAP admin user with only the necessary minimal user and group querying permissions for use with LucidWorks.

LucidWorks also allows you to authenticate users without LDAP administrative credentials. This method is called "queryless" authentication, because LucidWorks does not query the LDAP directory for user information. Rather, LucidWorks uses the attribute value plus the user's login and the base suffix as the user's DN. This method only works if the exact location of your LDAP user data is known and is the same for all relevant users. Another limitation of queryless authentication is that LucidWorks cannot find members of a group, only individual users.

It is also possible, using standard Java SSL functionality, to use certificate authentication with a SSL-enabled LDAP server. More information on that is available here: http://docs.oracle.com/javase/jndi/tutorial/ldap/security/ssl.html.

For information about filtering search results based on LDAP permissions, see Restricting Access to Content and Search Filters.

Enabling LDAP

These steps need to be completed to successfully enable LDAP. Each step is required and should be done in this order:

1. Configure the LDAP Configuration File with the instructions below.

2. Map at least one user to have admin permissions using the LDAP section of the Settings page . Because the built-in authentication is disabled when LDAP authentication is enabled, you cannot map a user or group to the Admin authorization after LDAP is enabled. If no one has Admin authorization, no one will be able to access the Administration User Interface. So, before enabling LDAP, go to the System Settings page and map an LDAP username or a group to "Admin UI" by adding it to the Group or User section of the Admin UI definition.
3. Enable LDAP by setting the environment variable `lweui.ldap.enabled` to **true** in the `master.conf` file found in `$LWE_HOME/conf/`.
4. Restart LucidWorks.

LDAP Configuration File

The main configuration file for configuring LDAP is `ldap.yml`, found in the `$LWE_HOME/conf/` directory. The default settings must be modified as needed for LucidWorks to connect to the LDAP server and query the database for user authentication. If LDAP is already enabled and this file is editing, you will need to restart the server for changes to take effect.

Below is the main section of the `ldap.yml` configuration file that needs to be edited. Note that the file also includes sample configurations for standard LDAP authentication, queryless authentication, and Microsoft ActiveDirectory integration for use with Windows Shares data sources.

Lines Must Be Indented

When customizing the `ldap.yml` file, keep in mind that the attributes must be indented at least two spaces. So, when removing the hash mark (#), do not remove the extra spaces. All lines must also be indented the same number of spaces (so, if some lines are indented three spaces, then all lines must be indented three spaces).

```
########################################################
# Warning: Always restart the application after adjusting
#   your LDAP config, or unpredictable behavior may result.
########################################################
# production:
#   host: localhost
#   port: 389 # 636 for SSL
#   attribute: uid
#   base: dc=xyz,dc=corp,dc=com
#   # user_query: '$ATTR=$LOGIN' # default query is '$ATTR=$LOGIN', set this if you
need something more complex
#   admin_user: cn=Manager,dc=xyz,dc=corp,dc=com  # If you don't have an admin
password, you can disable
#   admin_password: secret                        # admin login in the UI "Settings"
page
#   ssl: false
#   group_base: ou=groups,dc=xyz,dc=corp,dc=com
#   group_membership_attribute: uniqueMember
#   group_name_attribute: cn
#   # group_query: '(&(objectclass=groupOfUniqueNames)($ATTR=$USER))' # default query
is '$ATTR=$USER' where $USER is user's DN
```

The attribute definitions included in the `ldap.yml` file are as follows:

Attribute	**Definition**
host	The hostname of the LDAP server that contains the user information.
port	The port to use while connecting to the LDAP server that contains the user information.
attribute	The attribute of the user object that the system will use to search for the user, or assume when constructing an explicit DN via query-less authentication.
base	Search base for user queries, or suffix appended to attribute + login for queryless authentication.
user_query	Optional: supplies an arbitrarily complex query if the default user query is not sufficient. Variable substitutions are as follows: $ATTR will be substituted with the value of 'attribute' from above; $LOGIN will be substituted with the value the user entered in the login form in the UI. Search is performed using 'base' as a search base.
admin_user	Administrative login to use for searching the directory. Not used for queryless authentication.

admin_password	Administrative password to use for searching the directory. Not used for queryless authentication.
ssl	Enable/disable SSL.
group_base	Search base for group queries. Not used with queryless authentication.
group_membership_attribute	The attribute to look for in the group object that will contain members' user DNs.
group_name_attribute	The attribute of the group object that the system will use to search for the group.
group_query	Optional: supplies an arbitrarily complex query if the default group query is not sufficient. Variable substitutions are as follows: `$ATTR` will be substituted with the value of 'group_name_attribute'; `$USER` will be substituted with the logged-in user's fully-qualified LDAP DN. Search is performed using 'group_base' as a search base. The default query `($ATTR=$USER)` does not specify the object type for groups. Several different group object types are common, such as group, groupOfNames, groupOfUniqueNames, and so on. Therefore, non-group objects may also match if they contain a matching attribute.

User to Group Mappings

LucidWorks supports mapping users to groups with the `group_membership_attribute` setting. This allows LucidWorks to do an additional query while the user is logging in to find all the groups the user is a member of.

Manual User Management

LucidWorks also includes a REST API that allows creation and authentication of users. Using this API and the Perl Examples provided with the application, users can be created, passwords reset, and accounts deleted. As mentioned previously, API-based user management and LDAP authentication are mutually exclusive: you can only use one user management method.

Related Topics

- Restricting Access to Content
- Search Filters

Solr Direct Access

LucidWorks Search is Solr-powered at its core. Solr, an Apache Software Foundation project, provides an easy-to-use HTTP interface above and beyond Lucene, a very fast and scalable Java search engine library. Both Solr and Lucene are entirely open source, available under the Apache Software License.

LucidWorks Search exposes the Solr interface directly. This means that applications can leverage both Solr's power and openness and LucidWorks Search's ease of use.

This guide covers Solr when LucidWorks and Solr intersect but it does not provide an extensive overview of the inner workings of Solr, and in places assumes some basic knowledge of Solr. For a good introduction to Solr, LucidWorks has produced an Apache Solr Reference Guide which provides a lot of information about how Solr works "under the hood".

Solr Version

For information about the Solr version included in this release of LucidWorks, see the SOLR_VERSION.txt file in `$LWE_HOME/app/SOLR_VERSION.txt`. For LucidWorks v2.5, we have included Solr version 4.0 (the official release) and the following patches (descriptions are either from the Solr commits or the Solr Jira issues):

- SOLR-3861: Refactor SolrCoreState so it's managed by SolrCore.
- SOLR-3897: Preserve multi-value fields during hit highlighting.
- SOLR-3932: SolrCmdDistributorTest either takes 3 seconds or 3 minutes.
- SOLR-3933: Distributed commits are not guaranteed to be ordered within a request.
- SOLR-3938: PrepareCommit command omits commitData causing a failure to trigger replication to slaves.
- SOLR-3939: An empty or just replicated index cannot become the leader of a shard after a leader goes down.
- SOLR-3961: Fixed error with LimitTokenCountFilterFactory.
- SOLR-3971: A collection that is created with numShards=1 turns into a numShards=2 collection after starting up a second core and not specifying numShards.
- SOLR-3981: Fixed bug that resulted in document boosts being compounded in <copyField/> destination fields.
- SOLR-3988: Fixed SolrTestCaseJ4.adoc(SolrInputDocument) to respect field and document boosts.
- SOLR-3992: QuerySenderListener doesn't populate document cache.
- SOLR-3994: Create more extensive tests around unloading cores.
- SOLR-3995: Recovery may never finish on SolrCore shutdown if the last reference to a SolrCore is closed by the recovery process.
- SOLR-3998: Atomic update on uniqueKey field itself causes duplicate document.
- SOLR-4005: If CoreContainer fails to register a created core, it should close it.
- SOLR-4009: OverseerCollectionProcessor is not resilient to many error conditions and can stop running on errors.

You can also get detailed Solr version information for all releases of LucidWorks Search from our public Github fork here: https://github.com/lucidimagination/lucene-solr. To see information for a specific LucidWorks version, select the tag for that version from the **Switch Tags** drop-down list. Please note, however, that this is not a stand-alone, runnable Lucene or Solr release; it is intended as a source reference only.

How the LucidWorks-Bundled Solr is Different

The primary difference between using Solr and LucidWorks is the base URL. Solr's example application is accessed by default at `http://localhost:8983/solr/`, whereas the LucidWorks default collection instance of Solr is rooted at `http://localhost:8888/solr/collection1/`. If using multiple collections, replace `collection1` with the correct collection name. The Solr URL for each collection is displayed under each collection listing on the main Collections page in the Admin UI.

In addition, some of the examples that are usually included with Solr are not included with LucidWorks. This includes detailed examples and explanations that are provided in the `schema.xml` and `solrconfig.xml` files. Those examples will likely still work with LucidWorks, but would need to be inserted manually into those files.

Other differences are mentioned specifically in sections that discuss certain features. If a limitation with Solr is not mentioned, it can be assumed that the Solr functionality works as you would expect with a stand-alone Solr instance.

Related Topics

- Apache Solr Reference Guide
- Apache Solr project homepage
- Apache Solr Wiki

Performance Tips

A number of configuration items can be manipulated for better performance when benchmarking LucidWorks. Implementing some of these optimizations may require directly configuring Solr via `schema.xml` and `solrconfig.xml`. See the Solr documentation at http://wiki.apache.org/solr/ for more details on Solr customizations that may be right for your implementation.

- Ensure that you are running the JVM in server mode.
- Allocate only as much memory as needed to the JVM heap. The rest should be left free to allow the operating system to cache as much of the Lucene index files as possible.

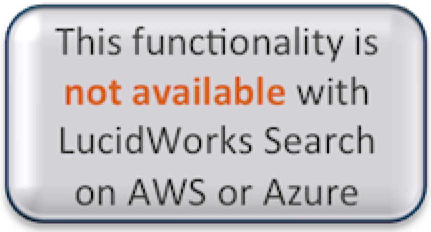
Improving indexing speed

- Minimize indexing the same content in more than one field. Each field should be either indexed on its own or Solr's copyField functionality can be used to copy it to an indexed catch-all field.
- Avoid storing the same content more than once. The target field of copyField commands should almost never be stored.
- Avoid commits during the indexing process. Turn off Solr auto-commit and avoid explicitly committing until indexing has completed.
- Disable rules processing if not using business rules as part of your implementation. See the section on Disabling Business Rules for details on how to disable rules processing.

Improving Search speed

- Perform a variety of searches before starting any timings. This warms up the server JVM, and causes parts of the index, commonly used sort fields and filters to be cached by the operating system.
- Search in as few fields as possible. A single indexed catch-all text field containing the contents of all the other searchable fields (generated by **copyField** commands) will be faster to search than a multi-field query across many indexed fields.
- If necessary, turn off relevancy enhancers such as proximity phrase queries, date recency boosts, and synonym expansion to generate benchmarks for comparison with later tests when those features are re-enabled.
- Retrieve the minimum number of stored fields that still provide a optimal search experience for users.
- Only retrieve the number of documents that are immediately necessary. The **start** and **rows** query arguments may be used to request pages of results.
- Disable rules processing if not using business rules as part of your implementation. See the section on Disabling Business Rules for details on how to disable rules processing.
- For a large index (on *NIX), force key parts of the indexed portion into operating system cache by changing to the index directory and executing `cat *.prx *.frq *.tis > /dev/null`
- Review the section on Wildcards at Start of Terms if leading wildcards have been enabled for important performance considerations.

Expanding Capacity


This functionality is **not available** with LucidWorks Search on AWS or Azure

As your search application grows, you may need to scale the system to add space for indexes or to increase query responsiveness. This section discusses advanced deployment options to enhance system performance and ensure seamless application scaling.

With Solr 4, which is included with LucidWorks Search, the best way to scale is in SolrCloud mode. How to start LucidWorks in SolrCloud mode is discussed in the section Using SolrCloud in LucidWorks.

If you only need to extend your index across multiple servers **Index Replication** shows how to configure multiple shards for a master-slave environment. Or you can use **Distributed Search and Indexing** to distribute search and indexing processes across multiple servers or shards for peak performance. Note, however, that distributed search and replication are no longer in active development by the Solr community.

Using SolrCloud in LucidWorks

This functionality is **not available** with LucidWorks Search on AWS or Azure

SolrCloud is a set of Solr features that expands the capabilities of Solr's distributed search, enabling and simplifying the creation and use of Solr clusters. SolrCloud is still under active development, but already supports the following features:

- Central configuration for the entire cluster
- Automatic load balancing and fail-over for queries
- Zookeeper integration for cluster coordination and configuration

Topics discussed in this section:

The Apache Solr Reference Guide includes an extensive section on SolrCloud, which includes background information and configuration instructions. Some changes have been made for LucidWorks Search, however, which are described below.

LucidWorks Search implements SolrCloud as a purely Solr feature; as with other features in LucidWorks that are inherited from Solr, it can be directly manipulated with the Solr examples and instructions, keeping in mind the caveats and modifications detailed below, particularly around bootstrapping ZooKeeper and the cluster nodes.

Enabling SolrCloud Mode

The standard instructions for starting SolrCloud are modified slightly for LucidWorks Search. While much of the SolrCloud documentation in the Apache Solr Reference Guide and in the Solr Wiki can be used, it is important to only start LucidWorks Search in SolrCloud mode with the instructions included here.

Installing LucidWorks Search

To start LucidWorks Search in SolrCloud mode, you need to bootstrap the configuration and start ZooKeeper. To show how to do this, we'll use a very simple two-node cluster as an example.

Because we need two servers for this example, we will make two installations of LucidWorks, one on the server "`example`" and the other on the server "`example2`". During installation, do not start LucidWorks Search. Instead, start them manually, as shown below.

We recommend that you only install LucidWorks using the installer application; copying the `LucidWorksSearch` directory to another directory to create another server may cause conflicts with ports. Information on installing LucidWorks is available in the section on Installation.

The installation in `example` should use port 8983 for the LWE-Core component, which is selected during the installation process. The installation on `example2` should use the default port (8888) for the LWE-Core component. If enabling other components, be sure to modify the ports for each installation as well. If new to LucidWorks, see the section on Working With LucidWorks Search Components. Your port selections might look like this:

Component	`example` **Ports**	`example2` **Ports**
LWE-Core	8983	8888
LWE-Connectors	8965	8765
LWE-UI	8889	8989

ZooKeeper will run on the LWE-Core port + 1000, so in this scenario we expect ZooKeeper to run on port 9983. It's important to keep that in mind while planning the installation ports so there isn't an inadvertent conflict with LucidWorks Search ports.

SolrCloud uses ZooKeeper to manage nodes, and it's worth taking a look at the ZooKeeper website to understand how ZooKeeper works before configuring SolrCloud. Solr can embed ZooKeeper, but for a production use, it's recommended to run a ZooKeeper ensemble, as described in the ZooKeeper section of the SolrCloud wiki page.

Starting LucidWorks Search

Start `example`:

```
$LWE/app/bin/start.sh -lwe_core_java_opts "-Dbootstrap_conf=true -DzkRun -DnumShards=2"
```

This will define `example` as the leader node. We use the usual LucidWorks start script, but pass some Java options to it. The `bootstrap_conf` allows copying of the configuration files for each collection to the nodes, while `zkRun` starts ZooKeeper. The `numShards` value defines how many nodes there will be in the cluster. Be sure to set this accurately, as Solr cannot yet easily increase the number of shards without re-bootstrapping the cluster.

We only need to pass `bootstrap_conf` and `numShards` the first time LucidWorks is started in SolrCloud mode. In subsequent LucidWorks restarts, start this leader node with `./start.sh -lwe_core_java_opts "-DzkRun"`. The `-DzkRun` could be added to `master.conf`, in which case the `start.sh` script alone would start ZooKeeper each time.

For each node of the cluster, we use a slightly different command. This would start `example2`:

```
$LWE/app/bin/start.sh -lwe_core_java_opts "-DzkHost=localhost:9983"
```

Note that the port defined as the `zkHost` is the port of the LWE-Core component + 1000. So, if LWE-Core was defined at port 8983, ZooKeeper would be started at port 9983.

The above instructions assume a Linux-based operating system. For Windows-based systems, use `start.bat` as in these examples:

Start `example`:

```
$LWE\app\bin\start.bat -lwe_core_java_opts "-Dbootstrap_conf=true -DzkRun
-DnumShards=2"
```

Start `example2`:

```
$LWE\app\bin\start.bat -lwe_core_java_opts "-DzkHost=localhost:9983"
```

Bootstrapping Solr vs. LucidWorks Search

This table outlines the differences between the Solr instructions for bootstrapping SolrCloud mode and the LucidWorks Search instructions. It is meant as a summary if you are already familiar with how SolrCloud works.

SolrCloud	LucidWorks Search
Use `start.jar`	Use `start.sh` or `start.bat` with `-lwe_core_java_opts` defined
Use `bootstrap_confdir` to upload configuration files to ZooKeeper	`bootstrap_conf=true`
Use `collection.configName`	Not needed with `bootstrap_conf=true`
Default configuration directory is `./solr/conf`	Default configuration directory is `$LWE_HOME/conf/solr/cores/collection1_0/conf`

How SolrCloud Works with LucidWorks

There are some caveats to using SolrCloud with LucidWorks Search, as it is so far only partially integrated with the system. Future releases of LucidWorks Search will contain more tight integration points with SolrCloud functionality.

When running LucidWorks Search in SolrCloud mode, some LucidWorks Search-specific features are not yet fault tolerant and highly available. While the index and configuration files are fully SolrCloud supported, the following are not currently replicated across shards:

- Data sources and their related metadata (such as crawl history)
- The LucidWorks user database, which stores manually created users (such as the default "admin" user)
- User alerts
- LDAP configuration files
- SSL configuration

Even though these features aren't replicated, they can still be used with LucidWorks Search in SolrCloud mode. The files that hold this metadata are in the `$LWE_HOME/conf` folder and could be copied to the other nodes in the cluster to act as backup if the main node goes down for any length of time. This is a manual process and not yet automated by LucidWorks Search.

To accommodate for this limitation, we recommend that you do a full LucidWorks Search installation (i.e., all components) on every machine in your cluster. You should then choose one node to use for the Admin UI. This is the node that will store your data sources and associated metadata. Another node can be chosen as the node that does crawling, or you can use the same node used by the Admin UI. Document updates will still be sent to the nodes, via the index update processes that make up SolrCloud functionality. If the node used for the Admin UI goes down, you can choose another node to act as the Admin UI node, but unless the related configuration files have been copied to that node you will not have the same user accounts and data sources in the other nodes. Once you bring the node originally used for the Admin UI back, it should still have your data sources and other LucidWorks-specific metadata.

The following LucidWorks features may encounter significant problems when working in SolrCloud mode:

- Click Scoring cannot be used in SolrCloud mode at this time.
- Auto-complete-related suggestions should be pulled from a single index node if auto-complete is enabled; distributed auto-complete indexing is not available.
- De-duplication does not work in SolrCloud due to a bug in Solr (SOLR-3473).

When creating a new collection (with either the Admin UI or the API), and you are working in SolrCloud mode, you can specify the number of shards to break it up into. This number, however, cannot be higher than the number of shards defined when LucidWorks Search was bootstrapped.

Related Topics

- Getting Started with SolrCloud
- SolrCloud Wiki page

Index Replication

This functionality is **not available** with LucidWorks Search on AWS or Azure

As of Solr 4.0, SolrCloud is the preferred way to distribute indexes for redundancy, failover, and improved performance. Index Replication and Distributed Search are considered obsolete technologies; while still supported, they are not in active development. See the section on Using SolrCloud in LucidWorks for more information on using SolrCloud with LucidWorks Search.

Index Replication distributes complete copies of a master index to one or more slave servers. The master server continues to manage updates to the index. All querying is handled by the slaves. This division of labor enables Solr to scale to provide adequate responsiveness to queries against large search volumes. The master server's index is replicated on the slaves, which then process requests such as queries.

LucidWorks Search supports index replication, but it is not configured through the Admin UI. Instead, replication configuration requires editing XML configuration files in the Solr release included with LucidWorks Search. This section explains how replication works and how to edit the configuration files. Detailed examples are provided, so even if you're new to XML and Solr configuration, you should be able to set up and configure master/slave replication servers with ease.

When the Click Scoring Relevance Framework is enabled, LucidWorks ensures that also the click boost data is replicated together with index files. See the section on Click Scoring Tools and Index Replication for more information.

Configuring Replication on the Master Server

To set up replication, you will need to edit the `solrconfig.xml` file on the master server. To edit the file, you can use an XML editor or even a simpler tool such as Notepad on a PC or TextEdit on a Mac.

Within the `solrconfig.xml` file, you will edit the definition for a Request Handler. A Request Handler is a Solr process that responds to requests. In this case, you will be configuring the Replication RequestHandler, which processes requests specific to replication.

The example below shows how to configure the Replication RequestHandler on a master server.

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
 <lst name="master">
  <!-- Replicate on 'optimize'. Other values can be 'commit', 'startup'.
       It is possible to have multiple entries of this config string -->
  <str name="replicateAfter">optimize</str>
  <!-- Create a backup after 'optimize'. Other values can be 'commit', 'startup'.
       It is possible to have multiple entries of this config string.
       Note that this is just for backup, replication does not require this.
   -->
  <!-- <str name="backupAfter">optimize</str> -->
  <!-- If configuration files need to be replicated give the names here,
       separated by comma -->
  <str name="confFiles">schema.xml,stopwords.txt,elevate.xml</str>
  <!-- The default value of reservation is 10 secs. See the documentation
      below. Normally, you should not need to specify this -->
  <str name="commitReserveDuration">00:00:10</str>
 </lst>
</requestHandler>
```

Operations that Trigger Replication

The value of the `replicateAfter` parameter in the ReplicationHandler configuration determines which types of events should trigger the creation of snapshots for use in replication.

The `replicateAfter` parameter can accept multiple arguments.

`replicateAfter` Setting	**Description**
`startup`	Triggers replication whenever the master index starts up.
`commit`	Triggers replication whenever a commit is performed on the master index.
`optimize`	Triggers replication whenever the master index is optimized.

If you are using `startup` setting for `replicateAfter`, you'll also need a `commit` or `optimize` if you want to trigger replication on future commits/optimizes as well. If only the `startup` option is given, replication will not be triggered on subsequent commits/optimizes after it is done for the first time at the start.

Configuring Replication on Slave Servers

The code below shows how to configure a ReplicationHandler on a slave server.

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
 <lst name="slave">
  <!-- fully qualified url for the replication handler of master.
       It is possible to pass on this as a request param for the
       fetchindex command
    -->
  <str
name="masterUrl">http://master.solr.company.com:8983/solr/corename/replication</str>
  <!-- Interval in which the slave should poll master. Format is HH:mm:ss.
       If this is absent slave does not poll automatically.
       But a fetchindex can be triggered from the admin or the http API
    -->
  <str name="pollInterval">00:00:20</str>
  <!-- THE FOLLOWING PARAMETERS ARE USUALLY NOT REQUIRED -->
  <!-- To use compression while transferring the index files.
       The possible values are internal|external
       if the value is 'external' make sure that your master Solr
       has the settings to honor the accept-encoding header.
       see here for details http://wiki.apache.org/solr/SolrHttpCompression
       If it is 'internal' everything will be taken care of automatically.

       USE THIS ONLY IF YOUR BANDWIDTH IS LOW.
       THIS CAN ACTUALLY SLOW DOWN REPLICATION IN A LAN -->
  <str name="compression">internal</str>
  <!-- The following values are used when the slave connects to the
       master to download the index files.
       Default values implicitly set as 5000ms and 10000ms respectively.
       The user DOES NOT need to specify these unless the bandwidth
       is extremely low or if there is an extremely high latency
    -->
  <str name="httpConnTimeout">5000</str>
  <str name="httpReadTimeout">10000</str>
  <!-- If HTTP Basic authentication is enabled on the master,
       then the slave can be configured with the following -->
  <str name="httpBasicAuthUser">username</str>
  <str name="httpBasicAuthPassword">password</str>
 </lst>
</requestHandler>
```

The master server is unaware of the slaves. Each slave server continuously polls the master (depending on the `pollInterval` parameter) to check the current index version of the master. If the slave finds out that the master has a newer version of the index it initiates a replication process. The steps are as follows:

1. The slave issues a filelist command to get the list of the files. This command returns the names of the files as well as some metadata (e.g., size, a lastmodified timestamp, an alias if any).

2. The slave checks with its own index if it has any of those files in the local index. It then runs the filecontent command to download the missing files. This uses a custom format (akin to the HTTP chunked encoding) to download the full content or a part of each file. If the connection breaks in between, the download resumes from the point it failed. At any point, the slave tries 5 times before giving up a replication altogether.
3. The files are downloaded into a temp directory, so that if either the slave or the master crashes during the download process, no files will be corrupted. Instead, the replication process will simply abort.
4. After the download completes, all the new files are 'mv'ed to the live index directory, and the file's timestamp is set to be identifical to the file's counterpart on the master master.
5. A commit command is issued on the slave by the Slave's ReplicationHandler, and the new index is loaded.

Configuring Replication on a Repeater Server

A master may be able to serve only so many slaves without affecting performance. Some organizations have deployed slave servers across multiple data centers. If each slave downloads the index from a remote data center, the resulting download may consume too much network bandwidth. To avoid performance degradation in cases like this, you can configure one or more slaves as repeaters. A repeater is simply a node that acts as both a master and a slave. To configure a server as a repeater, the definition of the Replication requestHandler in the `solrconfig.xml` file must include file lists of use for both masters and slaves. Be sure to set the replicateAfter parameter to commit, even if replicateAfter is set to optimize on the main master. This is because on a repeater (or any slave), a commit is called only after the index is downloaded. The optimize command is never called on slaves. Optionally, one can configure the repeater to fetch compressed files from the master through the compression parameter to reduce the index download time.

Here's an example of a ReplicationHandler configuration for a repeater:

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
 <lst name="master">
  <str name="replicateAfter">commit</str>
  <str name="confFiles">schema.xml,stopwords.txt,synonyms.txt</str>
 </lst>
 <lst name="slave">
  <str
name="masterUrl">http://master.solr.company.com:8983/solr/corename/replication</str>
  <str name="pollInterval">00:00:60</str>
 </lst>
</requestHandler>
```

Replicating Configuration Files

To replicate configuration files, list them with the `confFiles` parameter in the master's configuration. Only files found in the conf directory of the master's Solr instance will be replicated.

Solr replicates configuration files only when the index itself is replicated. Even if a configuration file is changed on the master, that file will be replicated only after there is a new commit/optimize on master's index.

As a precaution when replicating configuration files, Solr copies configuration files to a temporary directory before moving them into their ultimate location in the conf directory. The old configuration files are then renamed and kept in the same `conf/` directory. The ReplicationHandler does not automatically clean up these old files.

Unlike the index files, where the timestamp is good enough to figure out if they are identical, configuration files are compared against their checksum. If a replication involved downloading at least one configuration file with a modified checksum, the ReplicationHandler issues a core-reload command instead of a commit command.

Replicating the solrconfig.xml File

To keep the configuration of the master servers and slave servers in sync, you can configure the replication process to copy configuration files from the master server to the slave servers. In the `solrconfig.xml` on the master server, include a `confFiles` value like the following:

```
<str name="confFiles">solrconfig_slave.xml:solrconfig.xml,x.xml,y.xml</str>
```

This ensures that the local configuration `solrconfig_slave.xml` will be saved as `solrconfig.xml` on the slave. All other files will be saved with their original names. On the master server, the file name of the slave configuration file can be anything, as long as the name is correctly identified in the `confFiles` string; then it will be saved as whatever file name appears after the colon ':'.

Related Topics

- Using SolrCloud in LucidWorks
- Scaling and Distribution chapter from LucidWorks' Apache Solr Reference Guide

Distributed Search and Indexing

This functionality is **not available** with LucidWorks Search on AWS or Azure

As of Solr 4.0, SolrCloud is the preferred way to distribute indexes for redundancy, failover, and improved performance. Index Replication and Distributed Search are considered obsolete technologies; while still supported, they are not in active development. See the section on Using SolrCloud in LucidWorks for more information on using SolrCloud with LucidWorks Search.

Consider using distributed search when an index becomes too large to fit on a single system, or when a single query takes too long to execute. Distributed search can reduce the latency of a query by splitting the index into multiple shards and querying across all shards in parallel, merging the results.

Distributed search should not be used if queries to a single index are fast enough but one simply wishes to expand the capacity (queries per second) of the system. In this case, standard Index Replication should be used.

Distributed Indexing

To utilize distributed search, the index must be split into shards across multiple servers. Each shard is a LucidWorks Search server containing a complete index that can be queried independently, but which only contains a fraction of the complete search collection.

If using distributed indexing with a Solr XML data source type, you may encounter a situation where the crawl never ends without a restart of LucidWorks. This is due to a problem in the distributed index processor and the way Solr XML files are crawled by LucidWorks.

There are two possible solutions to this problem:

1. Use SolrCloud. The distributed indexing is handled automatically by ZooKeeper, and provides automatic failover in case of server failure.
2. Disable the `DistributedUpdateProcessor` on all but the primary, master, node. It is not really required to be running on slave nodes since LucidWorks crawlers send their files through only one node during processing.

Manual Distributed Indexing

One method of splitting the search collection into multiple shards is to index some documents to each shard instead of sending all documents to a single shard. Updates to a document should always be sent to the same shard, and documents should not be duplicated on different shards.

Manual Configuration

A Distributed Update Processor can be enabled to automatically support distributed indexing by sending update requests to multiple servers (shards).

Enabling distributed indexing is done via the `solrconfig.xml` file, found in `$LWE_HOME/solr/cores/collection/conf` (replace `collection` with the name of the collection that is being configured for distributed indexing). By default it is not enabled. The `solrconfig.xml` file needs to be installed on each shard, and the shards should be listed in the same order in each file.

The distributed update processor is controlled by two parameters, `shards` and `self`, which may either be specified in `solrconfig.xml`, or supplied with a specific update request to Solr.

- `shards` lists the servers in the cluster. The list should be exactly the same (that is, in the same order) in the configuration file for every server in the cluster.
- `self` should be different for each server in the cluster and should match the entry in `shards` for the particular server. It is used to allow updates for the particular server to be directly added rather than going through the HTTP interface. If it is missing, distributed update will still work, but will be less efficient.

To start using distributed indexing, find the following section in `solrconfig.xml`, and uncomment the shard location definitions. Below is an example of shard definition that is not commented out.

```
<updateRequestProcessorChain name="lucid-update-chain">
  <processor class="com.lucid.update.DistributedUpdateProcessorFactory">
    <!-- example configuration...
     "shards should be in the *same* order for every server
      in a cluster.  Only "self" should change to represent
      what server *this* is. -->

    <str name="self">localhost:8983/solr</str>
    <arr name="shards">
      <str>localhost:8983/solr</str>
      <str>localhost:7574/solr</str>
    </arr>
  </processor>
  <processor class="solr.LogUpdateProcessorFactory">
    <int name="maxNumToLog">10</int>
  </processor>
  <processor class="com.lucid.update.FieldMappingUpdateProcessorFactory"/>
  <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
```

Indexing Documents

If distributed indexing has been configured as above, then any indexing initiated from the LucidWorks Search administration user interface, such as crawling directories, will be appropriately handled by sending some documents to each server. One can use the distributed update processor in conjunction with any update handler while directly updating Solr. The `/update/xml` and `/update/csv` update handlers are already configured to use `distrib`, the distributed update processor, by default.

If an update handler has not been configured to use the distributed update processor, it may be specified in the URL via the `update.processor` parameter:

```
http://localhost:8888/solr/collection1/update?update.processor=distrib
```

If the `self` and `shards` parameters are not configured in solrconfig.xml, then they may be specified as arguments on the update url.

```
http://localhost:8888/solr/collection1/update?update.processor=distrib&self=localhost:8888
```

Update commands may be sent to any server with distributed indexing configured correctly. Document adds and deletes are forwarded to the appropriate server/shard based on a hash of the unique document id. **commit** commands and **deleteByQuery** commands are sent to every server in `shards`.

Distributed Search

After a logical index is split across multiple shards, distributed search is used to make requests to all shards, merging the results to make it appear as if it came from a single server.

Programmatic Distributed Search

One can use distributed search with Solr request handlers such as `standard`, `dismax`, or `lucid` (the handler used by the LucidWorks Search), or any other search handler based on `org.apache.solr.handler.component.SearchHandler`.

Supported Components

The following Solr components currently support distributed searching:

- The Query component that returns documents matching a query
- The Facet component, for `facet.query` and `facet.field` requests where `facet.sorted=true` (the default: return the constraints with the highest counts)
- The Highlighting component, which highlights results
- The Debug component

The presence of the `shards` parameter in a request will cause that request to be distributed across all shards in the list. The syntax of `shards` is `host1:port1/base_url1,host2:port2/base_url2,...`

The example below would query across 3 different shards, combining the results:

```
http://localhost:8888/solr/collection1/select?shards=localhost:8983/solr,localhost:7574/sp
```

As a convenience to clients, a new request handler could be created with `shards` set as a default like any other ordinary parameter.

⚠ The `shards` parameter should not be set as a default in the standard request handler as this could cause infinite recursion.

Scalability and Fault Tolerance

To provide fault tolerance and increased scalability, standard replication can be used to provide multiple identical copies of each index shard. Each shard would have a master and multiple slaves.

Indexing in a Fault Tolerant Distributed Configuration

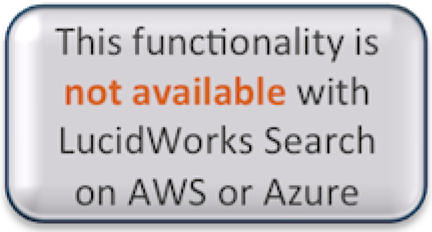
Only the master for each shard should be configured in distributed indexing or specified to the distributed update processor. There is no fault tolerance while indexing - if the master for a shard goes down, indexing should be suspended.

Searching in a Fault Tolerant Distributed Configuration

Each shard will have multiple replicas. A Virtual IP (VIP) should be configured in the load balancer for each shard, consisting of all replicas. LucidWorks Search distributed search configuration, and the `shards` parameter for distributed search requests should use these VIPs.

A single VIP consisting of all the shard VIPs should be configured for all external systems to use the search service.

Integrating Monitoring Services


This functionality is not available with LucidWorks Search on AWS or Azure
Monitoring your application always is an important part of running production system. Most system administrators have used various tools to ensure everything is ok from the health of server's filesystem to the the temperature of CPUs. LucidWorks Search provides additional capabilities to integrate application level statistics information into these monitoring tools.

LucidWorks Search and Solr make available several JMX MBeans which can be used with stand-alone JMX clients, or integrated with servers that support MBeans, such as Nagios or Zabbix. More information on all these options is below.

JMX

JMX is a standard way for managing and monitoring all varieties of software components for Java applications. JMX uses objects called MBeans (Managed Beans) to expose data and resources from your application. LucidWorks Search provides number of read-only monitoring beans that provide useful statistical/performance information. Combined with JVM (platform JMX MBeans) and OS level information, it becomes powerful tool for monitoring.

Enabling JMX for LucidWorks Search

By default JMX is enabled in LucidWorks Search for local access only. If you want to connect and monitor application remotely you need to change `lwecore.jvm.params` parameter in the `LWE_HOME/conf/master.conf` file and add the following JVM parameters:

```
lwecore.jvm.params=... -Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=3000 -Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
-Djava.rmi.server.hostname=my.server.name
```

Where 3000 is an unused TCP port number.

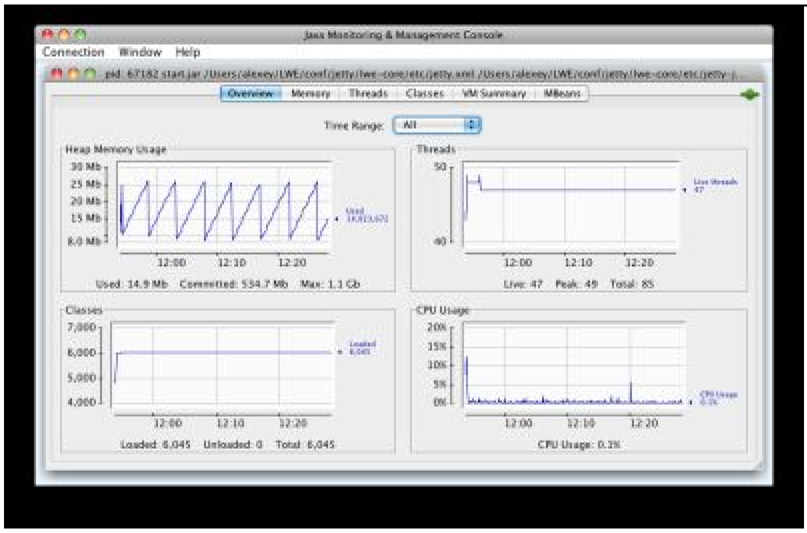
You might want to secure remote JMX access either by configuring a software or hardware firewall to allow connections to specified port only from your hosts/network or by configuring password authentication and/or SSL encryption. For more information about various security options please refer to the JMX documentation.

JMX Clients

There are number of various JMX clients you can use to connect to the LucidWorks Search server and browse available information.

JConsole

JConsole is a standard (part of the JDK) graphical monitoring tool to monitor Java Virtual Machine (JVM) and Java applications which provides a nice way to display memory and CPU information as well MBeans from arbitrary applications.

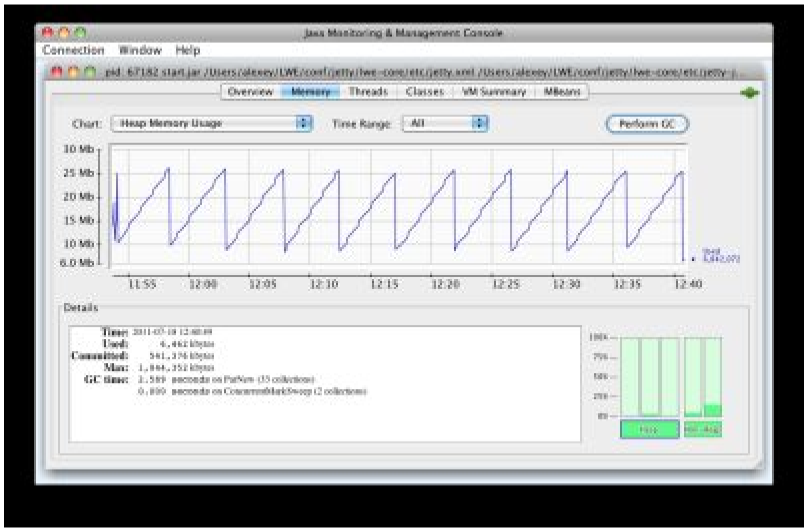

Java Monitoring & Management Console
Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
Time Range: All
Heap Memory Usage
Threads
Classes
CPU Usage

Used: 14.9 Mb Committed: 534.7 Mb Max: 1.1 Gb

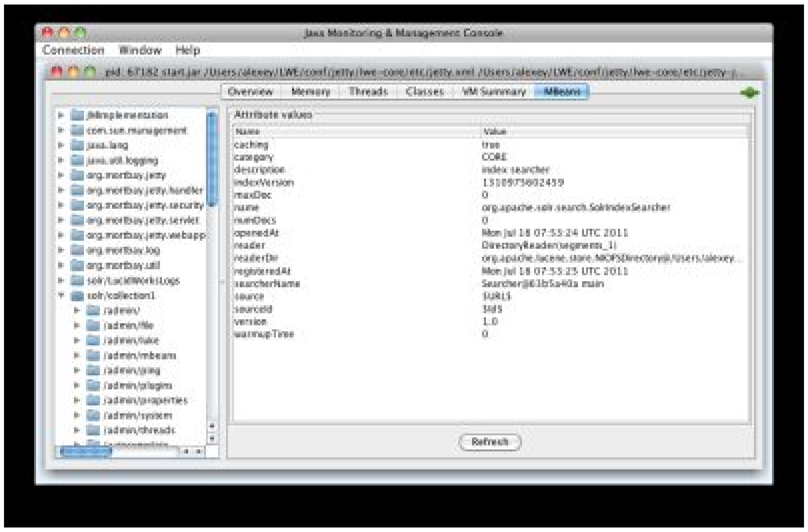
Live: 47 Peak: 49 Total: 85

Loaded: 6,045 Unloaded: 0 Total: 6,045

CPU Usage: 0.1%



```
       Time: 2011-07-18 12:40:49
       Used:      6,462 kbytes
  Committed:    541,376 kbytes
        Max:  1,044,352 kbytes
    GC time:  1.589  seconds on ParNew (35 collections)
              0.030  seconds on ConcurrentMarkSweep (2 collections)
```



Attribute values

Name	Value
caching	true
category	CORE
description	index searcher
indexVersion	1310975602459
maxDoc	0
name	org.apache.solr.search.SolrIndexSearcher
numDocs	0
openedAt	Mon Jul 18 07:53:24 UTC 2011
reader	DirectoryReader(segments_1)
readerDir	org.apache.lucene.store.NIOFSDirectory@/Users/alexey...
registeredAt	Mon Jul 18 07:53:23 UTC 2011
searcherName	Searcher@63b5a40a main
source	URL
sourceId	Id
version	1.0
warmupTime	0

JMXTerm

Jmxterm is an open source command line based interactive JMX client. It allows you to easily navigate JMX MBeans on remote servers without running a graphical interface or opening a JMX port. It can also be integrated with script languages such as Bash, Perl, Python, Ruby, etc. See the following as an example of how it can be used:

```
sh> java -jar jmxterm-1.0-alpha-4-uber.jar
Welcome to JMX terminal. Type "help" for available commands.

$>jvms
67183    ( ) - start.jar /Users/alexey/LWE/conf/jetty/rails/etc/jetty.xml
/Users/alexey/LWE/conf/jetty/rails/etc/jetty-jmx.xml
/Users/alexey/LWE/conf/jetty/rails/etc/jetty-ssl.xml
67182    (m) - start.jar /Users/alexey/LWE/conf/jetty/lwe-core/etc/jetty.xml
/Users/alexey/LWE/conf/jetty/lwe-core/etc/jetty-jmx.xml
/Users/alexey/LWE/conf/jetty/lwe-core/etc/jetty-ssl.xml
93534    ( ) - jmxterm-1.0-alpha-4-uber.jar
8554     ( ) -

$>open 67182
#Connection to 67182 is opened

$>domains
#following domains are available
JMImplementation
com.sun.management
java.lang
java.util.logging
org.mortbay.jetty
org.mortbay.jetty.handler
org.mortbay.jetty.security
org.mortbay.jetty.servlet
org.mortbay.jetty.webapp
org.mortbay.log
org.mortbay.util
solr/LucidWorksLogs
solr/collection1

$>domain solr/collection1
#domain is set to solr/collection1

$>beans
#domain = solr/collection1:
...
solr/collection1:id=collection1,type=core
solr/collection1:id=org.apache.solr.handler.StandardRequestHandler,type=standard
...
solr/collection1:id=org.apache.solr.search.FastLRUCache,type=fieldValueCache
solr/collection1:id=org.apache.solr.search.LRUCache,type=documentCache
solr/collection1:id=org.apache.solr.search.LRUCache,type=filterCache
```

```
solr/collection1:id=org.apache.solr.search.LRUCache,type=queryResultCache
solr/collection1:id=org.apache.solr.search.SolrFieldCacheMBean,type=fieldCache
...
solr/collection1:id=org.apache.solr.search.SolrIndexSearcher,type=searcher
solr/collection1:id=org.apache.solr.update.DirectUpdateHandler2,type=updateHandler

$>bean type=updateHandler,id=org.apache.solr.update.DirectUpdateHandler2
#bean is set to
solr/collection1:type=updateHandler,id=org.apache.solr.update.DirectUpdateHandler2

$>info
#mbean =
solr/collection1:type=updateHandler,id=org.apache.solr.update.DirectUpdateHandler2
#class name = org.apache.solr.core.JmxMonitoredMap$SolrDynamicMBean
# attributes
  %0   - adds (java.lang.String, r)
  %1   - autocommit maxTime (java.lang.String, r)
  %2   - autocommits (java.lang.String, r)
  %3   - category (java.lang.String, r)
  %4   - commits (java.lang.String, r)
  %5   - cumulative_adds (java.lang.String, r)
  %6   - cumulative_deletesById (java.lang.String, r)
  %7   - cumulative_deletesByQuery (java.lang.String, r)
  %8   - cumulative_errors (java.lang.String, r)
  %9   - deletesById (java.lang.String, r)
  %10  - deletesByQuery (java.lang.String, r)
  %11  - description (java.lang.String, r)
  %12  - docsPending (java.lang.String, r)
  %13  - errors (java.lang.String, r)
  %14  - expungeDeletes (java.lang.String, r)
  %15  - name (java.lang.String, r)
  %16  - optimizes (java.lang.String, r)
  %17  - rollbacks (java.lang.String, r)
  %18  - source (java.lang.String, r)
  %19  - sourceId (java.lang.String, r)
  %20  - version (java.lang.String, r)
#there's no operations
#there's no notifications

$>get cumulative_adds
#mbean =
solr/collection1:type=updateHandler,id=org.apache.solr.update.DirectUpdateHandler2:
cumulative_adds = 125;
```

JMX MBeans

LucidWorks includes a number of useful JMX MBeans, some available through Solr and some developed in LucidWorks Search itself:

Solr MBeans

Domain	Objects	Available attributes	Comments
solr/ *collection*	type=updateHandler, id=org.apache.solr.update. DirectUpdateHandler2	cumulative_adds, cumulative_deletesById, cumulative_deletesByQuery, cumulative_errors, commits, autocommits, optimizes, rollbacks, docsPending, etc	This MBean provides comprehensive information about indexing activity like number of added documents, number of errors, number of commits, autocommits and optimize operations. It is really useful to plot that information into graphs in your monitoring system. The *cumulative_errors* parameter shows the number of low level IO exceptions.
solr/ *collection*	type=/update, id=org.apache.solr.handler. XmlUpdateRequestHandler	request, errors, avgTimePerRequest, etc	If using direct Solr API, there are separate beans for all types of handlers you can use to index documents into the system, such as XML, CSV, JSON request handlers. It makes sense to add this UpdateRequest Handler information to indexing graphs as well. You might also setup monitoring alert on a number of errors for particular update handler to make sure LucidWorks Search clients don't hit any errors during indexing like invalid fields names or types, no required fields in indexed documents, etc.

solr/ *collection*	type=/lucid, id=org.apache.solr.handler. StandardRequestHandler	requests, errors, timeouts, avgTimePerRequest	This MBean represents the default LucidWorks Search request handler and provides statistics about number of search requests, errors, timeouts and average response time for search requests. It's pretty useful to display this information on monitoring graphs as well as setup monitoring alerts, such as, "notify administrator if average response time is more than 0.5 second or total number of errors and timeouts is more than 1% of total requests".

solr/ *collection*	type=searcher, id=org.apache.solr.search. SolrIndexSearcher	numDocs, warmupTime	*numDocs* is the total number of documents in the index. *warmupTime* is the amount of time a new Searcher takes to warm. When LucidWorks Search commits new data into index, a new Searcher is opened and warmed. The warming operation regenerates caches from the previous Searcher instance and runs some predefined in *solrconfig.xml* queries to warm up IO filesystem cache and load Lucene FieldCache in memory. This attribute basically defines how long does it take to commit before new data will be available to users. It makes sense to monitor this parameter and setup trigger to alert the LucidWorks Search administrator if it takes more time than you expect.

solr/ *collection*	type=filterCache, id=org.apache.solr.search. LRUCache	cumulative_evictions, cumulative_hitratio, cumulative_hits, cumulative_inserts, cumulative_lookups, warmupTime, etc	Solr caches popular filter query (fq=category:IT) attributes as unordered sets of document ids. This technique significantly improves search filtering/faceting performance. *size* is the current number of cached filter queries. *cumulative_hitratio* represents if this cache is successfully utilized by giving the ratio of successful cache hits to overall number of lookups. If it's low (such as < 0.3 or 30%) over long period of time then you might want either increase cache size or disable it at all to reduce performance overhead.
solr/ *collection*	type=queryResultCache, id=org.apache.solr.search. LRUCache	cumulative_evictions, cumulative_hitratio, cumulative_hits, cumulative_inserts, cumulative_lookups, warmupTime, etc	This cache stores ordered sets of document IDs and the top N results of a query ordered by some criteria. It has the same attributes as filterCache.
solr/ *collection*	type=documentCache, id=org.apache.solr.search. LRUCache	cumulative_evictions, cumulative_hitratio, cumulative_hits, cumulative_inserts, cumulative_lookups, etc	The documentCache stores Lucene Document objects that have been fetched from disk.

LucidWorks Search MBeans

Domain	Objects	Available attributes	Comments

lwe	id=crawlers, name=<data_source_id>, type=datasources	total_runs, total_time, num_total, num_new, num_updated, num_unchanged, num_failed, num_deleted	This MBean displays crawlers statistics information for specific data source (like number of processed documents, number of errors, etc). If you have periodically or long running scheduled data source then you might want to monitor and alert if there's any problem with the underlying source (web site, SharePoint server, etc) or how optimized your incremental crawl is (percentage of num_unchanged to num_total), for example.
lwe	id=crawlers, name=<collection_name>, type=collections	total_runs, total_time, num_total, num_new, num_updated, num_unchanged, num_failed, num_deleted	If you have multiple data sources and don't want to monitor on per data source level, but keep an eye on aggregate numbers for the whole collection you might want to use this bean.
lwe	id=crawlers, type=total	total_runs, total_time, num_total, num_new, num_updated, num_unchanged, num_failed, num_deleted	You can use this MBean if you have multiple collections (homogeneous collections or multi-tenant architecture) to monitor on per instance level.

Integrating with Monitoring Systems

Using JConsole and JmxTerm tools is a good way to explore information hidden in JMX, but what you really need is to monitor your application automatically, record historical information, display it in a graphical form, configure parameters thresholds as triggers and send alerts in case of denial of service or performance problems. There are various standard sysadmin tools for that and integrating LucidWorks with them is no different than with any other Java application. The idea is that you can retrieve application information and send it to external monitoring system. In our documentation we provide two examples of integrating LucidWorks server with popular open source monitoring tools - *Zabbix* and *Nagios*.

Zabbix

Zabbix is an enterprise-class open source distributed monitoring solution for networks and applications. It comes with pre-defined templates for almost all operating systems as well as various open source applications. It also has a great template for JVM that contains the most vital statistics of arbitrary Java application. There are different ways how you can integrate LucidWorks with Zabbix and the best approach depends on the Zabbix release version.

Pre-2.0 Releases

Zabbix does not contain built-in support for monitoring Java applications prior to v2.0, but if you are handy with scripting and command line tools then there are two possible approaches:

UserParameter: You can configure the Zabbix system agent to send custom monitored items using `UserParameter`. For retrieving JMX statistics you can use either cmdline-jmxclient or jmxterm as command line clients.

```
UserParameter=jvm.maxthreads, java -jar cmdline-jmxclient.jar localhost:3000
java.lang:type=Threading PeakThreadCount
```

zabbix_sender utility: If you have a large number of JMX monitored items, or you need to monitor some items quite frequently, then spawning a Java Virtual Machine process to get a single object/attribute can be too expensive. In this case consider scripting JMX interactions using the JMXTerm command line tool and your favorite scripting language. The solution below is in Ruby but could be implemented using any scripting language. The main idea is that you can run a JMXTerm java application from your script and communicate with it using `stdin` and `stdout` streams using the expect library.

```
require "open3"
require 'expect'
....
# run jmxterm java application
stdin, stdout, wait_thr = Open3.popen2e('java -jar jmxterm-1.0-alpha-4-uber.jar')
# wait for prompt
result = stdout.expect('$>', 60)
...
# connect to specific jvm
stdin.puts("open #{process_id}")
result = stdout.expect('$>', 60)
...
stdin.puts('get -d solr/collection1 -b
type=searcher,id=org.apache.solr.search.SolrIndexSearcher numDocs')
result = stdout.expect('$>', 60)
# parse response from jmxterm command
...
# run zabbix_sender command to send single item or save multiple values into file and
send as a batch
output = `zabbix_sender -z #{@server_name} -p #{@server_port} -i file.txt`.chomp
# parse response and validate that operation is successful
...
```

1.9x and 2.x Releases

Zabbix 2.0 is currently in beta (the latest version was 1.9.9 as of 15 February 2012) and contains built-in support for monitoring Java applications (Zabbix Java proxy). For more information please see the JMX Monitoring section of the Zabbix manual.

The following steps describe how to integrate LucidWorks Search with the Zabbix 2.0 (1.9.x) release.

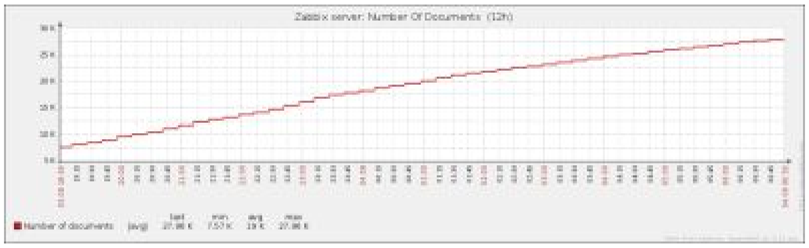
1. Download and install 2.0 (1.9.x) release according to the official documentation.
2. In order to build Zabbix JMX proxy you should build Zabbix package with the `--enable-java` configuration option, such as `./configure --enable-server --with-mysql --enable-java`.

   If you intend to run Zabbix on the same server where you installed LucidWorks, you may want to add the `--enable-agent` option, such as `./configure --enable-server --with-mysql --enable-java --enable-agent`.
3. After `make install`, copy the example `init.d` start script from `misc/init.d/debian/zabbix-server` into the `/etc/init.d` directory and edit it to start the JMX proxy daemon by adding `<install_dir>/sbin/zabbix_java/startup.sh` and `<install_dir>/sbin/zabbix_java/shutdown.sh` calls to the corresponding options in `init.d`.

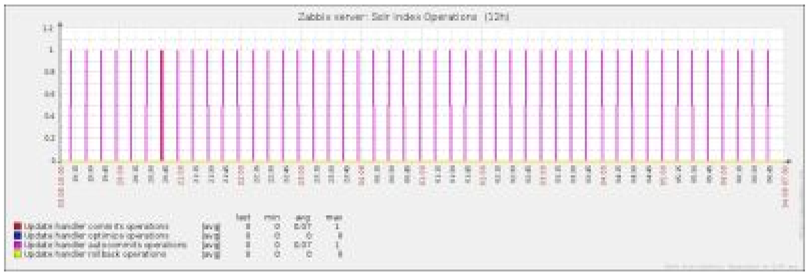
4. Configure JMX proxy in `/etc/zabbix/zabbix_server.conf` by editing the `JavaGateway`, `JavaGatewayPort` and `StartJavaPollers` parameters. The `JavaGatewayPort` should match the `LISTEN_PORT` defined in `<install_dir>/sbin/zabbix_java/settings.sh`. It is also recommended to enable JMX proxy verbose logging by editing `<install_dir>/sbin/zabbix_java/lib/logback.xml` and changing the `file` element to point to your log file directory and setting the `level` attribute to "debug".
5. Import, using the Zabbix UI, the sample templates found in `$LWE_HOME/app/examples/zabbix` called `lwe_zabbix_templates.xml` (there are 3 in that file).
6. Install the Zabbix agent to the server where LucidWorks Search is installed and configure it to connect to the Zabbix server.
7. Add Zabbix host and assign proper template for the specific operating system (i.e., linux, freebsd, etc.).
8. Assign the imported templates (Template_JVM, Template_Solr, Template_LWE) to that host.
9. Enable JMX monitoring in LucidWorks and allow the Zabbix server connect to JMX interface over the network. Instructions to enable JMX monitoring are in the Enabling JMX for LucidWorks Search section of this Guide.
10. Add the JMX interface to the host where LucidWorks is installed. This is done via the Zabbix UI by creating JMX agents for each counter.
11. Start any activity in LucidWorks (such as, crawling, indexing, or serving queries) and review the graphs for the monitored host (see screenshots below).

Example graphs

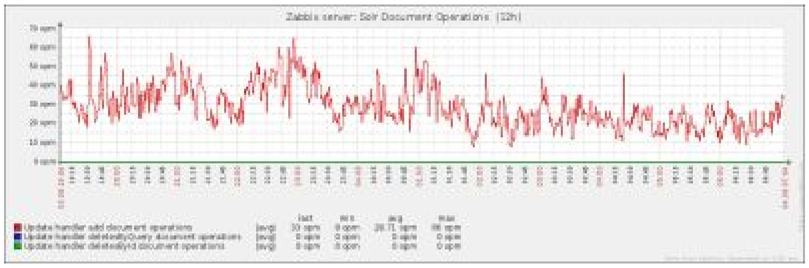
- Total number of documents in search index



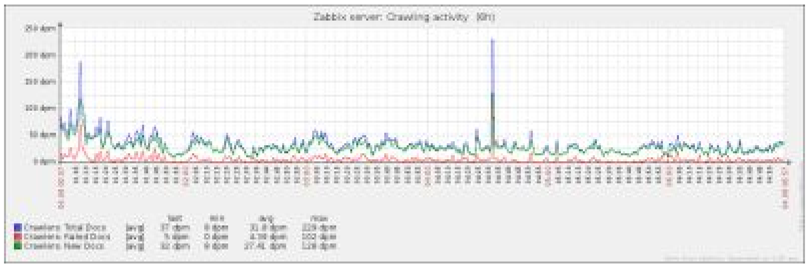
- Solr index operations (commits, optimizes, rollbacks)



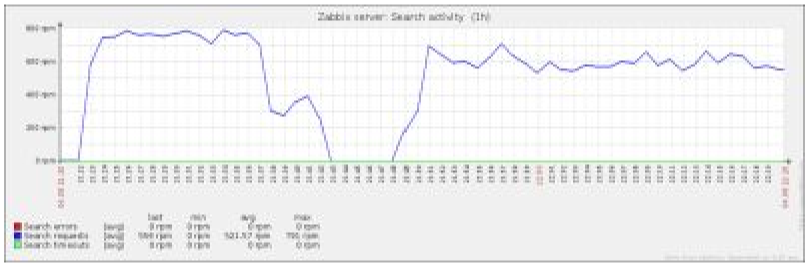
- Solr document operations (adds, deletes by id or query)



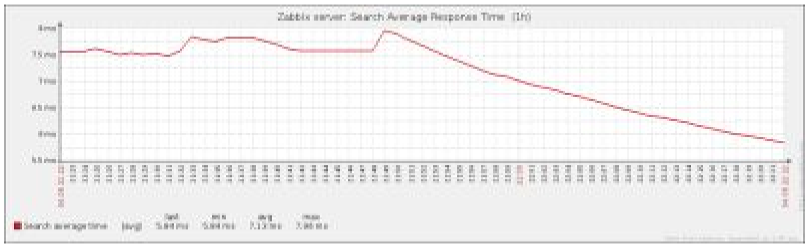
- Crawling activity - number of total documents processed, number of failures (retrieve, parsing), number of new documents



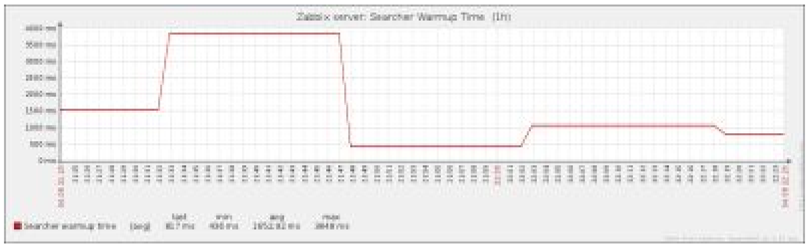
- Search activity - number of search requests



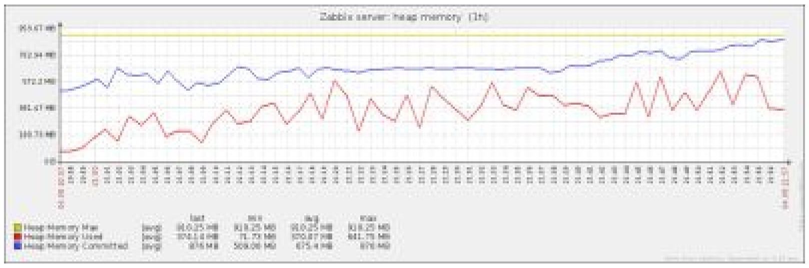
- Search Average Response Time



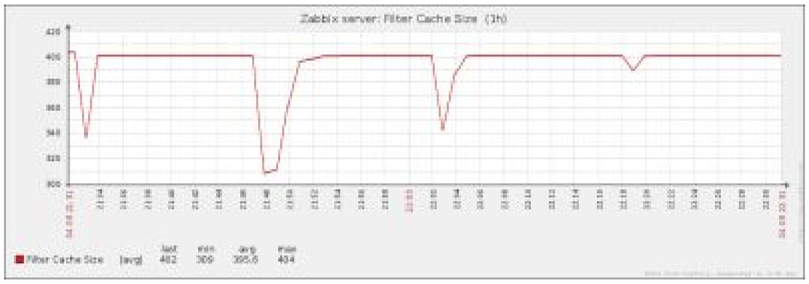
- Searcher Warmup Time (how fast committed docs become visible/searchable)

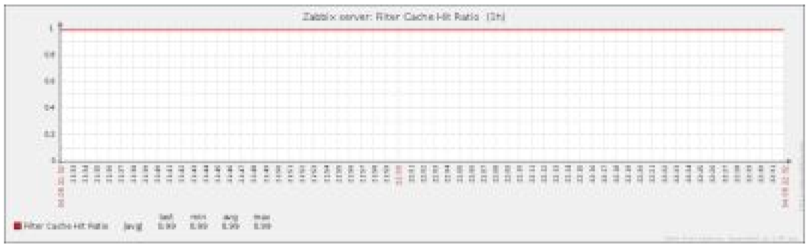


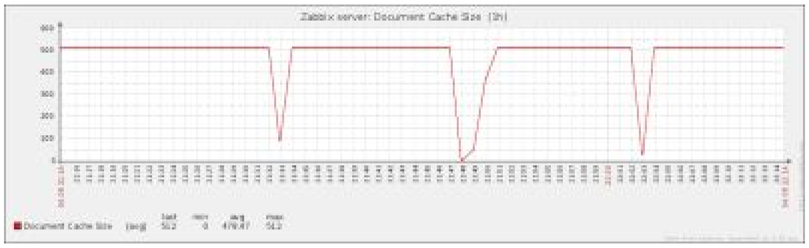
- Java Heap Memory Usage

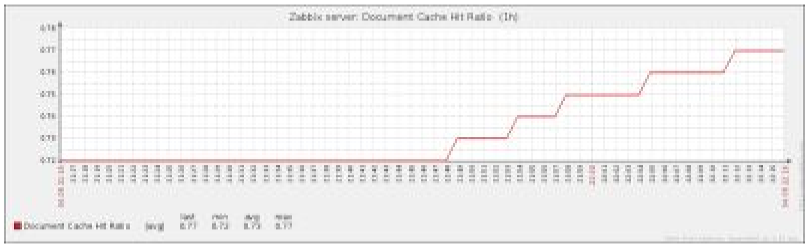


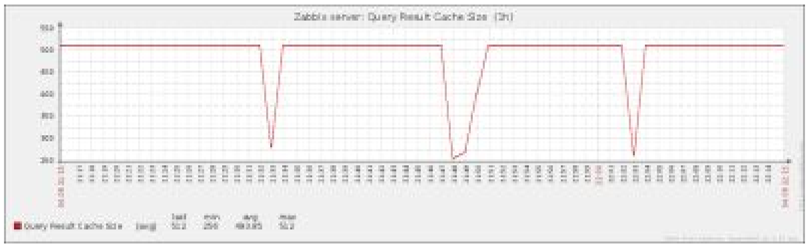
- Caches stats

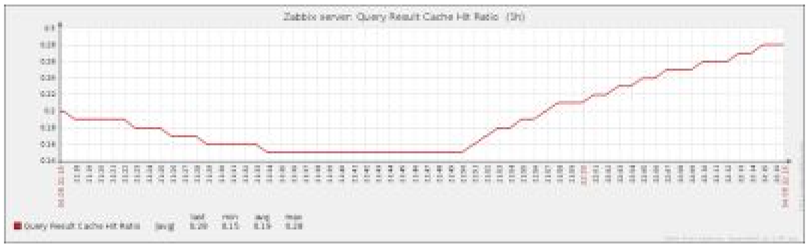












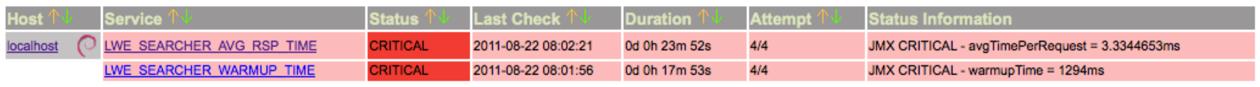
Nagios

Nagios is a popular open source computer system and network monitoring software application. It watches hosts and services, alerting users when things go wrong and again when they get better. There are different Nagios plugins that allow you to monitor Java applications using JMX interface. We recommend you to use Syabru Nagios JMX Plugin as the most mature plugin that supports different data types (integers, floats, string regular expressions) and advanced Nagios threshold syntax. In order to install Syabru Nagios JMX Plugin you should copy `check_jmx` and `check_jmx.jar` from the downloaded package to Nagios `plugins` directory and add check_jmx command definition to either global `commands.cfg` configuration file or put the `jmx.cfg` file into `nagios_plugins` configuration directory. The next step is to define Nagios services, as in this example:

```
# LWE searcher warmup time is no more than 1) 1 second - warning state 2) 2 seconds -
critical state
define service {
        hostgroup_name                  all
        service_description             LWE_SEARCHER_WARMUP_TIME
        check_command                   check_jmx!3000!-O
"solr/collection1:type=searcher,id=org.apache.solr.search.SolrIndexSearcher" -A
warmupTime -w 1000 -c 2000 -u ms
        use                             generic-service
        notification_interval           0
}
# LWE search average response time is no more than 1) 100ms - warning state 2) 200ms -
critical state
define service {
        hostgroup_name                  all
        service_description             LWE_SEARCHER_AVG_RSP_TIME
        check_command                   check_jmx!3000!-O
"solr/collection1:type=/lucid,id=org.apache.solr.handler.StandardRequestHandler" -A
avgTimePerRequest -w 100 -c 200 -u ms
        use                             generic-service
        notification_interval           0
}
```

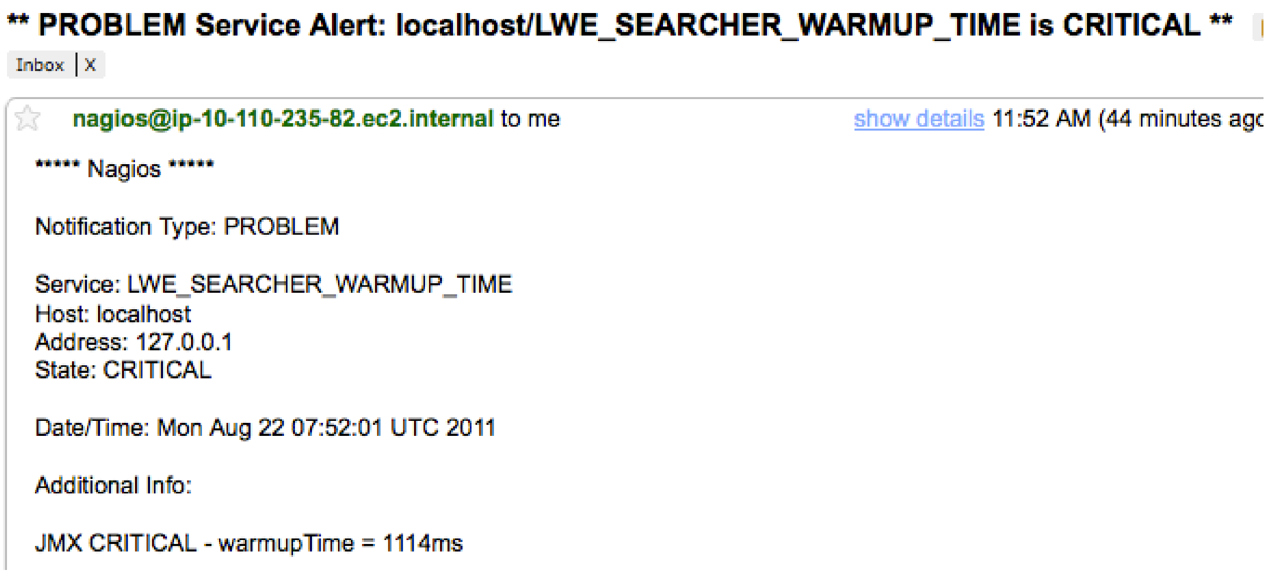
After you setup your services and reload the Nagios configuration you can monitor application state using either the Nagios web UI or receive email notifications.

- Nagios UI screenshot (thresholds on the screenshots are lowered to trigger critical state as an example)



Host	Service	Status	Last Check	Duration	Attempt	Status Information
localhost	LWE_SEARCHER_AVG_RSP_TIME	CRITICAL	2011-08-22 08:02:21	0d 0h 23m 52s	4/4	JMX CRITICAL - avgTimePerRequest = 3.3344653ms
	LWE_SEARCHER_WARMUP_TIME	CRITICAL	2011-08-22 08:01:56	0d 0h 17m 53s	4/4	JMX CRITICAL - warmupTime = 1294ms

- Nagios email alert



** PROBLEM Service Alert: localhost/LWE_SEARCHER_WARMUP_TIME is CRITICAL **

Inbox | X

nagios@ip-10-110-235-82.ec2.internal to me show details 11:52 AM (44 minutes ago

```
***** Nagios *****

Notification Type: PROBLEM

Service: LWE_SEARCHER_WARMUP_TIME
Host: localhost
Address: 127.0.0.1
State: CRITICAL

Date/Time: Mon Aug 22 07:52:01 UTC 2011

Additional Info:

JMX CRITICAL - warmupTime = 1114ms
```

New Relic

Helpful Tips

- **OS file system cache**: One of the frequent problems with LucidWorks Search and Lucene/Solr applications is that if you do not have enough free memory and a significant index size you might notice performance problems because there's not enough free memory for the file system cache. IO cache is a crucial resource for search applications, so it definitely makes sense to monitor this parameter and display it in graphs with other memory information like free memory, jvm heap memory, swap, etc. This parameter is part of the OS level monitoring in Zabbix (name is `vm.memory.size[cached]`).
- **File descriptors**: Another problem is that sometimes your application can hit OS or per process file descriptor limits. It is also recommended to monitor these parameters and set trigger thresholds for these parameters.
- **CPU usage**: Default Zabbix templates have triggers for CPU load average numbers. You might want to tune thresholds for your server based on number of CPUs and expected load.
- **Heap memory usage and garbage collector statistics**: Zabbix Java template contains multiple items and triggers for memory and garbage collector invocation counts. You should also tune these parameters to match your scenario.
- **Solr index size and free disk space**: These should be set properly to avoid "Out Of Disk Space" errors.

LucidWorks REST API Reference

In a system in which requirements remain fairly static, or at least predictable, the LucidWorks user interface may be sufficient for managing your system. For more complex or dynamic systems, LucidWorks Search provides programmatic, remote access to many aspects of configuration and operation through a REST API. All tasks that can be accomplished with the Admin UI can also be done with the REST API. Some of the more complex administration tasks are available through the API only.

Topics covered in this section:

About the LucidWorks Search REST APIs

The LucidWorks Search REST APIs focus on administrative tasks of manipulating settings, fields, data sources, and other configuration options as well as some system monitoring functions. Programmatic access to search itself is by making GET requests to Solr directly, as documented in the section Getting Search Results. The Apache Solr Reference Guide also contains a great deal of information about searching Solr.

The LucidWorks Search APIs use GET, POST, PUT and DELETE requests. The results of a REST request depend not only on the type of request, but on whether you called it on an object or group of objects. When you use REST requests with LucidWorks, you send a JSON request to the endpoint specified for the object (or group) you want to work with. The system then sends back the results as a JSON object. Currently, LucidWorks only recognizes JSON requests.

About Server Addresses in the API Documentation

The LucidWorks Search REST API uses the Core component, installed at http://localhost:8888/ by default and many examples in this Guide use this as the server location. If you changed this location on install, or if you are using LucidWorks Search hosted on Azure or AWS, be sure to change the destination of your REST requests.

About LucidWorks REST APIs for Hosted Customers

If using LucidWorks Search hosted by LucidWorks in AWS or Azure, the server call **must** include the API Key used as authentication credentials. This is basic authentication, and your instance will have a URL with "https://s-**XXXXXXXX**.lucidworks.io" where XXXXXXXX is 8 characters (letters or numbers). So, if your instance URL is "https://s-9sdff10b.lucidworks.io/" you would use that in place of any example API calls that used "http://localhost:8888". For example, this call to get all collections:

```
curl 'http://localhost:8888/api/collections'
```

would be changed to:

```
curl -u 'API_Key:password' 'https://s-9sdff10b.lucidworks.io/api/collections'
```

The API_Key can be found by logging in to your LucidWorks Search hosted instance, and clicking "My Account" at the upper right of the screen. Click "API Access" on the left to view the API key. The password is 'x' by default. There is not currently a way to change the default password so you should take care not to expose this key when posting to our forums, as that information is publicly available to other customers.

For users hosted on Windows Azure, the above URL would be: `'https://s-9sdff10b.azure.lucidworks.io/api/collections'`

Quick Start

A few pages give examples of using the APIs in configuring and managing a LucidWorks-based search application.

- Getting Started Indexing gives an overview of the calls required to set up a data source and index content.
- Error Response Format shows the format of errors if the calls go wrong.
- Overview of REST API calls provides a quick reference of the APIs and what they do.
- Advanced Operations Using the REST API contains examples of more advanced tasks, such as creating and editing fields, monitoring data sources, and configuring users.
- Example Clients includes three sample clients (in Perl, Python and C#) that were implemented using the REST API.

List of Available APIs

- Version: Shows information about the LucidWorks and Solr versions being used by the system.

- Collections: Groups of documents that are logically separate.
    - Collection Information: Get information about the collection.
    - Collection Templates: Get information about templates that can be used when creating new collections.
    - Collection Index Delete: Delete the entire index or only data from a single data source.
    - Activities: Control schedules for resource-intensive operations such as optimization and building indexes.
        - Status: The status of currently running Activities.
        - History: Statistics for the last 50 runs of an Activity.
    - Data Sources: The conduits by which data enters LucidWorks indexes.
        - Schedules: Control when data is imported.
        - Data Source Jobs: Start and Stop data source jobs.
        - Status: Get the status of currently running data sources.
        - History: Statistics for the last 50 runs of a data source.
        - Crawl Data: Delete the crawling history for a specific data source.
    - Batch Crawling: Create and manage crawling when the data shouldn't be indexed until a later time.
    - JDBC Drivers: Load required JDBC drivers for database indexing.
    - doc:Field Types: Create new field types or modify existing types.
    - Fields: How data from a single document is organized in LucidWorks indexes.
    - Dynamic Fields: Define dynamic fields for on-the-fly field creation based on specific patterns.
    - Filtering Results: Use Access Control Lists to filter results for Windows Shares.
    - Search Handler Components: List active search components for a particular search handler.
    - doc:Settings: Many different query- and index-time settings.
    - Caches: Configure how Solr caches documents to speed results.
    - Click Scoring: Integrate Click Scoring with your own search application to record which documents are most popular with users.
    - Roles: Configure search filters to control access to documents.
- Alerts: Create and modify alerts for users.
- Users: Create user accounts (if not using an external user management system, such as LDAP).
- SSL Configuration: Configure LucidWorks to work with SSL.
- Crawler Status: Status of communication with the Connectors component.

When creating or updating a resource (using a POST or a PUT), you generally only need to include those entries in your map that you need to set. When LucidWorks creates a resource, entries you omit will get their default values. When you update a resource, entries you omit will retain their previous values.

JSON primitives can be used and will be returned, but LucidWorks also accepts string parsable equivalents. For example, both 4 or "4" are valid inputs for an integer type, but 4 will be returned by the REST API.

APIs that take a list will also take a single variable (for example, a list<string> accepts string) and treat it as a list of size 1.

For convenience, you can append ".json" to any method.

Overview of REST API calls

This section provides an overview of the various API calls, what they do, with links to more documentation for each call.

Collection-Specific APIs

These APIs all require the collection name to be specified in the request. They apply only to the collection named; changes to multiple collections require multiple API calls to each collection.

Collection APIs

API	Description
/api/collections	GET: Retrieve a list of collections for the entire installation with basic information about name and data location on disk. POST: Create a new collection.
/api/collections/*collection*	GET: Retrieve basic information for a specific collection. DELETE: Delete a collection.

/api/collections/*collection*/info	GET: Retrieve detailed information about the size and contents of a collection.
/api/collections/*collection*/info/*name*	GET: Retrieve a single data point about the collection such as the free disk space or the number of documents.
/api/collections/*collection*/index	DELETE: Delete the index for an entire collection.

Data Source & Crawler APIs

API	Description
/api/collections/*collection*/datasources	GET: Retrieve a list of all data sources and their configurations. POST: Create a new configuration.
/api/collections/*collection*/datasources/*id*	GET: Retrieve details of the configuration for a specific data source. PUT: Update a specific data source. DELETE: Delete a data source.
/api/collections/*collection*/datasources/*id*/schedule	GET: Retrieve the schedule for a specific data source. PUT: Update the schedule for a data source.
/api/collections/*collection*/datasources/all/job	GET: Retrieve the crawling status of all data sources. PUT: Instruct all data sources to start crawling. DELETE: Instruct all data sources to stop crawling.
/api/collections/*collection*/datasources/*id*/job	GET: Retrieve the crawling status of a specific data source. PUT: Instruct a specific data source to start crawling. DELETE: Instruct a specific data source to stop crawling.
/api/collections/*collection*/datasources/*id*/status	GET: Retrieve the crawling status of a specific data source.
/api/collections/*collection*/datasources/*id*/history	GET: Retrieve the history of the last 50 crawls of a specific data source.
/api/collections/*collection*/datasources/*id*/crawldata	DELETE: Delete the crawl history for a data source.
/api/collections/*collection*/datasources/*id*/index	DELETE: Delete indexed content for a specific data source.

/api/collections/*collection*/datasources/*id*/mapping	GET: Get only the field mapping settings for a specific data source. PUT: Update the field mapping settings for a specific data source. DELETE: Remove all custom field mapping settings for a specific data source, and return it to defaults.
/api/collections/*collection*/datasources/*id*/mapping/*part*	GET: List the values for a specific part of the field mapping settings for a data source. DELETE: Remove custom values for a specific part of the field mapping settings for a data source (and return it to defaults).
/api/collections/*collection*/datasources/*id*/mapping/*part*/*key*	GET: List the values for a specific field mapping setting, within a *part*. DELETE: Remove custom values for a specific field mapping setting, within a *part* (and return it to defaults).
/api/collections/*collection*/jdbcdrivers	GET: Retrieve a list of all JDBC drivers. POST: Upload a new JDBC driver.
/api/collections/*collection*/jdbcdrivers/*filename*	DELETE: Remove a JDBC driver.
/api/collections/*collection*/jdbcdrivers/classes	GET: Retrieve a list of all JDBC 4.0 compliant drivers.
/api/collections/*collection*/batches	GET: Retrieve a list of all batches. DELETE: Delete all batches.
/api/collections/*collection*/batches/*crawler*	GET: Retrieve a list of all batches for a specific crawler. DELETE: Delete all batches for a specific crawler.
/api/collections/*collection*/batches/*crawler*/job	GET: Retrieve a list of all jobs for a specific crawler. PUT: Define a job for a specific crawler.
/api/collection/*collection*/batches/*crawler*/job/*batch_id*	GET: Retrieve the status of a batch crawl job. PUT: Start a batch crawl job. DELETE: Stop a batch crawl job.

Activity APIs

API	Description
/api/collections/*collection*/activities	GET: Retrieve a list of activities and their schedules. POST: Create an activity and its schedule.

/api/collections/*collection*/activities/*id*	GET: Retrieve the schedule for a specific activity. PUT: Update the schedule for a specific activity. DELETE: Delete the schedule for a specific activity.
/api/collections/*collection*/activities/*id*/status	GET: Retrieve the status of a specific activity.
/api/collections/*collection*/activities/*id*/history	GET: Retrieve the history of the last 50 runs of a specific activity.

Field-Related APIs

API	**Verb & Description**
/api/collections/*collection*/fields	GET: Retrieve all fields and their attributes. POST: Create a new field.
/api/collections/*collection*/fields/*name*	GET: Retrieve attributes for a specific field. PUT: Update the attributes for a specific field. DELETE: Delete a field.
/api/collections/*collection*/fieldtypes	GET: Retrieve all field types and their attributes. POST: Create a new field types.
/api/collections/*collection*/fieldtypes/*name*	GET: Retrieve attributes for a specific field type. PUT: Update the attributes for a specific field type. DELETE: Delete a field type.
/api/collections/*collection*/dynamicfields	GET: Retrieve all dynamic fields and their attributes. POST: Create a new dynamic field.
/api/collections/*collection*/dynamicfields/*name*	GET: Retrieve attributes for a specific dynamic field. PUT: Update the attributes for a specific dynamic field. DELETE: Delete a dynamic field.

Setting APIs

API	**Verb & Description**
/api/collections/*collection*/settings	GET: Retrieve all settings for a collection. PUT: Update settings for a collection.
/api/collections/*collection*/settings/*name*	GET: Retrieve a specific setting. PUT: Update a specific setting.

/api/collections/*collection*/caches	GET: Retrieve details of all configured caches. POST: Create a new cache configuration.
/api/collections/*collection*/caches/*name*	GET: Retrieve details of a specific cache. PUT: Update details of a specific cache DELETE: Delete a cache configuration.

User & Search-Related APIs

API	Verb & Description
/api/collections/*collection*/click	GET: Retrieve statistics about recent Click Scoring events. PUT: Record Click Scoring events.
/api/collections/*collection*/roles	GET: Retrieve a list of all existing roles. POST: Create a new role.
/api/collections/*collection*/roles/*role*	GET: Retrieve details for a specific role. PUT: Update details for a specific role. DELETE: Delete a role.
/api/collections/*collection*/filtering	GET: Retrieve a list of existing search filter configurations. POST: Create a new search filter configuration.
/api/collections/*collection*/filtering/*instance*	GET: Retrieve details of a specific search filter configuration. PUT: Update the details of a specific search filter configuration. DELETE: Delete a specific search filter configuration.
/api/collections/*collection*/components/*list-name* ?handlerName=/*handlerName*	GET: List the search components for a search handler. PUT: Update the search components for a search handler.

Global APIs

These APIs are not specific to a collection, but instead apply to the entire LucidWorks installation.

API	Verb & Description
/api/collectiontemplates	GET: Retrieve a list of available collection templates.

/api/version	GET: Retrieve the Solr and LucidWorks version of the application.
/api/alerts	GET: Retrieve all alerts and their attributes. POST: Create an alert.
/api/alerts?username=*username*	GET: Retrieve all alerts for a specific user.
/api/alerts/*id*	GET: Retrieve details for a specific alert. PUT: Update details for a specific alert. DELETE: Delete a specific alert.
/api/alerts/*id*/check	PUT: Run an alert to check for new results.
/api/users	GET: Retrieve a list of all locally-created users. POST: Create a user locally.
/api/users/*username*	GET: Retrieve details for a specific user. PUT: Update details for a specific user. DELETE: Delete a user.
/api/config/ssl	GET: Retrieve details of the current SSL implementation. PUT: Update the SSL implementation.
/api/crawlers/status	GET: Get status and history of communication between the Core and Connectors components.

Pretty Printing JSON Output

JSON is easy for computers to read, but not so easy for people. If you'd like to have the JSON responses from your API calls nicely formatted when running on the command line, there are a few options.

- json.tool'>Python's `json.tool`
- Custom Shell Script Using Python
- Browser Options

Python's json.tool

If you have Python installed on the server running the LucidWorks Search Core component, you can usually just pipe the output to Python's `json.tool` by adding `python -m json.tool` to the end of every API call. For example:

```
curl http://localhost:8888/api/collections/collection1/settings | python -m json.tool
```

Custom Shell Script Using Python

1. Install Python's setuptools per the instructions in the link. This will install a tool called `easy_install` which includes `simplejson`. To check if you have it, `sudo easy_install simplejson`.
2. Create a python script as below, name it "pretty-json.sh", as we did, and save it somewhere convenient:

```
#!/usr/bin/env python
#// Convert JSON data to human-readable form.
#//(Reads from stdin and writes to stdout)
import sys
import simplejson as json
print json.dumps(json.loads(sys.stdin.read()), indent=4)
sys.exit(0)
```

Browser Options

Many of the browser plugins that support REST API calls, such as Chrome's Postman or Firefox's RESTClient, will format the JSON output into readable form.

Getting Started Indexing

The following examples use the LucidWorks REST API for some of the most common uses of the API, to control and monitor data sources and indexing. For example, suppose you wanted to create a data source that crawls a web site, such as http://www.grantingersoll.com; in general, if you were using `curl` to make your REST requests directly, the process would look like this:

1. First, create the data source:

```
curl -H "Content-Type: application/json"
-d '{"url" : "http://www.grantingersoll.com", "crawl_depth" : "2", "type" : "web",
     "name" : "Sample Site", "crawler": "lucid.aperture"}'
'http://<server address>/api/collections/collection1/datasources'
```

Most of these keys, such as `url` and `type`, are obvious; check the full documentation for the keys involved in creating various types of Data Sources.

The response is a JSON representation of the object you just created, which includes the `id` value:

```
{
    "id": 1,
    "collection": "collection1",
    "type": "web",
    "url": "http://www.grantingersoll.com/",
    "crawler": "lucid.aperture",
    "bounds": "tree",
    "category": "Web",
    "name": "Sample Site",
    "crawl_depth": 2,
    "max_bytes": 10485760,
    "include_paths": [

    ],
    "collect_links": true,
    "exclude_paths": [

    ] ,
    "mapping": {
        "multiVal": {
            "fileSize": true,
            "body": true,
            ...
        },
        "defaultField": null,
        "mappings": {
            "slide-count": "pageCount",
            "content-type": "mimeType",
            "body": "body",
            ...
        },
        "dynamicField": "attr",
        "types": {
            "filesize": "INT",
            "pagecount": "INT",
            "lastmodified": "DATE",
            "datecreated": "DATE",
            "date": "DATE"
        },
        "uniqueKey": "id" ,
        "datasourceField": "data_source"
    }
}
```

2. The next step is to tell LucidWorks to index this data source by creating a new `job`.

In this case, that means sending a `PUT` request for collection `collection1`, data source `1`:

```
curl -X PUT 'http://<server address>/api/collections/collection1/datasources/1/job'
```

(Note the -X PUT switch.)

In this case, there is no JSON object to pass; like many of the REST API calls, the important information is in the URL. In this case, you're passing the collection (`collection1`) and the data source number on which the job should be run (`1`). Check the full documentation for more information on stopping and starting Data Source Jobs.

This request does not return anything, but it does start the index running.

3. To check the status, you can send a `GET` request:

```
curl 'http://<server address>/api/collections/collection1/datasources/1/status'
```

Once again, you are passing the collection and data source number in the URL. The response tells you that the data source is still being indexed:

```
{
    "id": 1,
    "crawlStarted": "2011-03-17T16:34:45+0000",
    "numUnchanged": 0,
    "crawlState": "RUNNING",
    "crawlStopped": null,
    "jobId": "1",
    "numUpdated": 0,
    "numNew": 90,
    "numFailed": 0,
    "numDeleted": 0
}
```

When the job is complete, the response will look something like this:

```
{
    "id": 1,
    "crawlStarted": "2011-03-17T16:34:45+0000",
    "numUnchanged": 0,
    "crawlState": "FINISHED",
    "crawlStopped": "2011-03-17T28:26:06+0000",
    "jobId": "1",
    "numUpdated": 0,
    "numNew": 328,
    "numFailed": 2,
    "numDeleted": 0
}
```

4. You can also inspect the properties of the overall index itself:

```
curl 'http://<server address>/api/collections/collection1/info'
```

Again, you are passing the collection name (`collection1`) in the URL. Check the full documentation for more information on working with Collections.

This call gives you a response that shows all of the information about the collection itself:

```
{
    "free_disk_space": "12.2 GB",
    "index_last_modified": "2011-03-17T21:55:45+0000",
    "index_has_deletions": false,
    "data_dir":
"C:\\Users\\Nick\\LucidImagination\\LucidWorksSearch\\bin\\..\\solr\\cores\\collection1_0\
"index_size": "1.3 MB",
    "index_directory": {
        "directory":
"C:\\Users\\Nick\\LucidImagination\\LucidWorksSearch\\solr\\cores\\collection1_0\\data\\in
"lockID": "lucene-87258 8020481afad83452b87f7517d46",
        "readChunkSize": 104857600,
        "lockFactory": {
            "lockDir":
"C:\\Users\\Nick\\LucidImagination\\LucidWorksSearch\\solr\\cores\\collection1_0\\data\\in
"lockPrefix": null
        }
    },
    "collection_name": "collection1",
    "index_is_optimized": false,
    "index_size_bytes": 1318288,
    "free_disk_bytes": 13072003072,
    "index_max_doc": 0,
    "index_num_docs": 0,
    "index_version": 1300398945085,
    "index_is_curr ent": true,
    "root_dir": "C:\\",
    "instance_dir": "collection1_0",
    "total_disk_space": "222.7 GB",
    "total_disk_bytes": 239171792896
}
```

When you are satisfied that your data has been indexed, you are free to start executing searches.

Error Response Format

In an ideal world, all of your programming calls will work perfectly. Unfortunately, we do not live in an ideal world, and occasionally you will need to deal with error messages. LucidWorks returns errors as JSON maps that provide all of the information you need to determine the problem. They are in the following format:

```
{
   "http_status_name":{string},
   "http_status_code":{integer},
   "errors":[
      {
         "message":{string},
         "key":{string}
      },
      ...
   ]
}
```

These values correspond to the following information:

http_status_name: The name of the status code.

http_status_code: The integer status code that classifies the error. These integer codes correspond to standard HTTP response codes. For example, if you reference an object that does not exist, the error code will be "404", the traditional "Not Found" response. For more information, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

errors: This object contains more detailed information on the reasons for the error status, including:

message: A human-readable error message explaining the problem.

key: The input key that the message pertains to. This may be an empty string if the error does not correspond to a submitted key.

Example

```
{
   "http_status_name":"Unprocessable Entity",
   "http_status_code":422
   "errors":[
      {
         "message":"Unknown type:bad_type",
         "key":"type"
      },
      {
         "message":"start_time could not be parsed as a date",
         "key":"start_time"
      }
   ]
}
```

Note that more than one error may be passed in an error response.

Version

The version API shows the LucidWorks and Solr version and build information. This information should be supplied to support when requesting assistance, as it will help identify the version being used and will speed resolution of any problems.

- API Entry Point
- Get Version Information

API Entry Point

`/api/version`: show the version information

Get Version Information

```
GET /api/version
```

Input

Path Parameters

None.

Query Parameters

None.

Output

Output Content

A JSON list of version information in two sections labeled **solr** and **lucidworks**.

The Solr section contains the following information about the version of Solr embedded with LucidWorks:

Key	Description
svn.repo	Address of the Subversion repository Solr was taken from.
build.user	The user who created the build.
git.repo	Address of the Git repository.
build.date	The date the build was made.
solr.svn.rev	The last revision ID of the Solr build.
git.commit	The last commit ID of the Solr build.

build.host	The machine that created the build.
build.os	The operating system that created the build.

The LucidWorks section contains the following information about the LucidWorks software:

Key	Description
environment	Describes how the build was made.
version	The LucidWorks version number.
build.user	The user who created the build.
hudson.build.number	The build number.
build.date	The date the build was made.
git.commit	The last commit ID of the LucidWorks build.
build.host	The machine that created the build.
build.os	The operating system that created the build.

Response Codes

200: OK

Examples

Input

```
curl 'http://localhost:8888/api/version'
```

Output

```
{
    "lucidworks": {
        "build.date": "2012/03/13 12:53",
        "build.host": "dev.lcimg.com",
        "build.os": "Mac OS X",
        "build.user": "hudson",
        "environment": "production",
        "git.commit": "65e9f82178a601c89fdee4f357a3ed44c78e5743",
        "hudson.build.number": "3462",
        "version": "2.1"
    },
    "solr": {
        "build.date": "2012/03/12 10:34",
        "build.host": "beast",
        "build.os": "Linux",
        "build.user": "rmuir",
        "git.commit": "79e27b53705305866239f7206d92912a9c58c289",
        "git.repo": "git@github.com:lucidimagination/lucene-solr.git",
        "solr.svn.rev": "1296914",
        "svn.repo": "http://svn.apache.org/repos/asf/lucene/dev/trunk"
    }
}
```

Collections

Data in LucidWorks Search is organized into Collections. A collection contains data that is logically distinct from data in other collections. Collections have their own Data Sources, Settings, Fields, and Role Mappings. In other words, collections are distinct except that they happen to run in the same instance of LucidWorks Search.

For more information about collections, see also Working with Collections.

API Entry Points

`/api/collections`: list collections or create a new collection
`/api/collections/collection`: Get details or remove a collection

List Collection Names

```
GET /api/collections
```

Input

Path Parameters

None

Query Parameters

None

Output

Output Content

A JSON List of Maps mapping Collection keys to values.

Key	Type	Description
name	string	The name of the collection.
instance_dir	string	Advanced: *<filepath>* . Displays the directory name of the collection in `$LWE_HOME/data/solr/cores`.

Response Codes

200: success ok

Examples

Get a list of all collections and their locations:

Input

```
curl http://localhost:8888/api/collections
```

Output

```
[
    {
        "name": "new_collection",
        "instance_dir": "new_collection_1"
    },
    {
        "name": "collection 1",
        "instance_dir": "collection1_0"
    },
    {
        "name": "social",
        "instance_dir": "socialdata"
    }
]
```

Create Collection

```
POST /api/collections
```

Input

Path Parameters

None

Query Parameters

None

Input content

JSON block with all keys.

Key	Type	Required	Default	Description
name	string	Yes	null	The name of the collection.
template	string	No	None	The name of the collection template to use while creating the collection. To use the default set of LucidWorks configuration files, enter **default.zip**. See Using Collection Templates for more information on how to create and use collection templates.

instance_dir	string	No	None	Advanced: *<filepath>* Allows you to optionally override the name and/or location of the Solr instance directory for the collection. By default, this will be the next available collection *n* directory in `$LWE_HOME/data/solr/cores`. File paths that are not absolute will be relative to `$LWE_HOME/data/solr/cores` .
num_shards	integer	No	None	Used only when running in SolrCloud mode, this attribute is used to specify the number of shards to split the index to after the collection is created. This will register the new collection and its configuration files with ZooKeeper, and share the configurations with the other shards. This number should match the number of shards running to avoid errors, but it is possible to create a collection and start more shards later.

⚠ Because collection templates are based on an instance_dir, it's recommended to specify either parameter when creating a new collection, not both.

Output

Output Content

JSON representation of new collection

Key	Type	Description
instance_dir	string	The name of the directory that was created under `$LWE_HOME/conf/solr/cores` that contains the configuration files.
name	string	The name of the collection.

Return Codes

201: created

Examples

Create a new collection called "social" and specify that the collection should be split across two shards of a SolrCloud cluster.

Input

```
curl -H 'Content-type: application/json' -d
'{"name":"social","num_shards":2,"template":"default.zip"}'
   http://localhost:8888/api/collections
```

Output

```
{
    "instance_dir": "social_1",
    "name": "social"
}
```

List Information for Specific Collection

`GET /api/collections/collection`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Output

Output Content

JSON representation of the collection.

Key	Type	Description
name	string	The name of the collection.
instance_dir	string	The collection directory name relative to `$LWE_HOME/solr/cores`.

Response Codes

200: success ok

Examples

Get the collection information for the "social" collection:

Input

```
curl http://localhost:8888/api/collections/social
```

Output

```
{
   "name":"social",
   "instance_dir":"socialdata"
}
```

Delete Collection

`DELETE /api/collections/collection`

> To delete the index for a collection without deleting the entire collection, see the Collection Index Delete API.

Deleting a collection will delete all associated indexes and settings, but not alerts or manually created users. Use the Alerts API or the Users API to delete those.

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

None

Output

Output Content

None

Return Codes

204: success no content

404: not found

Examples

Delete the "social" collection:

Input

```
curl -X DELETE http://localhost:8888/api/collections/social
```

Output

None.

Related Topics

- Working with Collections
- Using Collection Templates
- Using SolrCloud in LucidWorks

Collection Info

The Collection Info API can be used to retrieve some statistics about a particular collection. You can use this API to retrieve all data for a collection, or just a specific key, such as the `data_dir` or `free_disk_space`.

- API Entry Points
- Get All Information About a Collection
- Get Specific Information About a Collection

API Entry Points

`/api/collections/collection/info`: get all info about the collection.

`/api/collections/collection/info/name`: get specific info about the collection.

Get All Information About a Collection

`GET /api/collections/collection/info`

Returns all information about the collection.

Input

Path Parameters

Key	Description
collection	the collection name

Query Parameters

None

Output

Output Content

A JSON map of collection information keys mapped to their values.

Key	Type	Description
collection_name	string	The name of collection.
data_dir	string	The directory where the index and other data resides.
free_disk_space	string	A human-readable string indicating the amount of free disk space on the partition where the index resides.
free_disk_bytes	64-bit integer	The total number of bytes available on the partition where the index resides.
index_is_current	boolean	Is **true** unless the index on disk has changes not yet visible by this instance.
index_directory	string	The current Directory implementation being used for this index.
index_has_deletions	boolean	Is **true** if the index has deleted documents since the last optimization.
index_last_modified	date string	The date and time that the index was last modified (1-second resolution).
index_max_doc	64-bit integer	The largest doc ID in the index.
index_num_docs	64-bit integer	The number of documents in the index.
index_is_optimized	boolean	Is **true** if the index was optimized after the last document was indexed.
index_size	64-bit integer	A human-readable string indicating the total size of the index.
index_size_bytes	64-bit integer	The exact size of the index in bytes.

index_version	64-bit integer	A version stamp for the index.
instance_dir	string	The home directory for the collection.
root_dir	string	The root directory of the partition on which the index resides.
total_disk_space	string	A human-readable string indicating the amount of total disk space on the partition where the index resides.
total_disk_bytes	64-bit integer	The number of total bytes on the partition where the index resides.

Response Codes

200: OK

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/info'
```

Output

```
{
    "free_disk_space": "10.7 GB",
    "index_last_modified": "2011-03-18T03:46:59+0000",
    "index_has_deletions": true,
    "data_dir":
"C:\\Users\\Nick\\LucidImagination\\LucidWorksEnterpriseDocs\\bin\\..\\solr\\cores\\collec
"index_size": "3.8 MB",
    "index_directory": "org.apache.lucene.store.MockDirectoryWrapper",
    "collection_name": "collection1",
    "index_is_optimized": false,
    "index_size_bytes": 3990150,
    "free_disk_bytes": 11491110912,
    "index_max_doc": 169,
    "index_num_docs": 136,
    "index_version": 1300398945104,
    "index_is_current": true,
    "root_dir": "C:\\",
    "instance_dir": "collection1_0",
    "total_disk_space": "222.7 GB",
    "total_disk_bytes": 239171792896
}
```

Get Specific Information About a Collection

```
GET /api/collections/collection/info/name
```

Input

Path Parameters

Key	Description
collection	the collection name
name	the name of the info key

Query Parameters

None

Input Content

Key	Type	Description
collection_name	string	The name of collection.
data_dir	string	The directory where the index and other data resides.
free_disk_space	string	A human-readable string indicating the amount of free disk space on the partition where the index resides.
free_disk_bytes	64-bit integer	The total number of bytes available on the partition where the index resides.
index_is_current	boolean	Is **true** unless the index on disk has changes not yet visible by this instance.
index_directory	string	The current Directory implementation being used for this index.
index_has_deletions	boolean	Is **true** if the index has deleted documents since the last optimization.
index_last_modified	date string	The date and time that the index was last modified (1-second resolution).
index_max_doc	64-bit integer	The largest doc ID in the index.
index_num_docs	64-bit integer	The number of documents in the index.
index_is_optimized	boolean	Is **true** if the index was optimized after the last document was indexed.
index_size	64-bit integer	A human-readable string indicating the total size of the index.

index_size_bytes	64-bit integer	The exact size of the index in bytes.
index_version	64-bit integer	A version stamp for the index.
instance_dir	string	The home directory for the collection.
root_dir	string	The root directory of the partition on which the index resides.
total_disk_space	string	A human-readable string indicating the amount of total disk space on the partition where the index resides.
total_disk_bytes	64-bit integer	The number of total bytes on the partition where the index resides.

Output

Output Content

A JSON map of Collection information keys mapped to their values. For a list of the keys, see GET: Output Content.

Response codes

200: OK

404: Not Found

Tip

When requesting collection info, you can pass a comma-separated list of keys (such as index_current,index_version) rather than separate requests for each key.

Examples

Request the number of documents and index version for the collection.

Input

```
curl
'http://localhost:8888/api/collections/collection1/info/index_version,index_num_docs'
```

Output

```
{
    "index_num_docs":136,
    "index_version":1300398945104
}
```

Collection Templates

This API lists available collection templates for use in creating new collections.

Each template is a .zip file that consists of LucidWorks configuration files. The default LucidWorks configuration is available as `default.zip` found in `$LWE_HOME/app/collection_templates`. The default configuration can be customized as needed and put in a .zip archive for use when creating new collections in the future. The .zip file can have any name, including `default.zip`, although using the same name would overwrite the system default template, meaning it would not be available at a later time if needed. All templates must be placed in `$LWE_HOME/conf/collection_templates` to be available during collection creation.

For more information about creating templates, see Using Collection Templates.

Information for LucidWorks Search in the Cloud Users
You can create custom templates with LucidWorks Search **on-premise only**. Customers using LucidWorks Search hosted in AWS or Azure have two collection templates out of the box that can be accessed using this API, but there is no mechanism to create and upload custom templates.

API Entry Points

`/api/collectiontemplates`: list available collection templates

List Collection Templates

`GET /api/collectiontemplates`

Input

Path Parameters

None.

Query Parameters

None.

Output

Output Content

A JSON array of available template file names.

Response Codes

200: OK

Examples

Input

```
curl 'http://localhost:8888/api/collectiontemplates'
```

Output

```
["default.zip"]
```

Collection Index Delete

The Collections Index API is used to delete the index for an entire collection or only documents associated with a specific data source.

- API Entry Points
- Delete the Index for a Collection
- Delete Indexed Data for a Data Source

API Entry Points

`/api/collections/collection/index`: delete the collection's index
`/api/collections/collection/datasources/datasource/index`: delete indexed content for the data source

Delete the Index for a Collection

`DELETE /api/collections/collection/index`

Stops all running data sources, clears the main search index for the collection, and deletes all crawl history for the collection.

Input

Path Parameters

Key	Description
collection	The collection name

Query Parameters

Key	Description
key	Always set to iaccepttherisk

Output

Output Content

None

Response Codes

204: No Content

Examples

Input

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/index?key=iaccepttherisk
```

Output

None.

Delete Indexed Data for a Data Source

`DELETE /api/collections/collection/datasources/id/index`

Stops the specified data source, deletes all documents from the collection's search index, and deletes all history from crawling activities. If the data source is SolrXML it must have been configured to include data source metadata before the contents can be deleted with this API. Otherwise, you will need to delete all documents in the collection to delete these documents.

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Output

Output Content

None

Response Codes

204: No Content

Examples

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/datasources/8/index
```

Output

None.

Activities

Activities are scheduled events that perform intensive operations on a collection, such as optimizing the index or creating the auto-complete index. Typically, schedules are recurring so that LucidWorks Search performs these activities periodically. However, activities can also be started at with this API for a single run. Activities controlled by this API are:

- Optimizing the index
- Indexing autocomplete data
- Processing click-scoring boost data

API Entry points

`/api/collections/collection/activities`: get a list of activities, or create a new one
`/api/collections/collection/activities/id`: view, delete or update an existing activity

Get a List of Activities

`GET /api/collections/collection/activities`

Note that activities that have never been run will not show in this list.

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Output

Output Content

A list of activities. For each activity, the fields are:

Key	Type	Description
start_time	date string	The start date and time for this schedule, in the format `yyyy-MM-dd'T'HH:mm:ss'+/-'hhmm`. The '+/-' is adjusted for the time zone relative to UTC.
period	64-bit integer	The number of seconds in between repeated invocations of this schedule; set to 0 if this should only occur once.
type	string	The type of activity: **optimize**, **click**, **autocomplete**.
active	boolean	If **true**, this schedule will be run at the next scheduled time.

Activity Types

Activity	Description
optimize	Optimizes the index.
click	Processes logs of user clicks to calculate boost values.
autocomplete	Runs the auto-complete source build phase.

Return Codes

200: OK

Examples

Get a list of all the active activities for the collection.

Input

```
curl 'http://localhost:8888/api/collections/collection1/activities'
```

Output

```
[
    {
        "id": 9,
        "active": true,
        "start_time": "2011-03-18T04:44:39+0000",
        "type": "optimize" ,
        "period": 86400
    },
    {
        "id": 11,
        "active": true,
        "start_time": "2011-03-1 8T04:45:03+0000",
        "type": "autocomplete",
        "period": 86400
    }
]
```

Create an Activity

POST /api/collections/collection/activities

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input Content

Key	Type	Required	Default	Description

start_time	date string	Yes	null	The start date and time for this schedule, in the format `yyyy-MM-dd'T'HH:mm:ss'+/-'hhmm`. The API can accept a relative time of "now". The 'T' is entered without quotes. The '+/-' is for the time zone relative to UTC, and also entered without quotes. If preferred, an integer can be entered instead, which will calculate a start time based on that number of seconds from "now". So, for example, `"start_time":120` would start the activity 2 minutes (120 seconds) from when the activity was scheduled.
period	64-bit integer	No	0	The number of seconds in between repeated invocations of this schedule; set to 0 if this should only occur once.
type	string	Yes	null	The type of activity: **optimize**, **click**, **autocomplete**.
active	boolean	No	false	If **true**, this schedule will be run at the next scheduled time.

Output

Output Content

JSON representing the new resource.

Return Codes

201: Created

Examples

Input

```
curl -H 'Content-type: application/json' -d '
{
  "period": 0,
  "type": "optimize",
  "start_time": "2011-03-29T12:10:32-0700",
  "active": true
}' 'http://localhost:8888/api/collections/collection1/activities'
```

Output

```
{
   "id":19,
   "active":true,
   "start_time":"2011-03-29T19:10:32+0000",
   "type":"optimize",
   "period":0
}
```

View a Specific Activity

```
GET /api/collections/collection/activities/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The activity's ID.

Query Parameters

None

Input Content

None

Output

Output Content

Key	Type	Description
start_time	date string	The start date and time for this schedule, in the format `yyyy-MM-dd'T'HH:mm:ss'+/-'hhmm`. The '+/-' is adjusted for the time zone relative to UTC.
period	64-bit integer	The number of seconds in between repeated invocations of this schedule; set to 0 if this should only occur once.
type	string	The type of activity: **optimize**, **click**, **autocomplete**.
active	boolean	If **true**, this schedule will be run at the next scheduled time.

Response Codes

200: OK

404: Not Found

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/activities/19'
```

Output

```
{
   "id":19,
   "active":true,
   "start_time":"2011-03-29T19:10:32+0000",
   "type":"optimize",
   "period":0
}
```

Update an Activity's Schedule

```
PUT /api/collections/collection/activities/id
```

Input

Path Parameters

Key	Description
collection	The collection name
id	The activity's ID.

Query Parameters

None

Input Content

Key	Type	Description

start_time	date string	The start date and time for this schedule, in the format `yyyy-MM-dd'T'HH:mm:ss'+/-'hhmm`. The API can accept a relative time of "now". The 'T' is entered without quotes. The '+/-' is for the time zone relative to UTC, and also entered without quotes. If preferred, an integer can be entered instead, which will calculate a start time based on that number of seconds from "now". So, for example, `"start_time":120` would start the activity 2 minutes (120 seconds) from when the activity was scheduled.
period	64-bit integer	The number of seconds in between repeated invocations of this schedule; set to 0 if this should only occur once.
type	string	The type of activity: **optimize**, **click**, **autocomplete**.
active	boolean	If **true**, this schedule will be run at the next scheduled time.

Activity Types

Activity	Description
optimize	Optimizes the index.
click	Processes logs of user clicks to calculate boost values.
autocomplete	Runs the auto-complete source build phase.

Output

Output Content

None

Return Codes

204: No Content

Examples

Set the server to optimize the index once an hour.

Input

```
curl -X PUT -H 'Content-type: application/json' -d '
{
  "period": 6000
}' 'http://localhost:8888/api/collections/collection1/activities/19'
```

Output

None. (Check properties to confirm changes.)

Delete a Scheduled Activity

`DELETE /api/collections/name/activities/id`

Input

Path Parameters

Key	Description
collection	The collection name
id	The activity's ID

Query Parameters

None

Output

Output Content

None

Return Codes

204: No Content

404: Not Found

Examples

Input

```
curl -X DELETE 'http://localhost:8888/api/collections/collection1/activities/19'
```

Output

None. (Check properties to confirm changes)

Related Topics

- Click Scoring Relevance Framework
- Auto-Complete of User Queries

Activity Status

The Activities Status API allows you to get information about whether or not an Activity is currently running.

API Entry points

`/api/collections/collection/activities/id/status`: get the status of an activity.

Get the Current Status of an Activity

```
GET /api/collection/collection/activities/id/status
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The activity ID. Use 'all' for all activities.

Query Parameters

None

Input Content

None

Output

Output Content

Key	Type	Description
id	int	The activity ID.
running	boolean	**True** indicates the activity is currently running.
type	string	The activity type. Possible types are **click**, **autocomplete**, or **optimize**.

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/activities/2/status'
```

Output

While the activity is running:

```
{
     "id" : 2,
     "running" : true,
     "type" : "optimize"
}
```

Once process is finished, and the activity is idle:

```
{
     "id" : 2,
     "running" : false,
     "type" : "optimize"
}
```

Activity History

The Activity History API provides a means to get the historical statistics for previous Activity runs.

- API Entry points
- Get the History of a Data Source

API Entry points

`/api/collections/name/activities/id/history`: get statistics for the last 50 runs of the given Activity.

Get the History of a Data Source

`GET /api/collections/collection/activities/id/history`

Input

Path Parameters

Key	Description
collection	The collection name.
id	The activity ID. Use 'all' for all activities.

Query Parameters

None

Input Content

None

Output

Output Content

Key	Type	Description
history	JSON map	Contains the following two fields (`activity_started` and `activity_finished`) in a JSON map.
activity_started	date string	When the activity began.
activity_finished	date string	When the activity finished.
id	integer	The ID of the activity, if the API call was not for a specific activity.

Examples

Get a history of all activities:

Input

```
curl 'http://localhost:8888/api/collections/collection1/activities/all/history'
```

Output

```
[
    {
        "history": [
            {
                "activity_finished": "2011-03-18T04:44:39+0000",
                "activity_started": "2011-03-18T04:44:39+0000"
            }
        ],
        "id": 9
    },
    {
        "history": [
            {
                "activity_finished": "2011-03-18T0 4:44:44+0000",
                "activity_started": "2011-03-18T04:44:44+0000"
            }
        ],
        "id": 10
    },
    {
        "history" : [
            {
                "activity_finished": "2011-03-18T04:45:03+0000",
                "activity_started": "2011-03-18T0 4:45:03+0000"
            }
        ],
        "id": 11
    }
]
```

Get the history for activity number 9:

Input

```
curl 'http://localhost:8888/api/collections/collection1/activities/9/history'
```

Output

```
[
    {
        "activity_finished": "2011-03-18T04:44:39+0000",
        "activity_started": "2011-03-18T04:44:39+0000"
    }
]
```

Data Sources

Data sources are conduits by which LucidWorks acquires new content. Data sources describe the target repository of documents and how to access documents in the repository. This description is then used to create a crawl job to be executed by a specific crawler implementation (called "crawler controllers").

In this section:

A data source is defined by selecting a crawler controller, then specifying a valid type for that crawler. Some crawlers work with several types, other crawlers only support one type. It is important to match the correct crawler with the type of data source to be indexed.

Each crawler and specified type has different supported attributes. A few attributes are common across all crawlers and types, and some types share attributes with other types using the same crawler. Review the supported attributes carefully when creating data sources with the API.

When working with data sources and their content, it helps to understand how content is handled during the initial crawl and in subsequent re-crawls to update the index with new, updated, or removed content. Most of the crawlers keep track of documents that have been "seen" before which helps speed later crawls by not processing unchanged content, but it can be confusing if the configuration settings change between crawls. In some cases, you may need to remove the crawl history in order to get the results you want; an example of this would be the `add_failed_docs` setting - if it is not set for the initial crawl of a repository, it will be skipped on subsequent crawls unless it has been modified in some way. Other examples include (but aren't limited to) settings to map fields from the incoming documents to another field, options to add LucidWorks-specific fields to the documents, as well as changes to fields themselves and any dynamic field rules. If making changes to a data source configuration after content has already been crawled and indexed, review the options in the section on Reindexing Content for possible approaches.

At present, the LucidWorks Search comes with the following built-in crawler controllers that support the following kinds of data sources:

Crawler Controller	Symbolic Name	Data Source Types Supported
Aperture-based crawlers	lucid.aperture	• Local file systems (LucidWorks Search on-premise only) • Websites

DataImportHandler-based JDBC crawler	lucid.jdbc	• JDBC databases (LucidWorks Search on-premise only)
SolrXML crawler	lucid.solrxml	• Solr XML files (LucidWorks Search on-premise only)
Google Connector Manager-based crawler	lucid.gcm	• Microsoft SharePoint servers (Microsoft Office SharePoint Server 2007, Microsoft Windows SharePoint Services 3.0, SharePoint 2010)
Remote file system and pseudo-file system crawler	lucid.fs	• SMB / CIFS (Windows Shares) filesystems • Hadoop Distributed File Systems (HDFS) • Amazon S3 buckets (also known as "S3 native") • HDFS over Amazon S3 • FTP servers
External data	lucid.external	• Externally generated data pushed to LucidWorks via Solr
Twitter stream	lucid.twitter.stream	• Twitter Streams using Twitter's stream API
High-Volume HDFS	lucid.map.reduce.hdfs	• High Volume crawling of Hadoop File Systems

API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes to values. The exact set of attributes for a particular data source depends on the `type`. There is, however, a set of attributes common to all data source types. Specific attributes are discussed in sections for those types, linked below.

Common Attributes

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

Attributes for Specific Data Source Types

Each data source type has attributes specifically for that type. To find the attributes for a specific data source type, see the API documentation for that type. The types are:

- Website
- Local Filesystem
- Databases with JDBC
- SolrXML
- Hadoop Distributed Filesystem
- Amazon S3 Bucket
- Hadoop over S3
- FTP Server
- Local Filesystem (with thread control)
- Twitter Stream
- High-Volume HDFS
- External Push

Response Codes

200: Success OK

Examples

Input

```
curl http://localhost:8888/api/collections/collection1/datasources
```

Output

JSON format of all configured data sources.

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name

Query Parameters

None

Input content

JSON block with all attributes. The ID field, if present, will be ignored. See attributes in section on getting a list of data sources.

Output

Output Content

JSON representation of new data source. Attributes returned are listed in the section on getting a list of data sources.

Return Codes

201: created

Examples

Create a data source that includes the content of the LucidWorks web site. To keep the size down, only crawl two levels, and do not index the blog tag links or any search links. Also, do not wander off the site and index any external links.

Input

```
curl -H 'Content-type: application/json' -d '{
 "crawler":"lucid.aperture",
 "type":"web",
 "url":"http://www.lucidworks.com/",
 "crawl_depth":1,
 "name":"Lucid Website"
   }'
http://localhost:8888/api/collections/collection1/datasources
```

Output

```
{
        "add_failed_docs": false,
        "auth": [],
        "bounds": "tree",
        "caching": false,
        "category": "Web",
        "collection": "collection1",
        "commit_on_finish": true,
        "commit_within": 900000,
        "crawl_depth": 1,
        "crawler": "lucid.aperture",
        "exclude_paths": [],
        "fail_unsupported_file_types": false,
        "id": 3,
        "ignore_robots": false,
        "include_paths": [],
        "indexing": true,
        "log_extra_detail": false,
        "mapping": {
            "datasource_field": "data_source",
            "default_field": null,
            "dynamic_field": "attr",
            "literals": {},
            "lucidworks_fields": true,
            "mappings": {
                "acl": "acl",
                "author": "author",
                "batch_id": "batch_id",
                "body": "body",
                "content-encoding": "characterSet",
                "content-length": "fileSize",
                "content-type": "mimeType",
                "contentcreated": "dateCreated",
                "contentlastmodified": "lastModified",
                "contributor": "author",
                "crawl_uri": "crawl_uri",
                "created": "dateCreated",
                "creator": "creator",
                "date": null,
                "description": "description",
                "filelastmodified": "lastModified",
                "filename": "fileName",
                "filesize": "fileSize",
                "fullname": "author",
                "fulltext": "body",
                "keyword": "keywords",
                "last-modified": "lastModified",
                "last-printed": null,
                "lastmodified": "lastModified",
                "lastmodifiedby": "author",
                "links": null,
                "messagesubject": "title",
                "mimetype": "mimeType",
                "name": "title",
                "page-count": "pageCount",
                "pagecount": "pageCount",
                "plaintextcontent": "body",
                "plaintextmessagecontent": "body",
                "slide-count": "pageCount",
                "slides": "pageCount",
                "subject": "subject",
                "title": "title",
                "type": null,
                "url": "url"
            },
            "multi_val": {
                "acl": true,
                "author": true,
                "body": false,
                "dateCreated": false,
                "description": false,
                "fileSize": false,
                "mimeType": false,
                "title": false
            },
            "original_content": false,
            "types": {
                "date": "DATE",
                "datecreated": "DATE",
                "filesize": "LONG",
                "lastmodified": "DATE"
            },
            "unique_key": "id",
            "verify_schema": true
        },
        "max_bytes": 10485760,
        "max_docs": -1,
        "name": "Lucid Website",
        "parsing": true,
        "proxy_host": "",
        "proxy_password": "",
        "proxy_port": -1,
        "proxy_username": "",
        "type": "web",
        "url": "http://www.lucidworks.com/",
        "verify_access": true,
        "warn_unknown_mime_types": false
    }
```

Get Data Source Details

`GET /api/collections/collection/datasources/id`

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name

Query Parameters

Key	Type	Description
id	string	The data source ID

Input content

None

Output

Output Content

Attributes returned are listed in the section on getting a list of data sources.

Return Codes

200: success ok

404: not found

Examples

Get all of the parameters for data source 6, created in the previous step.

Input

```
curl http://localhost:8888/api/collections/collection1/datasources/6
```

Output

```
{
    "id": 6,
    "max_bytes": 10485760,
    "include_paths": [
        "http://www\\.lucidworks\\.com/.*"
    ] ,
    "collect_links": true,
    "exclude_paths": [
        "http://www\\.lucidworks\\.com/blog/tag/.*",
        "http://www\\.lucidworks\\.com/search\\?.*"
    ],
    "mapping": {
        "multi_val ": {
            "fileSize": true,
            "body": true,
            "author": true,
            "title": true,
            "keywords": true,
            "subject": true,
            "description": false,
            "fileName": true,
            "dateCreated": false,
            "attr": true,
            "creator": true
        },
        "default_field": null,
        "mappings": {
            "slide-count": "pageCount",
            "content- type": "mimeType",
            "body": "body",
            "slides": "pageCount",
            "subject": "subject",
            "plainte xtmessagecontent": "body",
            "lastmodifiedby": "author",
            "content-encoding": "character Set",
            "type": null,
            "date": null,
            "creator": "creator",
            "author": "author",
            "title": "titl e",
            "mimetype": "mimeType",
            "created": "dateCreated",
            "plaintextcontent": "body",
            "page count": "pageCount",
            "contentcreated": "dateCreated",
            "description": "description",
            "contributor": "author",
            "name": "title",
            "filelastmodified": "lastModified",
            "fullname" : "author",
            "fulltext": "body",
            "messagesubject": "title",
            "last-modified": "lastModified",
            "keyword": "keywords",
            "contentlastmodified": "lastModified",
            "last-printed": null,
            "links": null,
            "batch_id": "batch_id",
            "crawl_uri": "crawl_uri",
            "filesize": "fileSize",
            "page-count": "pageCount",
            "content-length": "fileSize",
            "filename": "fileName"
        },
        " dynamic_field": "attr",
        "types": {
            "filesize": "INT",
            "pagecount": "INT",
            "lastmodified": "DATE",
            "datecreated": "DATE",
            "date": "DATE"
        },
        "unique_key": "id",
        "datasource_field": "data_source"
    },
    "collection": "collection1",
    "type": "web",
    "url": "http://www.lucidworks.com/",
    "crawler": "lucid.aperture",
    "bounds": "tree",
    "category": "Web" ,
    "name": "LucidWorks Website",
    "crawl_depth": 2
}
```

Update a Data Source

`PUT /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	The collection name
id	The data source ID

Query Parameters

None

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (ID) cannot be updated.

Output

Output Content

None

Return Codes

204: success no content

Examples

Change the web data source so that it crawls three levels instead of just two:

Input

```
curl -X PUT -H 'Content-type: application/json' -d '
{
    "crawl_depth": 3
}' 'http://localhost:8888/api/collections/collection1/datasources/6'
```

Output

None (check properties to confirm changes).

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name
id	The data source ID

Query Parameters

None

Input content

None

Output

Output Content

None

Return Codes

204: success no content

404: not found

Examples

Input

```
curl -X DELETE  -H 'Content-type: application/json'
http://localhost:8888/api/collections/collection1/datasources/13
```

Output

None. Check the listing of data sources to confirm deletion.

Amazon S3 Data Sources

The Amazon S3 data source type allows crawling documents stored in an Amazon S3 bucket.

Amazon S3 Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an S3 data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.
type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem

crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.
category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.

output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description
mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.

default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

S3 Type-Specific Attributes

When creating a data source of `type` **s3**, the value **lucid.fs** must be supplied for the `crawler` attribute, described in the section on common attributes. In releases prior to v2.1, this data source type was called "S3N".

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document.
bounds	string	No	None	Either **tree** to limit the crawl to a filesystem sub-tree, or "none" for no limits. For example, if crawling `smb://10.0.0.50/docs`, the tree option is the equivalent of `smb://10.0.0.50/docs/*` and the lucid.fs would not crawl `smb://10.0.0.50/other`. If you require more advanced crawl limiting, you should choose **none** and using the includes or excludes options.
crawl_depth	32-bit integer	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if left blank.

exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. If left blank, all sub-directory paths will be followed (limited by the crawl_depth).
max_bytes	long	No	10,485,760	Defines the maximum size of a document to crawl. Optional, default is -1, which is 10Mb per document.
max_docs	integer	No	-1	Defines the maximum number of documents to crawl. If the value is -1, the crawler will collect all found documents (in conjunction with other data source attributes which may limit the crawl.
max_threads	integer	No	1	Defines the maximum number of threads the crawler should use. Optional, default is 1, which is single-threaded.
password	string	Yes	Null	Your Amazon S3 SecretAccessKey
type	string	True	Null	One of supported data source types, **must** be consistent with the root URL's protocol. For an Amazon S3 data source type, use **s3**.

url	string	Yes	Null	For S3 (Amazon S3 native), the root URL is a fully-qualified URL that starts with the `s3` protocol, the name of the bucket, and the path inside the bucket. All URLs that point to directories that contain files to be indexed must end with a trailing slash ("/"), which is a new requirement for v2.1. Both `AccessKeyId` and `SecretAccessKey` are needed: submit `AccessKeyId` as the username and `SecretAccessKey` as the password. You can also pass these credentials as part of the URL in the following format: `s3n://<username>@<password>:bucket/path/`. However, Amazon S3 credentials often contain characters that are not allowed in URLs. In that case, you must pass these credentials by setting the `username` and `password` properties explicitly.
username	string	Yes	Null	Your Amazon S3 AccessKeyId.
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

✓ The `include_paths` and `exclude_paths` attributes for all Remote File System data sources use Java Regular Expressions.

Example S3 data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "none",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.fs",
    "exclude_paths": [],
    "id": "02156ee7cfd24eb6b68953fb851e000a",
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 1,
    "name": "S3 Crawler",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "password": "AWS-SecretToken",
    "type": "s3",
    "url": "s3://s3crawler/test1/",
    "username": "AWS-AccessKey",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources:` list or create data sources in a particular collection

`/api/collections/collection/datasources/id:` update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Path Parameters

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Database Data Sources

This functionality is **not available** with LucidWorks Search on AWS or Azure

Databases can be indexed with the Database data source. The database must be accessible to the LucidWorks Search server, and they must have a valid JDBC driver available to the crawler that is appropriate for your RDBMS. Drivers are not shipped with LucidWorks, but can be loaded with either the JDBC Drivers API or the Admin UI. JDBC database data sources are available in **LucidWorks Search on-premise only**.

JDBC Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a JDBC data source.

The page Data Source does not exist.

JDBC Type-Specific Attributes

When creating a data source of `type` **jdbc**, the value **lucid.jdbc** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
delta_sql_query	string	No	Null	This allows incremental indexing, which will only retrieve updated records in subsequent crawls without having to retrieve the entire contents of an unconstrained query. Delta queries select only primary key values of your data and must include last_modified condition in the following form, "`SELECT id FROM table WHERE last_modified > $`". The "$" sign will hold the last successful import time from the database. If you do not use the "$" character, the query will fail.
driver	string	Yes	Null	The class name of the JDBC driver.
max_docs	integer	No	-1	The maximum number of documents to crawl. If set to -1, all document found will be collected (in conjunction with other data source attributes which may limit the crawl).

nested_queries	list of strings	No	Null	If you want to index one-to-many or many-to-many relations in addition to plain rows then you can specify an arbitrary number of additional SQL "nested" queries. For example, if you want to index the list of assigned tags for your documents you can specify a query in the following form `"SELECT tag FROM tag INNER JOIN document_tag ON document_tag.tag_id=tag.id WHERE document_tag.doc_id=$"`. The $ sign will hold the primary key of your main data row. This query will be executed for every record of your data and corresponding list of tags will be retrieved from database and indexed in the Solr index.
password	string	Yes	Null	The password of the account that should be used to access the data.
primary_key	string	No	Null	The column name of the primary key.
sql_select_statement	string	Yes	Null	The select statement to use to generate data.
username	string	Yes	Null	The username of a database account that should be used to access the data.
url	string	Yes	Null	A URI to the database. This should be in the form of `jdbc:mysql://localhost/test`. Database data sources are expected to be unique by URI, which means that creating two data sources for the same URI is not possible.
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

Example JDBC data source (without mapping attributes):

```
{
    "category": "Jdbc",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawler": "lucid.jdbc",
    "delta_sql_query": "",
    "driver": "com.mysql.jdbc.Driver",
    "id": 4,
    "mapping": {
        ...
    },
    "max_docs": -1,
    "name": "stackoverflow db",
    "nested_queries": [],
    "output_args": "http://127.0.0.1:8888/solr/collection1",
    "output_type": "solr",
    "password": "password",
    "primary_key": "id",
    "sql_select_statement": "select id, comment_text as body from comment",
    "type": "jdbc",
    "url": "jdbc:mysql://127.0.0.1/stackoverflow",
    "username": "username",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

```
DELETE /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None
Output

Output Content

None

External Data Sources

External data sources are those which do not use LucidWorks crawlers to fetch content for indexing. Instead, they feed documents directly to Solr via another process, such as SolrJ. The purpose of adding it as a LucidWorks Search data source is to link LucidWorks features, such as the Admin UI or display of data sources as facets, to the documents.

Field mapping is supported with external data sources by adding the parameter `fm.ds=<id>` to the update request (field mapping attributes are described in the section on common attributes). All updates sent to LucidWorks via regular Solr APIs (SolrXML, CSV, JSON, or others that use an update chain with the field mapping processor) which specify this parameter will be processed by a field mapping processor, which will modify the incoming documents according to the field mapping definition.

If the ID supplied with the `fm.ds` parameter does not match an existing data source ID, an error will be returned and the documents will not be loaded. It is also possible to supply '-1' for the `fm.ds` ID. In cases where there is only one external type of data source, the fields for that data source will be filled in to the documents and the mapping for that data source will be applied. If there are zero external data sources or more than one external data source, an error will be returned and the documents will not be loaded.

The `fm.ds` parameter pushes the documents through the LucidWorks field mapping processor, but it does not associate the documents with the data source in terms of being able to display document counts in the Admin UI or data source-specific facets. To link the incoming documents to the LucidWorks data source, use the `lucidworks_fields` parameter to add LucidWorks specific data source fields (i.e., `data_source`, `data_source_name` and `data_source_type`) to the documents. This parameter is added in the field mapping definition, described in the section on common attributes. If this option is not selected, it may be difficult to identify the added documents if they need to be deleted later.

More information about how external data sources work is available in the section Suggestions for External Data Sources.

External Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an External data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

External Type-Specific Attributes

When creating a data source of `type` **external**, the value of **lucid.external** must be specified for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
callback	string	No	Null	A URL that will be used to start the crawl from within LucidWorks. This URL will be used to initiate the crawl when the data source is scheduled, so it should be constructed to include the same parameters (including the `fm.ds` parameter) that would be used to push the documents to Solr without LucidWorks.
name	string	Yes	Null	The name of the data source.
source	string	Yes	Null	Can be anything, but recommended use is for this to be an address or other type of location information to differentiate the external source.
source_type	string	Yes	Null	Can be anything, but recommended use is for this to be the type of source ("wiki" or "finance data") that helps identify it.

Sample External data source (without mapping attributes)

```
{
    "callback": "",
    "category": "External",
    "collection": "collection1",
    "crawler": "lucid.external",
    "id": "b9deb25341d0473193b23dea171c211d",
    "mapping": {
    ...
    },
    "name": "External documents",
    "output_args": "http://127.0.0.1:8888/solr/collection1",
    "output_type": "solr",
    "source": "Solr docs",
    "source_type": "Raw SolrXML",
    "type": "external",
    "url": "external:Solr docs"
}
```

Summary of API Endpoints

`/api/collections/collection/datasources:` list or create data sources in a particular collection

`/api/collections/collection/datasources/id:` update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

`PUT /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

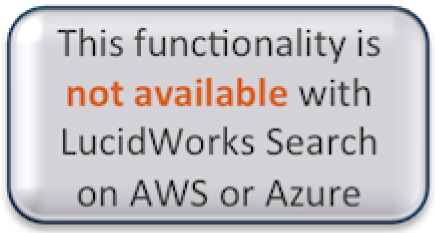
Input content

None
Output

Output Content

None

Filesystem Data Sources (Aperture)


This functionality is not available with LucidWorks Search on AWS or Azure

A local file system is one that is either on the same server as the LucidWorks installation or mounted to it. Other data source types exist for remote file systems, such as S3 or Windows Shares. Local file system data sources are available in **LucidWorks Search on-premise only**.

LucidWorks Search includes two data sources that can be used to crawl a local filesystem. This data source type differs from the other local filesystem data source type (referred to as a Simple Filesystem data source) in that this data source uses only one thread while crawling and all content is automatically parsed by the Aperture parser (no matter what the `parsing` attribute is set to). If processing content in batches, and you do not want the content parsed first, use the other filesystem data source type.

File Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a file data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.
type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem

crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.
category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.

output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description
mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.

default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

File Type-Specific Attributes

When creating a data source of `type` **file**, and intending a local filesystem, the value **lucid.aperture** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document. The default for this attribute is **false**.

bounds	string	No	None	Either **tree** to limit the crawl to a strict subtree, or **none** for no limits. If **tree** is chosen, the crawler will only access pages using the URL as the base path. For example, if crawling `/Path/to/Files`, the tree option is the equivalent of `/Path/to/Files/*` . When follow_links is true, choosing **none** will allow the crawl to go to directories outside the base directory path entered.
crawl_depth	32-bit integer	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth, which is also the default if left empty.
exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL. This can be used to exclude certain types of files from a crawl.
fail_unsupported_file_types	boolean	No	False	If **true**, documents that cannot be parsed (either because of unspecified errors or because of an unknown file format) will produce an error in the logs. The default behavior is to not report these documents as failures to the log. If `add_failed_docs` is also checked, the result of these failures will also be added to the index with whatever metadata could be extracted from the content. This is different from `warn_unknown_mime_types` in the sense that this parameter only applies to content that fails parsing and not content that doesn't have a defined mime type.

follow_links	boolean	No	False	Use **true** to instruct the crawler to follow symbolic links in the file system.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. This could be used to limit the crawl to only certain types of files or certain subdirectories.
max_bytes	long	No	10485760	Defines the maximum size of any crawled file. The default is -1, which is 10Mb per document.
path	string	Yes	Null	The path of the directory to start reading from. Paths should be entered as the complete directory path or they will be interpreted as relative to `$LWE_HOME`. On Unix systems, this means the path entered should start at root / level; on Windows, the drive letter and full path should be used (such as `C:\path`). Various types of relative paths, such as ../ or ~/, are not supported. Filesystem data sources are expected to be unique by URL, which means that creating two data sources for the same directory is not possible. The API will attempt to validate that LucidWorks Search can access the path, and will return an error if it cannot.
url	string	No	Null	Read-only value that shows the absolute path. In many cases this will be identical to the path as entered, but if the path was a shortcut or symbolic link, the url will show the real path.

verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.
warn_unknown_mime_types	boolean	No	False	If **true**, documents with no mime type specified in the format produce a warning in the log. If the file cannot be processed as plain text, it will be skipped and a warning message will be printed to the log. The default behavior is to skip these documents and not report warnings in the log. This parameter differs from `fail_unsupported_file_types` in the sense that it only applies to content that does not have a defined mime type.

✓ The include_paths and exclude_paths attributes for File System data sources use Java Regular Expressions.

Example file system data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "none",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.aperture",
    "exclude_paths": [],
    "fail_unsupported_file_types": false,
    "follow_links": false,
    "id": "9e3d6defb8d1438590c2aea02237db12",
    "include_paths": [],
    "indexing": true,
    "log_extra_detail": false,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "name": "Local hard drive",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "path": "/Users/cassandra4work/Documents",
    "type": "file",
    "url": "file:/Users/cassandra4work/Documents/",
    "verify_access": true,
    "warn_unknown_mime_types": false
}
```

Summary of API Endpoints

`/api/collections/collection/datasources:` list or create data sources in a particular collection

`/api/collections/collection/datasources/id:` update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

`POST /api/collections/collection/datasources`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

`GET /api/collections/collection/datasources/id`

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None
Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.
Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

FTP Data Sources

Only simple FTP is supported at this time. The FTP crawler does not support crawling FTP servers that require FTPS, SFTP or FTP over SSH.

By default, the FTP crawler uses passive mode; there is no setting to change to active mode at this time.

- Summary of API Endpoints
    - Get a List of Data Sources
    - Create a Data Source
    - Get Data Source Details
    - Update a Data Source
    - Delete a Data Source

FTP Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an FTP data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.
type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem

crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.
category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.

output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description
mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.

default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

FTP Type-Specific Attributes

When creating a data source of `type` **ftp**, the value **lucid.fs** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document.
bounds	string	No	None	Either **tree** to limit the crawl to a strict subtree, or "none" for no limits. For example, if crawling `smb://10.0.0.50/docs`, the tree option is the equivalent of `smb://10.0.0.50/docs*` and the lucid.fs would not crawl `smb://10.0.0.50/other`. If you require more advanced crawl limiting, you should choose **none** and using the includes or excludes options.
crawl_depth	32-bit integer	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if left blank.

exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. If left blank, all subdirectory paths will be followed (limited by the crawl_depth). This feature can be used to limit a filesystem crawl to specific subdirectories of a base directory path. For example, if the base directory path is `/Path/to/Files`, includes could be used to limit the crawl to subdirectories `/Path/to/Files/Archive/2010/*` and `/Path/to/Files/Archive/2011/*`.
max_bytes	long	No	10,485,760	Defines the maximum size of a document to crawl. Optional, default is -1, which is 10Mb per document.
max_threads	integer	No	1	Defines the maximum number of threads the crawler should use. Optional, default is 1, which is single-threaded. The FTP data source type does not support a max_threads value greater than 1.
password	string	No	Null	A password to access the FTP filesystem.
type	string	Yes	Null	One of supported data source types, **must** be consistent with the root URL's protocol. For a filesystem data source type, use **ftp**.
url	string	Yes	Null	For FTP access, the root URL includes the protocol, `ftp`, the host address, and the path to crawl: `ftp://_host_/path/`. By default, all files in the directory tree or folder hierarchy linked from this URL will be crawled, unless you select some of the limiting options available.
username	string	No	Null	A username to access the FTP filesystem.

verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

The include_paths and exclude_paths attributes for all Remote File System data sources use Java Regular Expressions.

Example FTP data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "tree",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.fs",
    "exclude_paths": [],
    "id": "b0a205c8076341dc8a251eb94e7ea129",
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 1,
    "name": "Pubmed Data",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "password": "crawl@example.com",
    "type": "ftp",
    "url": "ftp://ftp.ncbi.nih.gov/pubmed",
    "username": "anonymous",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

`POST /api/collections/collection/datasources`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Hadoop Over S3 Data Sources

This data source is similar to the Amazon S3 data source, but has some specific handlers for Hadoop in particular. It may be possible to configure an S3 data source to crawl a Hadoop File System over S3, but this S3H data source type is the preferred method.

This data source differs from the High Volume HDFS data source in that this data source type is designed to be polite and crawl through the content stored in Hadoop over S3 just as if it crawled a web site or any other file system. The High Volume HDFS, however, designed to take full advantage of the scaling abilities of Hadoop's Map-Reduce architecture.

Hadoop Over S3 (S3H) Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an S3H data source.

These attributes are used for all data source types (except where specifically noted).

▸

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

S3H Type-Specific Attributes

When creating a data source of `type` **s3h**, the value **lucid.fs** must be supplied for the `crawler` attribute, described in the section on common attributes. In releases prior to v2.1, this data source type was called "S3".

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document.
bounds	string	No	None	Either **tree** to limit the crawl to a strict subtree, or "none" for no limits. For example, if crawling `smb://10.0.0.50/docs`, the tree option is the equivalent of `smb://10.0.0.50/docs*` and the lucid.fs would not crawl `smb://10.0.0.50/other`. If you require more advanced crawl limiting, you should choose **none** and using the includes or excludes options.
crawl_depth	32-bit integer	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if not set.

exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty, then at least one pattern must match to include a file. If left blank, all sub-directory paths will be followed (limited by the crawl_depth).
max_bytes	long	No	10,485,760	Defines the maximum size of a document to crawl. Optional, default is -1, which is 10Mb per document.
max_docs	integer	No	-1	Defines the maximum number of documents to crawl. A value of -1 will crawl all found documents (in conjunction with other data source attributes which may limit the crawl).
max_threads	integer	No	1	Defines the maximum number of threads the crawler should use. Optional, default is 1, which is single-threaded.
password	string	Yes	Null	Your Amazon S3 SecretAccessKey
type	string	True	Null	One of supported data source types, **must** be consistent with the root URL's protocol. For a Hadoop over S3 data source type, use **s3h**.

url	string	Yes	Null	For S3 (Hadoop over Amazon), the root URL is a fully-qualified URL that starts with the `s3` protocol, the name of the bucket, and the path inside the bucket. All URLs that point to directories that contain files to be indexed must end with a trailing slash ("/"), which is a new requirement for v2.1. Both `AccessKeyId` and `SecretAccessKey` are needed: submit `AccessKeyId` as the `username` and `SecretAccessKey` as the `password`. You can also pass these credentials as part of the URL in the following format: `s3://<username>@<password>:bucket/path/`. However, Amazon S3 credentials often contain characters that are not allowed in URLs. In that case, you must pass these credentials by setting the `username}}and {{password` properties explicitly.
username	string	Yes	Null	Your Amazon S3 AccessKeyId
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

✓ The includes and excludes attributes for all Remote File System data sources use Java Regular Expressions.

Example S3H data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "tree",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.fs",
    "exclude_paths": [],
    "id": "c8a345d940a24bf6bfcdc1f65f4440d5",
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 1,
    "name": "Hadoop on S3",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "password": "AWS-SecretKey",
    "type": "s3h",
    "url": "s3://lucid-hadoop-s3-block/",
    "username": "AWS-AccessKey",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

HDFS Data Sources

LucidWorks has been tested with Hadoop v0.20.2 only at this time. Other releases labeled 0.20.2 are expected to use the same protocols and may work with LucidWorks, but have not been tested. Any release prior to 0.20.x or after 0.20.xxx is not expected to work as the protocols have changed.

This data source differs from the High Volume HDFS data source in that this data source type is designed to be polite and crawl through the content stored in HDFS just as if they it crawled a web site or any other file system. The High Volume HDFS, however, designed to take full advantage of the scaling abilities of Hadoop's Map-Reduce architecture.

HDFS Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an HDFS data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

HDFS Type-Specific Attributes

When creating a data source of `type` **hdfs**, the value **lucid.fs** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document.
bounds	string	No	None	Either **tree** to limit the crawl to a strict subtree, or "none" for no limits. For example, if crawling `hdfs://10.0.0.50/docs`, the tree option is the equivalent of `hdfs://10.0.0.50/docs/*` and the lucid.fs would not crawl `hdfs://10.0.0.50/other`. If you require more advanced crawl limiting, you should choose **none** and using the includes or excludes options.

converter	string	No	Null	Multi-record Hadoop files, such as SequenceFiles and MapReduceFiles, can be crawled with a sub-crawler to produce valid Solr documents (as many as there are Hadoop records in each fileset). The keys and values retrieved must be converted to SolrInputDocuments in order to be indexed. This can be done in the form of subclasses of com.lucid.crawl.fs.hdfs.Converter and specified by setting the value of the `converter` attribute to a fully qualified class name that extends Converter. If no value is specified, then a DefaultConverter will be used, which creates a single document per key-value pair and invokes the value's `toString()` method and adds this to the "body" field.
crawl_depth	32-bit integer	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if left blank.
exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. If left blank, all subdirectory paths will be followed (limited by the crawl_depth). This feature can be used to limit a filesystem crawl to specific subdirectories of a base directory path. For example, if the base directory path is `/Path/to/Files`, includes could be used to limit the crawl to subdirectories `/Path/to/Files/Archive/2010/*` and `/Path/to/Files/Archive/2011/*`.
max_bytes	long	No	10,485,760	Defines the maximum size of a document to crawl. Optional, default is 10,485,760 which is 10Mb per document.

max_docs	integer	No	-1	The maximum number of documents to crawl. The default is -1, which will collect documents found (in conjunction with the other data source attributes which may limit the number of documents).
max_threads	integer	No	1	Defines the maximum number of threads the crawler should use. Optional, default is 1, which is single-threaded. The FTP data source type does not support a max_threads value greater than 1.
type	string	True	Null	One of supported data source types, **must** be consistent with the root URL's protocol. For a Hadoop Filesystem data source type, use **hdfs**.
url	string	Yes	Null	A fully-qualified Hadoop file system URL, including the protocol (`hdfs`), host name and port of the namenode, and path of the target resource to crawl, such as `hdfs://namenode:9000/path/to/crawl`.
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

The includes and excludes attributes for all Remote File System data sources use Java Regular Expressions.

Example HDFS data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "none",
    "caching": false,
    "category": "FileSystem",
    "collection": "TestCollection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "converter": null,
    "crawl_depth": -1,
    "crawler": "lucid.fs",
    "exclude_paths": [],
    "id": 19,
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 10,
    "name": "HDFSTestSite",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "type": "hdfs",
    "url": "hdfs://hdfs.test.com:9000/",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.

id	The data source ID.

Query Parameters

None.

Input content

None
Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.
Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

High-Volume HDFS Data Sources

The High Volume HDFS (HV-HDFS) data source uses a Map-Reduce enabled crawler designed to leverage the scaling qualities of Apache Hadoop while indexing content into LucidWorks. In conjunction with LucidWork's usage of SolrCloud, applications should be able to meet their large scale indexing and search requirements.

To achieve this, HV-HDFS consists of a series of Map-Reduce enabled Jobs to convert raw content into documents that can be indexed into LucidWorks which in turn relies on the Behemoth project for Map-Reduce ready document conversion via Apache Tika and writing of documents to LucidWorks (we specifically use the LWE fork of this project).

The HV-HDFS data source is currently marked as "Early Access" and is thus subject to changes in how it works in future releases.

Before using the HV-HDFS Data Source type, please review the section Using the High Volume HDFS Crawler.

One feature that is not currently supported is field mapping. Because of this, most document metadata is dropped without being added to the index. Only the main text and the URL are retained.

HV-HDFS Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an HV-HDFS data source.

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.
type	string	Yes	Null	The type of this data source, **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem.
crawler	string	Yes	Null	Crawler implementation that handles this type of data source, **lucid.map.reduce.hdfs** for high_volume_hdfs type.
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.

name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.
category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

HV-HDFS Type-Specific Attributes

When creating a data source of `type` **high_volume_hdfs**, the value **lucid.map.reduce.hdfs** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
MIME_type	string	No	Null	Allows limiting the crawl to content MIME type. If a value is entered, Be skip Tika's MIME detection and proc content with the appropriate parser type should be entered in full, such "application/pdf" for PDF document

hadoop_conf	string	Yes	Null	The location of the Hadoop configu that contains the Hadoop `core-site` `mapred-site.xml` and other Hadoop files. This path must reside on the s as the LucidWorks server. Hadoop to be running on the same server a but the configuration directory mus from the LucidWorks server.
path	string	Yes	Null	The input path where your data res required that this be in HDFS to be the first step of the process conver one or more SequenceFiles in HDFS `hdfs://bob:54310/path/to/content` a `file:///path/to/local/content` wou valid inputs.
recurse	boolean	No	True	If **true**, the default, the crawler wil subdirectories of the input path. Se the crawler should stay within the t specified with the `path` attribute.
tika_content_handler	string	No	com.digitalpebble. tika.TikaProcessor	In most cases, the default, `com.digital-pebble.behemoth.tika.` , does not need to be changed. If y to change it, enter the fully qualifie of a Behemoth Tika Processor that extracting content from documents must be available on the classpath used by LucidWorks.
work_path	string	Yes	/tmp	A path to use for intermediate stor connector does not clean up tempo so it can be used in debugging, if n the job is complete, content stored temporarily can be safely deleted. `hdfs://bob:54310/tmp/hv_hdfs` woul path.
zookeeper_host	string	Yes	Null	The host and port where ZooKeepe and coordinates SolrCloud activity, `hostname:port`.

Example High Volume HDFS data source (without mapping):

```
{
    "MIME_type": "",
    "commit_on_finish": true,
    "mapping": {
      ...
    },
    "collection": "collection1",
    "work_path": "hdfs://example:54310/tmp",
    "type": "high_volume_hdfs",
    "recurse": true,
    "crawler": "lucid.map.reduce.hdfs",
    "id": 3,
    "category": "External",
    "tika_content_handler": "com.digitalpebble.behemoth.tika.TikaProcessor",
    "zookeeper_host": "allie:9888",
    "name": "Example",
    "path": "/Users/projects/content/citeseer/1",
    "commit_within": 900000,
    "hadoop_conf": "/Users/projects/hadoop/hadoop-0.20.2/conf"
    "output_args": null,
    "output_type": "solr",
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

`PUT /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

MapR Data Sources

The MapR data source allows indexing content from a MapR Hadoop distribution. It is very similar to the HDFS data source, but has been designed specifically for the MapR distribution.

MapR Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a file data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.
type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.

name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.
category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description
mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

MapR Type-Specific Attributes

When creating a data source of `type` **mapr**, the value **lucid.mapr** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document.
bounds	string	No	none	Either **tree** to limit the crawl to a strict subtree, or "none" for no limits. For example, if crawling `maprfs://10.0.0.50/docs`, the tree option is the equivalent of `maprfs://10.0.0.50/docs/*` and the crawler would not crawl `maprfs://10.0.0.50/other`. If you require more advanced crawl limiting, you should choose **none** and using the includes or excludes options.

converter	string	No	null	Multi-record Hadoop files, such as SequenceFiles and MapReduceFiles, can be crawled with a sub-crawler to produce valid Solr documents (as many as there are Hadoop records in each fileset). The keys and values retrieved must be converted to SolrInputDocuments in order to be indexed. This can be done in the form of subclasses of `om.lucid.crawl.fs.hdfs.Converter` and specified by setting the value of the converter attribute to a fully qualified class name that extends Converter. If no value is specified, then a DefaultConverter will be used, which creates a single document per key-value pair and invokes the value's `toString()` method and adds this to the "body" field.
crawl_depth	long number	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if left blank.
exclude_paths	list	No	empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
include_paths	list	No	empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. If left blank, all sub-directory paths will be followed (limited by the crawl_depth). This feature can be used to limit a filesystem crawl to specific sub-directories of a base directory path. For example, if the base directory path is `maprfs://host:port/to/Files`, includes could be used to limit the crawl to sub-directories `maprfs://host:port/to/Files/Archive/2010/*` and `maprfs://host:port/to/Files/Archive/2011/*`.
max_docs	integer	No	-1	The maximum number of documents to crawl. The default is -1, which will collect documents found (in conjunction with the other data source attributes which may limit the number of documents).

max_bytes	integer	No	10485760	Defines the maximum size of a document to crawl. Optional, default is 10,485,760 which is 10Mb per document.
max_threads	integer	No	1	Defines the maximum number of threads the crawler should use. Optional, default is 1, which is single-threaded. The FTP data source type does not support a max_threads value greater than 1.
password	string	Yes	null	The password used to access the MapR file system.
url	string	Yes	null	A fully-qualified MapR file system URL, including the protocol (`maprfs`), host name and port of the CLDB service, and path of the target resource to crawl, such as `maprfs://hostname:9000/path/to/crawl`.
username	string	Yes	null	The username used to access the MapR file system.
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to false. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

Example MapR data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "none",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "converter": null,
    "crawl_depth": -1,
    "crawler": "lucid.mapr",
    "exclude_paths": [],
    "id": "d09e93b8ddb7438c9dd0c37728e7d0c8",
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 1,
    "name": "MapR File System",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "password": "maprPw",
    "type": "mapr",
    "url": "maprfs://192.0.0.1:7222/crawl/data",
    "username": "maprUser",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None
Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.
Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

MongoDB Data Sources

The MongoDB crawler allows indexing of a MongoDB instance.

When a MongoDB data source runs for the first time, an initial synchronization with MongoDB is performed to get all content.

Once the initial synchronization is completed, updates and new documents are indexed by reading the oplog and processing the entries. At any time a full synchronization can be performed again if necessary.

MongoDB Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a file data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.
type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem

crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.
category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.

output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description
mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.

default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

MongoDB Type-Specific Attributes

When creating a data source of `type` **mongodb**,the value **lucid.mongodb** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description

collections	string	No	null	The MongoDB collections to index, i the format of `databaseName.collect` . Multiple collections be separated by commas. If this is left empty, a collections accessible this crawler will be indexed. If the `userna` and `password` entered for authentication on has access to a speci set of collections and `collections` is left empty, only the collections accessible that username and password will be indexed. Note that the collecti referred to here are different from the LucidWorks Search concept of collections and relate to the structure of the MongoDB instance or

converter_class	string	No	com.lucid. mongodb.converters. DotRepresentation-TwoConverter	Defines the class to u to convert the Mongc documents into a flat Solr document for indexing. Document structure is preservec by transforming the MongoDB nested fielc into "paths" for Solr field names. For example, a nested address in MongoDB would become `address.location.str` and `address.location.cit` The default should be retained unless you have implemented a custom converter cla and added it to the .j file of the crawler.
host	string	Yes	localhost	The hostname of the MongoDB instance.
port	integer	Yes	27017	The port of the MongoDB instance.

perform_initial_sync	boolean	Yes	True	If **true**, the connecto will process all data from the defined MongoDB collection. This should be true fo the initial synchronization. If or updates are needed, change this to **false**. you believe you have gotten out of sync wi the oplog (see the `process_oplog` option more detail), then yo could do an initial syn again and all the previously crawled da will be replaced with new crawl.
process_oplog	boolean	Yes	True	If **true**, will watch th MongoDB oplog for n insert, update, and/o delete operations and will process them accordingly. This sho be true for regular operation, after the intial synchronization Change this to **false** you intend to do a fu synchronization with MongoDB collection again.
username	string	No	null	The username to use the MongoDB instanc requires a username and password. If the username used has access to a limited se of collections, only those collections will indexed

password	string	No	null	The password to use the MongoDB instanc requires a username and password.

⚠ At the current time, the MongoDB data source type does not validate that the MongoDB instance is accessible to the LucidWorks Search crawler during creation, which many other crawlers do. This may lead to errors when attempting to crawl if it is not possible for the LWE-Connectors component to connect to the database with the supplied `host`, `port`, `username`, `password` and `collections` combination.

Example MongoDB data source (without mapping attributes):

```
{
    "category": "Other",
    "collection": "collection1",
    "collections": "",
    "commit_on_finish": true,
    "commit_within": 900000,
    "converter_class": "com.lucid.mongodb.converters.DotRepresentationTwoConverter",
    "crawler": "lucid.mongodb",
    "host": "localhost",
    "id": "30fd65168c0a480f94cc203b926d527d",
    "mapping": {
   ....
    },
    "name": "MongoDB test",
    "output_args": null,
    "output_type": "solr",
    "password": "admin",
    "perform_initial_sync": true,
    "port": 27017,
    "process_oplog": true,
    "type": "mongodb",
    "username": "admin",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

SharePoint Data Sources

LucidWorks supports crawling a SharePoint Repository running on the following platforms:

- Microsoft Office SharePoint Server 2007
- Microsoft Windows SharePoint Services 3.0
- Microsoft SharePoint 2010

LucidWorks Search does not support crawling a SharePoint repository running on Microsoft Portal SharePoint Server 2003 or Microsoft Windows SharePoint Services 2.0.

SharePoint data sources will only discover new or changed content, so if recrawling a SharePoint server and no documents are found, it is likely because none of the content was changed from prior crawls.

Getting Ready to Index SharePoint Content

In order to index SharePoint content, Google Services for SharePoint must be installed on the SharePoint server to be able to fully use the embedded APIs that crawl a SharePoint server. The files are included with LucidWorks when it is installed locally and should be moved to the SharePoint server as described below. Customers of LucidWorks Search hosted on AWS or Azure can download the required files from Google SharePoint Connector downloads; look for the latest version of the "Google SharePoint Connector" in either the Binary or Source distributions (not the "Google Search Box").

1. Login to the SharePoint server whose sites are to be crawled by the SharePoint data source.
2. Go to the ISAPI directory of SharePoint. If you are using the standard default installation, the path to this directory would be `C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI` for SharePoint 2007 and `C:\Program Files\Common Files\Microsoft Shared\web server extensions\14\ISAPI` for SharePoint 2010.
3. Copy the following files from your LucidWorks installation into SharePoint's ISAPI folder specified in previous step:

```
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Bulk Auth/SharePoint
2007/GSBulkAuthorization.asmx
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Bulk Auth/SharePoint
2007/GSBulkAuthorizationdisco.aspx
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Bulk Auth/SharePoint
2007/GSBulkAuthorizationwsdl.aspx

$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Site Discovery/SharePoint
2007/GSSiteDiscovery.asmx
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Site Discovery/SharePoint
2007/GSSiteDiscoverydisco.aspx
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Site Discovery/SharePoint
2007/GSSiteDiscoverywsdl.aspx

$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Acl/GssAcl.asmx
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Acl/GssAcldisco.aspx
$LWE_HOME/app/webapps/lwe-core/ext/sharepoint_service/Acl/GssAclwsdl.aspx
```

SharePoint Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a SharePoint data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

SharePoint Type-Specific Attributes

The SharePoint crawler will index all content in the repository, including public and personal sites, files, discussion boards, calendars, contacts, and images. When creating a data source of `type` **sharepoint**, the value of **lucid.gcm** must be used for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed pro formats, parsing errors, IO failure the index with whatever metadata from the document (if it was only example) and the reason for the e "fetch" field, which would be adde
aliases	map of string to string	No	Null	Allows mapping of source URL pat used to rewrite URLs before index
domain	string	No	Empty	The domain where the user is autl
enable_security_ trimming	boolean	No	False	This enables restriction of search the person submitting the query is
excluded_urls	string	No	Empty	Directories on the server that sho that should be excluded from the regular expression syntax can be excluded_urls as is used for incluc

feed_unpublished_ documents	boolean	No	True	If there are unpublished documen repository, you can still crawl and selecting this option. The documer to access control rules, the same document.
groupname_format_ in_ace	string	No	domain groupname	Defines how to store the access c groups. ACEs can be stored as the with the group domain added as a
included_urls	string	No	Empty	The directories on the server that indexing. If left blank, a default of paths will be followed, even if they original URL entered. To limit craw repeat the URL in this site with a r indicate all pages from the site. Th source uses GNU regular expressi different from the Java regular ex and file system data sources. Mor syntax can be found in the GNU re documentation.
kdcserver	string	No	Empty	Kerberos KDC Hostname.
ldap_auth_type	string	No	Simple	The bind type to use when connec server. The options are "none", "s
ldap_cache_ groups_membership	boolean	No	True	If **true**, LucidWorks Search will st group information on the first look faster access if it is needed again.
ldap_cache_ refresh_interval	integer	No	7200	How often to refresh the group m seconds. The default is 7200 seco
ldap_cache_size	integer	No	1000	The size of the group membership number of items to store.
ldap_read_ groups_type	string	No	TOKEN_GROUPS	The mode to use for reading grou server. There are three possible m • TOKEN_GROUPS: The faste token_group is a computed the list of SIDs. • IN_CHAIN: A matching rule chain of user to group until • RECURSIVE: The slowest m recursive expansion.

ldap_search_base	string	No	Null	The base node for LDAP user and
ldap_server_host_address	string	No	Null	The hostname of the LDAP server.
ldap_server_port_number	integer	No	389	The port of the LDAP server.
ldap_server_use_ssl	boolean	No	False	If the connection to the LDAP serv SSL connection, enable this optior
max_docs	integer	No	-1	The number of documents to craw -1, all documents found by the cra (in conjunction with the other data which may limit the crawl).
my_site_base_url	string	No	Null	Used for MOSS 2007 only. The My to determine the complete MySite http://server.domain/personal/ad would be entered as http://server credentials provided will allow the complete the MySite URL and crav
password	string	Yes	Null	Password for the username above
push_acls	boolean	No	False	This saves the access control list (each document as a field of each (field). If you are interested in trim the identity of the searcher, you s option. Enabling this without enab is a helpful way to troubleshoot pr
sharepoint_url	string	Yes	Null	The fully qualified URL for the Sha
use_sp_search_visibility	boolean	No	True	When true (default) then SharePo options will be respected. Google must be installed to use this featu
username	string	Yes	Null	Username with authorization to cr repository.
username_format_in_ace	string	No	username	Defines how to store the access cc individual users. ACEs can be stor username, or with the user domai

verify_access	boolean	No	True	By default, LucidWorks Search wil data source is accessible at create create a data source without verif this value to **false**. Note, however Search cannot access the data sou to crawl it.

✓ The included_urls and excluded_urls attributes for SharePoint data sources use GNU Regular Expressions.

Example SharePoint data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "aliases": null,
    "caching": false,
    "category": "GCM",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawler": "lucid.gcm",
    "domain": "Corp",
    "enable_security_trimming": false,
    "excluded_urls": null,
    "feed_unpublished_documents": true,
    "groupname_format_in_ace": "domain\\groupname",
    "id": "ab8f8dacdcbc4bd8a608c7a0d9537648",
    "included_urls": null,
    "indexing": true,
    "kdcserver": "",
    "ldap_auth_type": "Simple",
    "ldap_cache_groups_membership": true,
    "ldap_cache_refresh_interval": 7200,
    "ldap_cache_size": 1000,
    "ldap_read_groups_type": "TOKEN_GROUPS",
    "ldap_search_base": "",
    "ldap_server_host_address": "",
    "ldap_server_port_number": 389,
    "ldap_server_use_ssl": false,
    "mapping": {
...
},
    "max_docs": -1,
    "my_site_base_url": null,
    "name": "SharePoint",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "password": "password",
    "push_acls": false,
    "sharepoint_url": "http://test.example.com/",
    "type": "sharepoint",
    "use_sp_search_visibility": true,
    "username": "user1",
    "username_format_in_ace": "username",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

`POST /api/collections/collection/datasources`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Simple Filesystem Data Sources

The simple File data source is essentially the same as the Aperture-based Filesystem data source, with the exception that this data source type can use multiple threads while crawling, and does not go through Aperture-based parsing (and parsing can be skipped entirely if saving the crawl as a batch).

The filesystem must be directly accessible to the LucidWorks server, either because it is on the same server or because it is mounted as a local drive.

File Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a File data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: <ul><li>**file** for a filesystem (remote or local, but must be paired with the correct crawler, as below)</li><li>**web** for HTTP or HTTPS web sites</li><li>**jdbc** for a JDBC database* **solrxml** for files in Solr XML format</li><li>**sharepoint** for a SharePoint repository</li><li>**smb** for a Windows file share (CIFS)</li><li>**hdfs** for a Hadoop filesystem</li><li>**s3** for a native S3 filesystem</li><li>**s3h** for a Hadoop-over-S3 filesystem</li><li>**external** for an externally-managed data source</li><li>**twitter_stream** for a Twitter stream</li><li>**high_volume_hdfs** for high-volume crawling of a Hadoop filesystem</li></ul>
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: <ul><li>**lucid.aperture** for web and file types</li><li>**lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types</li><li>**lucid.gcm** for sharepoint type</li><li>**lucid.jdbc** for jdbc type</li><li>**lucid.solrxml** for solrxml type</li><li>**lucid.external** for external type</li><li>**lucid.twitter.stream** for twitter_stream type</li><li>**lucid.map.reduce.hdfs** for high_volume_hdfs type</li></ul>
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

File Type-Specific Attributes

Note however, that the data source `type` **file** can also be used for a local filesystem. If intending to crawl a remote filesystem, the value **lucid.fs** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document.
bounds	string	No	Tree	Either **tree** to limit the crawl to a strict subtree, or "none" for no limits. For example, if crawling `smb://10.0.0.50/docs`, the tree option is the equivalent of `smb://10.0.0.50/docs*` and the lucid.fs would not crawl `smb://10.0.0.50/other`. If you require more advanced crawl limiting, you should choose **none** and using the includes or excludes options.
crawl_depth	32-bit integer	No	-1	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if left blank.

exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. If left blank, all subdirectory paths will be followed (limited by the crawl_depth). This feature can be used to limit a filesystem crawl to specific subdirectories of a base directory path. For example, if the base directory path is `/Path/to/Files`, includes could be used to limit the crawl to subdirectories `/Path/to/Files/Archive/2010/*` and `/Path/to/Files/Archive/2011/*`.
max_bytes	long	No	10,485,760	Defines the maximum size of a document to crawl. Optional, default is -1, which is 10Mb per document.
max_docs	integer	No	-1	Defines the maximum number of documents to crawl. The default is -1, which will collect all documents found (in conjunction with other data source attributes).
max_threads	integer	No	1	Defines the maximum number of threads the crawler should use. Optional, default is 1, which is single-threaded. The FTP data source type does not support a max_threads value greater than 1.
type	string	Yes	Null	One of supported data source types, **must** be consistent with the root URL's protocol. For a filesystem data source type, use **file**.
url	string	Yes	Null	The path to the file or directory of files, expressed as a URL. For a filesystem, this would be "file:///<path>/".

The include_paths and exclude_paths attributes for all Remote File System data sources use Java Regular Expressions.

Example file system data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "bounds": "none",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.fs",
    "exclude_paths": [],
    "id": "0e06503c31fe43e2a9cf406132fa706c",
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 1,
    "name": "Desktop Files",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "path": "/Users/cassandra4work/Desktop",
    "type": "file",
    "url": "file:///Users/cassandra4work/Desktop",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources:` list or create data sources in a particular collection

`/api/collections/collection/datasources/id:` update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.

id	The data source ID.

Query Parameters

None.

Input content

None
Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.
Output

Output Content

None

Delete a Data Source

⚠ The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

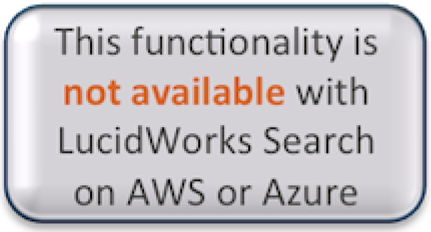
None

Output

Output Content

None

SolrXML Data Sources



This functionality is not available with LucidWorks Search on AWS or Azure

The Solr XML data source can only be used on files are formatted according to Solr's XML structure and cannot be used on XML files formatted for another XML standard. Per the Solr standard, all XML files must include the `<add>` tag in order for the documents to be added to the LucidWorks index. More information on properly formatting a Solr XML file is available at http://wiki.apache.org/solr/UpdateXmlMessages. SolrXML data sources are available in **LucidWorks Search on-premise only**.

SolrXML Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an S3 data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

SolrXML Type-Specific Attributes

When creating a data source of `type` **solrxml**, the value **lucid.solrxml** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
exclude_paths	list of strings	No	Empty	An array of URL patterns to exclude. Used when the data source has been configured to traverse a directory of possible Solr XML files.
generate_unique_key	boolean	No	True	Use **true** to add a unique identifier to each document if one is not specified already for each document in the file. If not specified, the default is **true**.
include_paths	list of strings	No	.* .xml	An array of URL patterns to include. Used when the data source has been configured to traverse a directory of possible Solr XML files.

include_datasource_metadata	boolean	No	True	Use **true** to add data_source and data_source_type fields to each document in addition to fields found in the file. This will allow documents to be included among "Data Source" facets. This has the same effect as the `lucidworks_fields` parameter defined in the mapping, but `include_datasource_metadata` will override `lucidworks_fields` if they are not the same (e.g., if `lucidworks_fields` is false and `include_datasource_metadata` is true, the data_source and data_source_type fields will be added to each document). This attribute also has an impact on how existing documents are handled during subsequent crawls. If **true**, every time a new crawl of this data source is started, all documents created by this data source in previous runs will be deleted at the end of the current crawl job (if it fails in the middle, then a mix of old/new documents will co-exist). In this way, LucidWorks will synchronize the documents for this data source with the source file or directory. If **false**, the deletion step is not performed and all documents found in the source file or directory are treated as new. If the ID of the incoming document matches an existing document, the existing document is replaced with the new document, but otherwise no deletions are performed. This is considered "incremental" crawling, as documents with unique IDs are accumulated across all crawl runs.

max_docs	integer	No	-1	The maximum number of documents to crawl. If the value is -1, then all documents found will be collected (in conjunction with other data source attributes which may limit the crawl).
path	string	Yes	Null	The name of the file to read, or directory containing files to read. Paths should be entered as the complete directory path or they will be interpreted as relative to `$LWE_HOME`. On Unix systems, this means the path entered should start at root / level; on Windows, the drive letter and full path should be used (such as `C:\path`). Various types of relative paths, such as ../ or ~/, are not supported. The API will attempt to validate that the path is accessible to the LucidWorks Search server and will return an error if it is not.
url	string	No	Null	Read-only value that shows the absolute path. In many cases this will be identical to the path as entered, but if the path was a shortcut or symbolic link, the url will show the real path to the files.
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.

The include_paths and exclude_paths attributes for Solr XML data sources use Java Regular Expressions.

Example SolrXML data source (without mapping attributes):

```
{
    "caching": false,
    "category": "SolrXml",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawler": "lucid.solrxml",
    "exclude_paths": [],
    "generate_unique_key": true,
    "id": "90519b810b2749cb9a848e2bacfd403f",
    "include_datasource_metadata": true,
    "include_paths": [
        ".*\\.xml"
    ],
    "indexing": true,
    "mapping": {
    ...
    },
    "max_docs": -1,
    "name": "Solr documents",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "path": "/Users/cassandra4work/Downloads/apache-solr-4.0.0/example/exampledocs",
    "type": "solrxml",
    "url":
"file:/Users/cassandra4work/Downloads/apache-solr-4.0.0/example/exampledocs/",
    "verify_access": true
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Twitter Stream Data Sources

The Twitter Stream data source type uses Twitter's streaming API to index tweets on a continuous basis.

This data source uses the `lucid.twitter.stream` crawler. Unlike other crawlers, which generally have some kind of defined endpoint (even if that endpoint is after hundreds of thousands or millions of documents), this crawler opens a stream and will not stop until Twitter stops. The Stop Crawl button in the Admin UI (or the Data Source Jobs API) will allow you to stop the stream if necessary. This data source is in early stages of development, and does not yet process deletes. Deletes will be shown in the Admin UI and Data Source History statistics, but these are deleted tweets marked as such from the streaming API - the actual tweets may or may not be in the LucidWorks index (and if they were, the data source does not yet process them).

Getting Ready to Index Twitter Streams

In order to successfully configure a Twitter stream, you must first register your application with Twitter and accept their terms of service. The registration process will provide you with the required OAuth tokens you need to access the stream. To get the tokens, follow these steps:

1. Make sure you have a Twitter account, and go to http://dev.twitter.com/ and sign in.
2. Choose "Create an App" link and fill out the required details. The callback field can be skipped. Hit "Create Application" to register your application.
3. The next page will contain the Consumer Key and Consumer Secret, which you will need to configure the data source in LucidWorks Search.
4. At the bottom of the same page, choose "Create My Access Token".
5. The next page will contain the Access Token and Token Secret, which you will also need to configure the data source in LucidWorks Search.

While you need a Twitter account to register an application, you do not use your Twitter credentials to configure this data source. Take the Consumer Key, Consumer Secret, Access Token, and Token Secret information and store it where you can access it while configuring the data source.

Twitter Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating a Twitter data source.

These attributes are used for all data source types (except where specifically noted).

▸

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	**Type**	**Description**

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules
The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing
The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

Twitter Type-Specific Attributes

When creating a data source of `type` **twitter_stream**, the value **lucid.twitter.stream** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
access_token	string	Yes	Null	The access token is provided after regis and requesting an access token (see ab
consumer_key	string	Yes	Null	The consumer key is provided after reg (see above).
consumer_secret	string	Yes	Null	The consumer secret is provided after r (see above). It should be treated as a p registered application.
filter_follow	list	No	Null	A set of specific Twitter user IDs to filte combined with another filter, they act a the stream (i.e., the tweet must match keyword or the location). Note that this or screen name, but a numeric ID assig the ID, you could do an API request like https://api.twitter.com/1/users/show.x }}, replacing "usaa" with the user hand ID is found in the "id" field of the XML c
filter_locations	list	No	Null	A set of bounding boxes (latitude/longit left, etc.) to filter the stream for geogra combined with another filter, they act a the stream (i.e., the tweet must match keyword or the location).

filter_track	list	No	Null	A set of keywords to filter the stream. I another filter, they act as OR statement the tweet must match the user ID or th location).
max_docs	long	No	-1	While testing the feed, it may be desira streams to a specific number of tweets. "-1", which doesn't close the connectior closed.
sleep	integer	Yes	10000	Twitter will occasionally throttle stream can configure the data source to wait th time before trying again. The default is which should be sufficient for most scer
token_secret	string	Yes	Null	The token key is provided after register requesting an access token (see above) as a password for your API access.
url	string	No	stream.twitter.com	It's recommended to use the default.

Example twitter_stream data source

```
{
    "access_token": "Twitter-AccessToken",
    "caching": false,
    "category": "Other",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "consumer_key": "Twitter-ConsumerKey",
    "consumer_secret": "Twitter-ConsumerSecret",
    "crawler": "lucid.twitter.stream",
    "filter_follow": [
        22072684
    ],
    "filter_locations": null,
    "filter_track": null,
    "id": "aec56efadeea48deb0b037d305ccb7fa",
    "indexing": true,
    "mapping": {
    ...
    },
    "max_docs": 100,
    "name": "Twitter",
    "output_args": "http://127.0.0.1:8888/solr/collection1",
    "output_type": "solr",
    "parsing": true,
    "sleep": 10000,
    "token_secret": "Twitter-TokenSecret",
    "type": "twitter_stream",
    "url": "https://stream.twitter.com"
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

`POST /api/collections/collection/datasources`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

`GET /api/collections/collection/datasources/id`

> Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Web Data Sources

The Web data source allows LucidWorks to crawl HTML pages over the internet or intranet. The Web crawler uses Aperture, an open source crawler, which is intended as a small-scale crawler not designed as a large-scale internet crawler (for example, it is single-threaded, which means a long list of websites may take a long time for a crawl to complete).

Aperture extracts fields from the HTML of a crawled page and passes the extracted data to LucidWorks for further processing, including any field mapping that has been defined. As of LucidWorks v2.1, the Web data source is able to extract all fields from the <META> section of an HTML page (prior versions were only able to extract the "author", "description", and "keywords" fields). These custom <META> fields are added to the index with a "meta_" prefix before the tag name (i.e., a custom field of "date" would be inserted in the index as "meta_date"). If you wish to map these fields to another field, use the "meta_*" field name; if you do not map them, they will be added to the index as "attr_meta_*". This is due to a default dynamic rule that adds "attr_" to any field that does not exist in the `schema.xml` for the collection; if you have removed this rule, the fields would be indexed as "meta_*".

If accessing the website is only possible through a proxy server, use the `proxy_*` attributes to configure access.

Web Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an Web data source.

These attributes are used for all data source types (except where specifically noted).

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

Web Type-Specific Attributes

When creating a data source of `type` **web**, the value **lucid.aperture** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Description
add_failed_docs	boolean	No	False	If **true**, documents that failed processing (due to invalid formats, parsing errors, IO failures, etc.) will be added to the index with whatever metadata that could be retrieved from the document (if it was only partially parsed, for example) and the reason for the error in the "parse" or "fetch" field, which would be added to the document. The default for this attribute is **false**.

auth	JSON map	No	Empty	Authentication details, if required. LucidWorks first tries to access the site without authentication. If a "401 Authentication Required" Error is found, the authentication method (Basic, Digest, NTLMv1 or NTLMv2 only) is selected automatically and the closest authentication tuple is selected from the ones configured for the data source. The values for this attribute are provided in a JSON map (multiples are allowed); each map has the following properties: • `host`: The hostname (or host:port) where this authentication should be used; may be null to indicate any host. • `realm`: The HTTP realm where this authentication should be used; may be null to indicate any realm. • `username`: The username; can not be null or empty. • `password`: The password; can not be null or empty.
bounds	string	No	Tree	Either **tree** to limit the crawl to a strict subtree, or **none** for no limits. If you choose **tree**, the crawler will only access pages using the seed URL as the base path. For example, if crawling http://www.cnn.com/US, the subtree option is the equivalent of http://www.cnn.com/US* and would not crawl http://www.cnn.com/WORLD, http://www.cnnmexico.com/, or http://us.cnn.com/. If you require more advanced crawl limiting, you should choose **none** and use the include_paths or exclude_paths options.

crawl_depth	32-bit integer	No	3	The maximum number of crawl cycles (hops) from the starting URL to be crawled. Use '0' to indicate only the seed URL, or any number higher than 0 to go deeper into a site. Use '-1' to indicate unlimited depth, which is also the default if left empty. Unlimited depth will crawl everything linked from the base URL and linked to those links, even if it is 10 or more levels away. If the base URL is a public internet site, unless you constrain the crawl to the subtree or define Allow/Disallow Paths, the crawler may run forever. Note that the lucid.aperture crawler is not designed to create an index of the entire internet, and there may be severe performance or index space problems if you do not constrain the crawl.
exclude_paths	list of strings	No	Empty	Regular expression patterns that URLs may not match. If not empty then a file is excluded if any pattern matches its URL. This attribute accepts Java regular expressions.
fail_unsupported_file_types	boolean	No	False	If **true**, documents that cannot be parsed (either because of unspecified errors or because of an unknown file format) will produce an error in the logs. The default behavior is to not report these documents as failures to the log. If `add_failed_docs` is also checked, the result of these failures will also be added to the index with whatever metadata could be extracted from the content. This is different from `warn_unknown_mime_types` in the sense that this parameter only applies to content that fails parsing and not content that doesn't have a defined mime type.

ignore_robots	boolean	No	False	If **false**, the default, the crawler will respect a robots.txt file found at a site, which is a way for site owners to request no crawling of parts or all of their site. This should only be changed to **true** if the site is known or if the site owner has granted permission to ignore it. The crawler obeys most of the robots.txt standard, with the exception of the Crawl-Delay directive.
include_paths	list of strings	No	Empty	Regular expression patterns to match on full URLs of files. If not empty then at least one pattern must match to include a file. If you leave this field empty, all paths will be followed (except when tree is chosen as a bounds parameter), even if they lead away from the original URL entered. To limit crawling to a specific site, repeat the URL with a regular expression to indicate all pages from the site (such as, enter `http://www\.lucidworks\.com/.*` if you want to crawl all pages under the URL `http://www.lucidworks.com`. This attribute accepts Java regular expressions.

log_extra_detail	boolean	No	False	If **false**, the default, LucidWorks limits the amount of information that is printed to the logs about crawl activities to reduce the size of the log files. However, this information can be helpful if documents aren't being indexed as expected. Change to **true** for more information to be printed to the data/logs/core.<YYYY_MM_DD>.log. The default is for one line to be printed to the log showing the status of the document accessed ("New", "Update", "Delete" or "Unchanged"). When this setting is enabled, two lines will be shown in the logs for every document: one to say the file is being accessed, and a second to show the status of the document. If there is an error in accessing the document, the first line of the extra log detail may be helpful in figuring out the problem.
max_bytes	long	No	10,485,760	Defines the maximum size of any crawled file. The default is -1, which is 10Mb per document.
max_docs	long	No	-1	Defines the maximum number of documents to crawl. The default is -1, which is all found documents, in accordance with other parameters for the data source.
proxy_host	string	No	Empty	The host name of the proxy. LucidWorks supports either open or authenticated proxies. If null or absent then defaults configured in `conf/lwe-core/defaults.yml` are used (see Configuring Default Settings for more information). If no system-wide defaults are defined, then direct access is assumed, and all other proxy-related parameters are ignored.
proxy_port	string	No	-1	The port number of the proxy.
proxy_username	string	No	Empty	optional username credential for the proxy.
proxy_password	string	No	Empty	optional password credential for the proxy.

url	string	No	Null	The URL that serves as the crawl seed. This is expected to be unique for each data source of this type, which means that creating two data sources for the same seed URL is not possible. Note that the lucid.aperture crawler may not always be able to work successfully with HTTP redirects and may fail on an initial attempt to crawl a site that redirects a user to a different address. In most cases, a second attempt to crawl the data source will be successful.
verify_access	boolean	No	True	By default, LucidWorks Search will attempt to verify the data source is accessible at create time. To be able to create a data source without verifying access, change this value to **false**. Note, however, that if LucidWorks Search cannot access the data source, it will not be able to crawl it.
warn_unknown_ mime_types	boolean	No	False	If **true**, documents with no mime type specified in the format produce a warning in the log. If the file cannot be processed as plain text, it will be skipped and a warning message will be printed to the log. The default behavior is to skip these documents and not report warnings in the log. This parameter differs from `fail_unsupported_file_types` in the sense that it only applies to content that does not have a defined mime type.

Example web data source (without mapping attributes):

```
{
    "add_failed_docs": false,
    "auth": [],
    "bounds": "tree",
    "caching": false,
    "category": "Web",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": 3,
    "crawler": "lucid.aperture",
    "exclude_paths": [],
    "fail_unsupported_file_types": false,
    "id": "9b787cd16e354498ae97adefe0817029",
    "ignore_robots": false,
    "include_paths": [
        "http://lucidworks.lucidimagination.com/display/solr/.*"
    ],
    "indexing": true,
    "log_extra_detail": false,
    "mapping": {
    ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "name": "Apache Solr Reference Guide",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "proxy_host": "",
    "proxy_password": "admin",
    "proxy_port": -1,
    "proxy_username": "admin",
    "type": "web",
    "url": "http://lucidworks.lucidimagination.com/display/solr",
    "verify_access": true,
    "warn_unknown_mime_types": false
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

`GET /api/collections/collection/datasources`

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

`POST /api/collections/collection/datasources`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

`GET /api/collections/collection/datasources/id`

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

`DELETE /api/collections/collection/datasources/id`

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Windows Shares Data Sources

The Windows Shares data source allows crawling SMB or CIFS filesystems.

Information about how LucidWorks Search interprets Access Control Lists for a Windows Share, is available at Crawling Windows Shares with Access Control Lists.

SMB (Windows Shares) Data Source Attributes

Common Data Source Attributes

Because all data sources share the same framework, there are a number of shared attributes between each. These should be combined with the type-specific attributes below when creating or updating an S3 data source.

These attributes are used for all data source types (except where specifically noted).

▸

Click here to expand the table of common attributes

General Attributes

Key	Type	Required	Default	Description
id	32-bit integer	No	Auto-assigned	The numeric ID for this data source.

type	string	Yes	Null	The type of this data source. Valid types are: • **file** for a filesystem (remote or local, but must be paired with the correct crawler, as below) • **web** for HTTP or HTTPS web sites • **jdbc** for a JDBC database* **solrxml** for files in Solr XML format • **sharepoint** for a SharePoint repository • **smb** for a Windows file share (CIFS) • **hdfs** for a Hadoop filesystem • **s3** for a native S3 filesystem • **s3h** for a Hadoop-over-S3 filesystem • **external** for an externally-managed data source • **twitter_stream** for a Twitter stream • **high_volume_hdfs** for high-volume crawling of a Hadoop filesystem
crawler	string	Yes	Null	Crawler implementation that handles this type of data source. The crawler must be able to support the specified type. Valid crawlers are: • **lucid.aperture** for web and file types • **lucid.fs** for file, smb, hdfs, s3h, s3, and ftp types • **lucid.gcm** for sharepoint type • **lucid.jdbc** for jdbc type • **lucid.solrxml** for solrxml type • **lucid.external** for external type • **lucid.twitter.stream** for twitter_stream type • **lucid.map.reduce.hdfs** for high_volume_hdfs type
collection	string	Yes	Null	The name of the document collection that documents will be indexed into.
name	string	Yes	Null	A human-readable name for this data source. Names may consist of any combination of letters, digits, spaces and other characters. Names are case-insensitive, and do not need to be unique: several data sources can share the same name.

category	string	No	Null	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.
output_type	string	No	solr	Advanced. Defines the way crawl output is handled. If not defined, this will default to **solr**, meaning the output will be sent to Solr for indexing. Another valid value for this is **NULL** (always all upper-case), in which case the crawl output would be discarded. This could be `com.lucid.crawl.impl.FileUpdateController`, which will send the output to a file. Alternatively, it could be another a fully-qualified class name of a custom implementation of `UpdateController`, which could be created when creating a custom connector.
output_args	string	No	See description	Advanced. Defines where crawler output should be sent. If missing, and the `output_type` is "solr", it will default to the Solr instance as defined in `master.conf` and the collection that uses the data source (so, for example, if LucidWorks has been installed in the default location and creating the data source for collection1, the path would be http://127.0.0.1:8888/solr/collection1). If the `output_type` is `com.lucid.crawl.impl.FileUpdateController`, this must be a URL string for a file path, starting with `file:`. It must point to an existing directory (which must exist) or a filename that will be created during the crawl (which must not exist). If using a custom implementation of `UpdateController`, this attribute can be however you defined its use in that class.

Field Mapping

The output also includes the field mapping for the data source, which is modifiable as part of the regular data source update API. The data source attribute `mapping` contains a JSON map with the following keys and values:

Key	Type	Description

mapping	JSON string-string	A map where keys are case-insensitive names of the original metadata key names, and values are case-sensitive names of fields that make sense in the current schema. These target field names are verified against the current schema and if they are not valid these mappings are removed. Please note that null target names are allowed, which means that source fields with such mappings will be discarded.
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore).
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Optional Commit Rules

The following attributes are optional and relate to when new documents will be added to the index:

Key	Type	Required	Default	Description
commit_within	integer	No	900000	Number of milliseconds that defines the maximum interval between commits while indexing documents. The default is 900,000 milliseconds (15 minutes).
commit_on_finish	boolean	No	True	When **true** (the default), then commit will be invoked at the end of crawl.

Batch Processing

The following attributes control batch processing and are also optional. See also Processing Documents in Batches as some crawlers only support a subset of batch processing options.

Key	Type	Required	Default	Description
parsing	boolean	No	True	When **true** (the default),the crawlers will parse rich formats immediately. When **false**, other processing is skipped and raw input documents are stored in a batch.
indexing	boolean	No	True	When **true** (the default), then parsed documents will be sent immediately for indexing. When **false**, parsed documents will be stored in a batch.
caching	boolean	No	False	When **true**, both raw and parsed documents will always be stored in a batch, in addition to any other requested processing. If **false** (the default), then batch is not created and documents are not preserved unless as a result of setting other options above.

SMB Type-Specific Attributes

When creating a data source of `type` **smb**, the value **lucid.fs** must be supplied for the `crawler` attribute, described in the section on common attributes.

Key	Type	Required	Default	Descripti
ad_cache_groups	boolean	No	False	If **true**, th Directory stored in r don't have search tim rebuilt wh re-crawled schedule t or more a group hier
ad_credentials	string	No	Null	The passw the Active
ad_group_ base_dn	string	No	Null	The Active directory v object resi
ad_group_filter	string	No	`(&(objectclass=group))`	An LDAP fi group obje Directory.
ad_url	string	No	Null	The URL, i portion an `ldap://lda`

ad_user_ base_dn	string	No	Null	The Active directory v object resi
ad_user_filter	string	No	`(&(objectclass=user)(userPrincipalName={0}))`	An LDAP fi object in A
ad_user_ principal_name	string	No	Null	The userna Directory s `login@doma`
add_failed_docs	boolean	No	False	If **true**, do processing formats, p failures, et the index v metadata retrieved f (if it was c for examp for the err "fetch" fiel added to t
bounds	string	No	None	Either **tre** to a strict for no limi crawling `s` , the tree equivalent `smb://10.0` lucid.fs wo `smb://10.0` require mo limiting, y **none** and or exclude
crawl_depth	32-bit integer	No	-1	How many descend. U unlimited the defaul
enable_security_ trimming	boolean	No	False	This enabl search res the person query is al

exclude_paths	list of strings	No	Empty	Regular ex that URLs not empty excluded i matches it
include_paths	list of strings	No	Empty	Regular ex match on not empty pattern mu a file. If le subdirecto followed (l crawl_dep be used to crawl to sp of a base example, i path is `/Pa` includes co the crawl t `/Path/to/F` and `/Path/to/F` .
max_bytes	long	No	10,485,760	Defines th document default is per docum
max_docs	integer	No	-1	Defines th of docume of -1 will c found (in other data which may documents
max_threads	integer	No	1	Defines th of threads use. Optio which is si FTP data s support a greater th

password	string	Yes	Null	Password and ACL R accessing
type	string	True	Null	One of sup types, **mu** with the ro For a Wind source typ
url	string	Yes	Null	For SMB (' filesystem includes th the host a to crawl: `s` `/path/to/d` files in the folder hier this URL w unless you limiting op
username	string	Yes	Null	A usernam ACL READ accessing It's recom special use
verify_access	boolean	No	True	By default will attemp source is a time. To b data sourc access, ch **false**. Not LucidWork access the not be abl
windows_domain	string	No	Null	The option of the sha

The `include_paths` and `exclude_paths` attributes for all Remote File System data sources use Java Regular Expressions.

There are a number of SMB related settings that can be controlled by using Java System Properties. See http://jcifs.samba.org/src/docs/api/overview-summary.html#scp for more information about those settings. Currently the easiest way to specify these System Properties is to edit the `$LWE_HOME/conf/master.conf` file and modify the value for the property `lwecore.jvm.params`. A restart of LucidWorks Search is needed to make the changes visible.

Example SMB data source (without mapping attributes):

```
{
    "ad_cache_groups": true,
    "ad_credentials": null,
    "ad_group_base_dn": null,
    "ad_group_filter": "(&(objectclass=group))",
    "ad_url": null,
    "ad_user_base_dn": null,
    "ad_user_filter": "(&(objectclass=user)(sAMAccountName={0}))",
    "ad_user_principal_name": null,
    "add_failed_docs": false,
    "bounds": "none",
    "caching": false,
    "category": "FileSystem",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.fs",
    "enable_security_trimming": false,
    "exclude_paths": [],
    "id": "c64edc3fac6d43a1bfc58ee465ffb7fd",
    "include_paths": [],
    "indexing": true,
    "mapping": {
    ... }
    "max_bytes": 10485760,
    "max_docs": -1,
    "max_threads": 1,
    "name": "Windows Share",
    "output_args": null,
    "output_type": "solr",
    "parsing": true,
    "password": "password",
    "type": "smb",
    "url": "smb://test.example.com/TestFolder/",
    "username": "user1",
    "verify_access": true,
    "windows_domain": ""
}
```

Summary of API Endpoints

`/api/collections/collection/datasources`: list or create data sources in a particular collection

`/api/collections/collection/datasources/id`: update, remove, or get details for a particular data source

This summary shows the API calls available, but does not provide examples. To see example calls using this API, please review the Data Sources page.

Get a List of Data Sources

```
GET /api/collections/collection/datasources
```

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON map of attributes, depending on data source type (see above).

Create a Data Source

```
POST /api/collections/collection/datasources
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input content

JSON block with all required attributes. The ID field, if defined, will be ignored.

Output

Output Content

JSON representation of new data source.

Get Data Source Details

```
GET /api/collections/collection/datasources/id
```

Note that the only way to find the id of a data source is to either store it on creation, or use the API call referenced above to get a list of data sources.

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None.

Input content

None

Output

Output Content

A JSON map of all data source attributes.

Update a Data Source

```
PUT /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Input content

JSON block with either all attributes or just those that need updating. The attributes `type` (data source type), `crawler` (crawler type), and `id` (data source ID) cannot be updated.

Output

Output Content

None

Delete a Data Source

The Data Source DELETE command will delete documents associated with the data source as of v2.5 (in prior versions it did not). To keep the documents, add `keep_docs=true` to the delete request, after the id. For example:

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/4?keep_docs=true
```

```
DELETE /api/collections/collection/datasources/id
```

Input

Path Parameters

Key	Description
collection	the collection name.
id	The data source ID.

Query Parameters

None

Input content

None

Output

Output Content

None

Field Mapping

Field mapping settings can be managed with the Data Sources API, but there is also a way to manipulate mappings and related settings with a dedicated API for field mapping.

Mapping allows control over how documents are indexed on a per-data source basis. If there are fields in the incoming documents that should be indexed to a specific field in the schema, field mapping provides a way to make that happen.

Other changes can be made during the mapping process. For example, you can define a specific dynamic field rule to be used when there isn't a matching field in the schema. You can define a default field to map incoming content to if there is no other explicit mapping. See the details below for more examples of what can be done with field mapping in data sources.

API Entry Points

`/api/collections/collection/datasources/id/mapping`: list, update or remove mapping settings
`/api/collections/collection/datasources/id/mapping/part`: list or remove a specific part of the mapping settings
`/api/collections/collection/datasources/id/mapping/part/key`: list or remove a specific mapping setting, within a *part*

List Field Mapping Settings

```
GET /api/collections/collection/datasources/id/mapping
```

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.

Query Parameters

None.

Output

Output Content

A JSON List of Maps mapping keys to values.

Key	Type	Description
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.
default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore). For example, if the `dynamic_field` is defined as "attr_" and the field name is "updated_price", the resulting field would be "attr_updated_price".
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents. If defined, these are applied as the last step of the overall field mapping.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

mappings	JSON string-string	The mappings section contains a list of source fields and the target fields they will be mapped to. When the mapping is created or updated, LucidWorks checks the mappings against the `schema.xml` for the collection and verifies that the target fields exist in the schema. During indexing, the field mapping process performs the following steps: 1. The mappings are checked for the existence of the source field name. If it exists, it will be mapped to the target field. 2. If the source field name does not exist in the mappings, the `schema.xml` for the collection is checked. If the source field name exists in the schema, it will be indexed to that field. 3. If a `dynamic_field` has been defined, a dynamic field will be created according to the dynamic field rule. 4. If a `default_field` has been defined, the source field will be mapped to the defined default field. 5. If none of these steps has produced a match, the field will be discarded.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Response Codes

List valid response codes and meaning

Examples

Input

```
curl
http://localhost:8888/api/collections/collection1/datasources/e4228671b8d446afb9d8a23124a3
```

Output

```
{
    "datasource_field": "data_source",
    "default_field": null,
    "dynamic_field": "attr",
    "literals": {},
    "lucidworks_fields": true,
    "mappings": {
        "acl": "acl",
        "author": "author",
        "batch_id": "batch_id",
        "body": "body",
        "content-encoding": "characterSet",
        "content-length": "fileSize",
        "content-type": "mimeType",
        "contentcreated": "dateCreated",
        "contentlastmodified": "lastModified",
        "contributor": "author",
        "crawl_uri": "crawl_uri",
        "created": "dateCreated",
        "creator": "creator",
        "date": null,
        "description": "description",
        "filelastmodified": "lastModified",
        "filename": "fileName",
        "filesize": "fileSize",
        "fullname": "author",
        "fulltext": "body",
        "keyword": "keywords",
        "last-modified": "lastModified",
        "last-printed": null,
        "lastmodified": "lastModified",
        "lastmodifiedby": "author",
        "links": null,
        "messagesubject": "title",
        "mimetype": "mimeType",
        "name": "title",
        "page-count": "pageCount",
        "pagecount": "pageCount",
        "plaintextcontent": "body",
        "plaintextmessagecontent": "body",
        "slide-count": "pageCount",
        "slides": "pageCount",
        "subject": "subject",
        "title": "title",
        "type": null,
        "url": "url"
    },
    "multi_val": {
        "acl": true,
        "author": true,
        "body": false,
        "dateCreated": false,
        "description": false,
        "fileSize": false,
        "mimeType": false,
        "title": false
    },
    "original_content": false,
    "types": {
        "date": "DATE",
        "datecreated": "DATE",
        "filesize": "LONG",
        "lastmodified": "DATE"
    },
    "unique_key": "id",
    "verify_schema": true
}
```

Update Field Mapping Settings

`PUT /api/collections/collection/datasources/id/mapping`

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.

Query Parameters

None.

Input Parameters

Key	Type	Description
datasource_field	string	A prefix for index fields that are needed for LucidWorks faceting and data source management. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default.

default_field	string	The field name to use if source name doesn't match any mapping. If null, then dynamicField will be used, and if that is null too then the original name will be returned.
dynamic_field	string	If not null then source names without specific mappings will be mapped to dynamicField_sourceName, after some cleanup of the source name (non-letter characters are replaced with underscore). For example, if the `dynamic_field` is defined as "attr_" and the field name is "updated_price", the resulting field would be "attr_updated_price".
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.
lucidworks_fields	boolean	If **true**, the default, then the field mapping process will automatically add LucidWorks-specific fields (such as data_source and data_source_type) to the documents. There may be some cases where the data source information is already added to the documents, such as with Solr XML documents, where this setting should be **false**. However, without this information, LucidWorks will not be able to properly identify documents from a specific data source and would not be able to show accurate document counts, display the documents in facets, or delete documents if necessary.

mappings	JSON string-string	The mappings section contains a list of source fields and the target fields they will be mapped to. When the mapping is created or updated, LucidWorks checks the mappings against the `schema.xml` for the collection and verifies that the target fields exist in the schema. During indexing, the field mapping process performs the following steps: 1. The mappings are checked for the existence of the source field name. If it exists, it will be mapped to the target field. 2. If the source field name does not exist in the mappings, the `schema.xml` for the collection is checked. If the source field name exists in the schema, it will be indexed to that field. 3. If a `dynamic_field` has been defined, a dynamic field will be created according to the dynamic field rule. 4. If a `default_field` has been defined, the source field will be mapped to the defined default field. 5. If none of these steps has produced a match, the field will be discarded.

multi_val	JSON string-boolean	A map of target field names that is automatically initialized from the schema based on the target field's multiValued attribute. In general, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. ⚠ Field mapping normalization is a step applied after all target names for field values have been resolved, including substitution with dynamic or default field names. This step checks that values are compatible with the index schema. The following checks are performed: • For the "mimeType" field, : if it is defined as multiValued=false then only the longest (probably most specific) value is retained, and all other values are discarded. • If field type is set to DATE in the field mapping, first the values are checked for validity and invalid values are discarded. If multiValued=false in the target schema, then only the first remaining value will be retained, and all other values are discarded. • If field type is STRING, and multiValued=false in the target schema, then all values are concatenated using a single space character, so that the resulting field has only single concatenated value. • For all other field types, if multiValued=false and multiple values are encountered, only the first value is retained and all other values are discarded.

original_content	boolean	If **true**, adds the ability to store the original raw bytes of any document. By default it is **false**. If this is enabled, a field called "original_content" will be added to each document, and will contain the raw bytes of the original document. The field is subject to normal field mapping rules, which means that if this field is not defined in the `schema.xml` file, it will be added dynamically as `attr_original_content` according to the default rules of field mapping. If the "attr_" dynamic rule has been removed, this field may be deleted during field mapping if it is not defined in `schema.xml` (which it is not by default, so possibly should be added, depending on your configuration). ⚠ The data source types that use the lucid.fs, lucid.aperture, and lucid.gcm crawlers (so, data source types Web, File, SMB, HDFS, S3, S3H, FTP, and SharePoint) are the only ones that support this attribute. It is not possible to store original binary content for the Solr XML, JDBC, External or Twitter data source types.
types	JSON string-string	A map pre-initialized from the current schema. Additional validation can be performed on fields with declared non-string types. Currently supported types are DATE, INT, LONG, DOUBLE, FLOAT and STRING. If not specified fields are assumed to have the type STRING. The map is pre-initialized from the types definition in `schema.xml` in the following ways: • Any class with DateField becomes DATE* Any class that ends with *DoubleField becomes DOUBLE • Any class that ends with *FloatField becomes FLOAT • Any class that ends with *IntField or *ShortField becomes INT • Any class that ends with *LongField becomes LONG • Anything else not listed above becomes STRING

unique_key	string	Defines the document field to use as the unique key in the Solr schema. For example, if the schema uses "id" as the unique key field name, and the `unique_key` attribute is set to "url", then field mapping will map "url" to "id". By default, this will be adjusted to match the `schema.xml` value. However, in cases where no indexing will be performed (i.e., a batch crawl is being performed), the schema.xml is not available for checking so it may be useful to edit this parameter manually to fit the expected schema value. If performing a normal crawl (i.e., the crawler finds the documents, parses them, and passes them along for indexing), this field should be left as the default. With external data sources, this parameter would map a field from the incoming documents to be the unique key for all documents.
verify_schema	boolean	If **true**, the default, then field mapping will be validated against the current schema at the moment the crawl job is started. This may result in dropping some fields or changing their multiplicity so they conform to the current schema. The modified mapping is not propagated back to the data source definition (i.e., it is not saved permanently). In this way, the schema can be modified without having to modify the data source mapping definition; however, it may also be more difficult to learn what the final field mapping was. If this value is **false**, then the field mapping rules are not verified and are applied as is, which may result in exceptions if documents are added that don't match the current schema (e.g., incoming documents have multiple values in a field when the schema expects a single value).

Output

Output Content

None.

Examples

Input

Change the mapping of the incoming field "type" to the schema field "mimeType":

```
curl -X PUT -H 'Content-type: application/json' -d '{"mappings":{"type":"mimeType"}}'
http://localhost:8888/api/collections/collection1/datasources/e4228671b8d446afb9d8a23124a3
```

List a Specific Mapping Part

`GET /api/collections/collection/datasources/id/mapping/part`

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.
part	The section of the mapping definition to list. The parts are any of the attributes listed in the table in the section List Field Mapping Settings, such as `mappings`, `unique_key` and `multi_val`.

Query Parameters

None.

Output

Output Content

A JSON List of keys to values, which will vary depending on the part. See the section List Field Mapping Settings for explanations of the parts of the mapping definitions.

Examples

Input

List the settings for the `multi_val` part:

```
curl http://localhost:8888/api/collections/collection1/datasources/
  e4228671b8d446afb9d8a23124a39233/mapping/multi_val
```

Output

```
{
    "acl": true,
    "author": true,
    "body": false,
    "dateCreated": false,
    "description": false,
    "fileSize": false,
    "mimeType": false,
    "title": false
}
```

List a Key for a Mapping Part

`GET api/collections/collection/datasources/id/mapping/part/key`

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.
part	The section of the mapping definition to list. The parts are any of the attributes listed in the table in the section List Field Mapping Settings, such as `mappings`, `unique_key` and `multi_val`.
key	The specific subset of the part to list, if the part is a map (with multiple sub-values, such as `literals`, `mappings`, `multi_val` and `types`) and not a primitive (with a single value, such as `default_field` or `unique_key`). Primitives can be returned with the syntax to return a part.

Query Parameters

None.

Output

Output Content

The value for the specified key. This may be a string, integer or boolean depending on the value of the specified key.

Examples

Input

Get the field that 'created' will be mapped to:

```
curl http://localhost:8888/api/collections/collection1/datasources/
   e4228671b8d446afb9d8a23124a39233/mapping/mappings/created
```

Output

```
dateCreated
```

Remove a Key for a Mapping Part

`DELETE /api/collections/collection/datasources/id/part/key`

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.
part	The section of the mapping definition to list. The parts are any of the attributes listed in the table in the section List Field Mapping Settings, such as `mappings`, `unique_key` and `multi_val`.
key	The specific subset of the part to list, if the part is a map (with multiple sub-values, such as `literals`, `mappings`, `multi_val` and `types`) and not a primitive (with a single value, such as `default_field` or `unique_key`). Primitives can be removed with the syntax to return a part.

Query Parameters

None.

Output

Output Content

None.

Examples

Input

Remove the mapping for the 'last-modified' field:

```
curl -X DELETE http://localhost:8888/api/collections/collection1/datasources/
   127d4e96a8f649d4a90e962e94c3cf1c/mapping/mappings/last-modified
```

Remove a Mapping Part

`DELETE /api/collections/collection/datasources/id/part`

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.
part	The section of the mapping definition to list. The parts are any of the attributes listed in the table in the section List Field Mapping Settings, such as `mappings`, `unique_key` and `multi_val`.

Query Parameters

None.

Output

Output Content

None.

Examples

Input

Remove the settings in the `types` part of the mapping:

```
curl -X DELETE http://localhost:8888/api/collections/collection1/datasources/
   127d4e96a8f649d4a90e962e94c3cf1c/mapping/types
```

Remove All Mappings

This will reset all of the mapping settings to their default values.

`DELETE /api/collections/collection/datasources/id/mapping`

Input

Path Parameters

Key	Description
collection	The name of the collection.
id	The ID of the data source.

Query Parameters

None.

Output

Output Content

None.

Examples

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/datasources/
   127d4e96a8f649d4a90e962e94c3cf1c/mapping
```

Data Source Schedules

Each data source has a corresponding schedule that determines when that particular data source will be updated. Typically, schedules are recurring, so LucidWorks can index new documents from a data source. However, schedules can also be programmed for a single point in time.

A schedule is a property of a data source. As such, it does not have an individual ID. Instead, the API references the ID of the data source itself. The schedule is created at the same time as the data source creation, so there is no mechanism to create a data source schedule.

⚠ A schedule set with this API may display in the Admin UI as "custom" if the options selected do not match those available via the UI. Some examples of cases where this might happen:

- The period is set to something other than hourly (3600 seconds), daily (86,400 seconds) or weekly (604,800 seconds).
- The start_time is set to a day that's not today.
- The start_time is set to an hour that is not this hour.

If the schedule is shown as "custom" in the UI, it is still possible to edit it via the UI. However the options will be limited to the options available through the UI at the current time.

API Entry Points

`/api/collections/collection/datasources/id/schedule`: Get or update this data source's schedule.

Get a Data Source Schedule

`GET /api/collection/collection/datasources/id/schedule`

Input

Path Parameters

Key	Description
collection	The collection name
id	The data source ID

Query Parameters

None

Output

Output Content

A JSON map of fields to values. Fields are:

Key	Type	Description
start_time	date string	The start date for this schedule, in the format `yyyy-MM-dd'T'HH:mm:ss'+/-'hhmm`. The '+/-' is adjusted for the time zone relative to UTC.
period	64-bit integer	The number of seconds in between repeated invocations of this schedule; set to 0 if this should only occur once.
type	string	Currently always `crawl`.
active	boolean	If **true**, this schedule will be run at the next scheduled time.

Response Codes

200: OK

Examples

Find out when data source 8 will be indexed next.

Input

```
curl 'http://localhost:8888/api/collections/collection1/datasources/8/schedule'
```

Output

The data source is scheduled to run every Monday at midnight:

```
{
    "active": true,
    "period": 604800,
    "start_time": "2012-12-17T00:00:00+0000",
    "type": "crawl"
}
```

Update a Data Source Schedule

`PUT /api/collections/collection/datasources/id/schedule`

Input

Path Parameters

Key	Description
collection	The collection name
id	The data source ID

Query Parameters

None

Input Content

Map of fields that should be updated to their new values.

Key	Type	Required	Default	Description
start_time	date string	Yes	create date/time of the data source	The start date for this schedule, in the format `yyyy-MM-dd'T'HH:mm:ss'+/-'hhmm`. The API can accept a relative time of "now". The '+/-' is for the time zone relative to UTC, and also entered without quotes.

period	64-bit integer	No	0	The number of seconds in between repeated invocations of this schedule; set to 0 if this should only occur once.
type	string	Yes	crawl	Currently always `crawl` (required).
active	boolean	No	false	If **true**, this schedule will be run at the next scheduled time.

Output

Output Content

None

Response Codes

204: No Content

Examples

Input

```
curl -X PUT -H 'Content-type: application/json' -d '
{
  "period": 300,
  "type": "crawl",
  "start_time": "2011-03-18T12:10:32-0700",
  "active": true
}' 'http://localhost:8888/api/collections/collection1/datasources/8/schedule'
```

Output

None.

Data Source Jobs

The Data Sources Jobs API allows direct control over the life cycle of data source crawl jobs.

- API Entry Points
- Get the Status of a Data Source in a Collection
- Start Crawling a Data Source in a Collection
- Stop Crawling a Data Source in a Collection
- Get the Status of All Data Sources in a Collection
- Start Crawling All Data Sources in a Collection
- Stop Crawling All Data Sources in a Collection

API Entry Points

`/api/collections/collection/datasources/id/job`: get the status of, start, or stop crawling a data source for a particular collection
`/api/collections/collection/datasources/all/job`: get the status of, start, or stop crawling all data sources for a particular collection

Get the Status of a Data Source in a Collection

```
GET /api/collections/collection/datasources/id/job
```

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Output

Output Content

Key	Description
batch_job	If **false**, the content crawled will be added to the index.
crawl_started	The date and time the crawl started.
crawl_state	The current state of the job. Entries are FINISHED, STOPPED, or RUNNING.
crawl_stopped	The date and time the crawl stopped.
id	The unique id of the datasource.
job_id	The ID of the job itself.
num_access_denied	The number of documents that could not be accessed because of file permissions or wrong authentication.
num_deleted	The number of documents that were removed from the index.
num_failed	The number of documents that could not be parsed.
num_filter_denied	The number of documents that could not be accessed because of inclusion or exclusion rules.

num_new	The number of documents considered "new".
num_not_found	The number of documents the crawler expected to find (because of a link from a known document, from a symlink, or a redirect) but the remote server responded with HTTP 404 NOT_FOUND or "file missing".
num_robots_denied	The number of documents that could not be crawled because of robots.txt rules.
num_total	The total number of documents found during the last crawl.
num_unchanged	The number of documents that were not changed.
num_updated	The number of documents that were updated.

Response Codes

204: SUCCESS_NO_CONTENT

422: CLIENT_ERROR_UNPROCESSABLE_ENTITY, including a list of errors encountered during starting.

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/datasources/6/job'
```

Output

```
{
    "batch_job": false,
    "crawl_started": "2012-02-06T18:40:12+0000",
    "crawl_state": "FINISHED",
    "crawl_stopped": "2012-02-06T18:42:19+0000",
    "id": 6,
    "job_id": "6",
    "num_access_denied": 0,
    "num_deleted": 0,
    "num_failed": 2,
    "num_filter_denied": 0,
    "num_new": 1099,
    "num_not_found": 0,
    "num_robots_denied": 0,
    "num_total": 1101,
    "num_unchanged": 0,
    "num_updated": 0
}
```

Start Crawling a Data Source in a Collection

`PUT /api/collections/collection/datasources/id/job`

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Output

Output Content

None.

Response Codes

204: SUCCESS_NO_CONTENT

422: CLIENT_ERROR_UNPROCESSABLE_ENTITY, including a list of errors encountered during starting.

Examples

Input

```
curl -X PUT 'http://localhost:8888/api/collections/collection1/datasources/8/job'
```

Output

None.

Stop Crawling a Data Source in a Collection

`DELETE /api/collections/collection/datasources/id/job`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None.

Output

Output Content

None.

Response Codes

204: SUCCESS_NO_CONTENT

422: CLIENT_ERROR_UNPROCESSABLE_ENTITY, including a list of errors encountered during stopping.

Examples

Input

```
curl -X DELETE 'http://localhost:8888/api/collections/collection1/datasources/8/job'
```

Output

None.

Get the Status of All Data Sources in a Collection

`GET /api/collections/collection/datasources/all/job`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

Key	Description
batch_job	If **false**, the content crawled will be added to the index.
crawl_started	The date and time the crawl started.
crawl_state	The current state of the job. Entries are FINISHED, STOPPED, or RUNNING.
crawl_stopped	The date and time the crawl stopped.
id	The unique id of the datasource.
job_id	The ID of the job itself.
num_access_denied	The number of documents that could not be accessed because of file permissions or wrong authentication.
num_deleted	The number of documents that were removed from the index.
num_failed	The number of documents that could not be parsed.
num_filter_denied	The number of documents that could not be accessed because of inclusion or exclusion rules.
num_new	The number of documents considered "new".
num_not_found	The number of documents the crawler expected to find (because of a link from a known document, from a symlink, or a redirect) but the remote server responded with HTTP 404 NOT_FOUND or "file missing".
num_robots_denied	The number of documents that could not be crawled because of robots.txt rules.
num_total	The total number of documents found during the last crawl.
num_unchanged	The number of documents that were not changed.
num_updated	The number of documents that were updated.

Response Codes

204: SUCCESS_NO_CONTENT

422: CLIENT_ERROR_UNPROCESSABLE_ENTITY, including a list of errors encountered during starting.

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/datasources/all/job'
```

Output

```
[
    {
        "batch_job": false,
        "crawl_started": "2012-02-06T18:53:53+0000",
        "crawl_state": "RUNNING",
        "crawl_stopped": null,
        "id": 7,
        "job_id": "7",
        "num_access_denied": 0,
        "num_deleted": 0,
        "num_failed": 0,
        "num_filter_denied": 0,
        "num_new": 7,
        "num_not_found": 0,
        "num_robots_denied": 0,
        "num_total": 7,
        "num_unchanged": 0,
        "num_updated": 0
    },
    {
        "batch_job": false,
        "crawl_started": "2012-02-06T18:40:12+0000",
        "crawl_state": "FINISHED",
        "crawl_stopped": "2012-02-06T18:42:19+0000",
        "id": 6,
        "job_id": "6",
        "num_access_denied": 0,
        "num_deleted": 0,
        "num_failed": 2,
        "num_filter_denied": 0,
        "num_new": 1099,
        "num_not_found": 0,
        "num_robots_denied": 0,
        "num_total": 1101,
        "num_unchanged": 0,
        "num_updated": 0
    }
]
```

Start Crawling All Data Sources in a Collection

`PUT /api/collections/collection/datasources/all/job`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

None.

Response Codes

204: SUCCESS_NO_CONTENT

422: CLIENT_ERROR_UNPROCESSABLE_ENTITY, including a list of errors encountered during starting.

Examples

Input

```
curl -X PUT 'http://localhost:8888/api/collections/collection1/datasources/all/job'
```

Output

None.

Stop Crawling All Data Sources in a Collection

`DELETE /api/collections/collection/datasources/all/job`

Input

Path Parameters

Enter path parameters.

Key	Description

collection	The collection name

Query Parameters

None.

Output

Output Content
None.

Response Codes

204: SUCCESS_NO_CONTENT
422: CLIENT_ERROR_UNPROCESSABLE_ENTITY, including a list of errors encountered during starting.

Examples

Input

```
curl -X DELETE 'http://localhost:8888/api/collections/collection1/datasources/all/job'
```

Output

None.

Data Source Status

The Data Source Status API provides a means to get information about whether a data source is currently being processed. This outputs the same information as the Data Source Jobs API but is available as a way to intermittently check the progress of the job.

- API Entry Points
- Get the Status of a Data Source

API Entry Points

`/api/collections/collection/datasources/id/status`: Get this data source's status

Get the Status of a Data Source

`GET /api/collection/collection/datasources/id/status`

Input

Path Parameters

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None

Output

Output Content

Key	**Description**
batch_job	If **false**, the content crawled will be added to the index.
crawl_started	The date and time the crawl started.
crawl_state	The current state of the job. Entries are FINISHED, STOPPED, or RUNNING.
crawl_stopped	The date and time the crawl stopped.
id	The unique id of the datasource.
job_id	The ID of the job itself.
num_access_denied	The number of documents that could not be accessed because of file permissions or wrong authentication.
num_deleted	The number of documents that were removed from the index.
num_failed	The number of documents that could not be parsed.
num_filter_denied	The number of documents that could not be accessed because of inclusion or exclusion rules.
num_new	The number of documents considered "new".
num_not_found	The number of documents the crawler expected to find (because of a link from a known document, from a symlink, or a redirect) but the remote server responded with HTTP 404 NOT_FOUND or "file missing".
num_robots_denied	The number of documents that could not be crawled because of robots.txt rules.
num_total	The total number of documents found during the last crawl.
num_unchanged	The number of documents that were not changed.
num_updated	The number of documents that were updated.

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/datasources/2/status'
```

Output

While the data source is being processed:

```
{
    "batch_job": false,
    "crawl_started": "2012-02-06T18:40:12+0000",
    "crawl_state": "RUNNING",
    "crawl_stopped": null,
    "id": 6,
    "job_id": "6",
    "num_access_denied": 0,
    "num_deleted": 0,
    "num_failed": 2,
    "num_filter_denied": 0,
    "num_new": 227,
    "num_not_found": 0,
    "num_robots_denied": 0,
    "num_total": 229,
    "num_unchanged": 0,
    "num_updated": 0
}
```

After processing is finished, and the data source is idle:

```
{
     "batch_job": false,
     "crawl_started": "2012-02-06T18:40:12+0000",
     "crawl_state": "FINISHED",
     "crawl_stopped": "2012-02-06T18:42:19+0000",
     "id": 6,
     "job_id": "6",
     "num_access_denied": 0,
     "num_deleted": 0,
     "num_failed": 2,
     "num_filter_denied": 0,
     "num_new": 1099,
     "num_not_found": 0,
     "num_robots_denied": 0,
     "num_total": 1101,
     "num_unchanged": 0,
     "num_updated": 0
}
```

Data Source History

The Data Source History API returns historical statistics for previous data source runs. History only returns information about *prior* crawls for a data source. Use Data Source Jobs or Data Source Status for details on currently running crawls.

- API Entry Points
- Get Data Source History

API Entry Points

`/api/collections/name/datasources/id/history`: Get statistics for the last 50 runs of the given data source.

Get Data Source History

`GET /api/collections/collection/datasources/id/history`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.
id	The data source ID.

Query Parameters

None

Output

Output Content

Key	Type	Description
id	integer	The ID of the datasource.
crawl_started	date string	When the crawl began.
crawl_stopped	date string	When the crawl finished.
crawl_state	string	The current state of the crawl (RUNNING, FINISHED, or STOPPED).
num_unchanged	32-bit integer	The number of documents found that were not modified and did not need to be indexed.
num_deleted	32-bit integer	The number of documents that were removed from the index because they were no longer found in the source.
num_new	32-bit integer	The number of new documents that were found in the source and added to the index.
num_updated	32-bit integer	The number of existing documents that were found in the source and updated in the index because they were modified since the last time they were indexed.
num_failed	32-bit integer	The number of documents from which the crawler failed to extract text.
num_total	32-bit integer	The total number of documents found.
batch_job	boolean	If **false**, documents found will be indexed after crawling.
job_id	integer	The ID of the job.

Examples

Input

```
curl 'http://localhost:8888/api/collections/myCollection/datasources/8/history'
```

Output

```
[
    {
        "id": 2,
        "crawl_started": "2011-03-17T22:16:46+0000",
        "num_unchanged": 0,
        "crawl_state" : "FINISHED",
        "crawl_stopped": "2011-03-17T22:16:51+0000",
        "job_id": "2",
        "num_updated": 0 ,
        "num_new": 6,
        "num_failed": 0,
        "num_deleted": 0,
        "num_total":6,
        "batch_job":false,
        "job_id":3
    },
    {
        "id": 2,
        "crawl_started": "2011-03-18T03: 25:04+0000",
        "num_unchanged": 0,
        "crawl_state": "FINISHED",
        "crawl_stopped": "2011-03-18T 03:25:12+0000",
        "job_id": "2",
        "num_updated": 0,
        "num_new": 6,
        "num_failed": 0,
        "num_deleted": 0,
        "num_total: 6,
        "batch_job":false,
        "job_id":2
    }
]
```

Data Source Crawl Data Delete

The Data Source Crawl Data API can be used to remove the entire crawl history for a data source. Without a crawl history, when a data source is re-crawled, all documents will be treated as "never before seen". This can be useful if field settings have been changed (such as whether the field is stored or not) and a re-crawl of content is required.

- API Entry Points
- Delete Crawl History of a Data Source

API Entry Points

`/api/collections/collection/datasources/id/crawldata`: Delete the crawl history (persistent crawl data) for a data source.

Delete Crawl History of a Data Source

`DELETE /api/collections/collection/datasources/id/crawldata`

Input

Path Parameters

Key	Description
collection	The collection name
id	the data source ID

Query Parameters

None

Output

Output Content

None

Response Codes

204: success no content

Examples

Input

```
curl -X DELETE
http://localhost:8888/api/collections/collection1/datasources/3/crawldata
```

Output

None.

Batch Operations

The Batch Operations API allows you to work with crawled batches of content and configure further batch processing jobs. Batches are created by setting the `indexing` attribute of a data source to **false** with the Data Sources API. For more background information about batch processing, including how to configure a data source for batch crawling, see Processing Documents in Batches.

- API Entry Points
- List All Existing Batches
- Delete All Existing Batches

API Entry Points

`api/collections/collection/batches`: List or delete existing batches.

`api/collections/collection/batches/crawler`: List or delete all batches managed by a given crawler controller.

`api/collections/collection/batches/crawler/job`: List all existing or define a new batch processing job.

`api/collections/collection/batches/crawler/job/batch_id`: Start, get the status of, or stop a running batch job.

List All Existing Batches

```
GET api/collections/collection/batches
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON Array with these parameters:

Key	Type	Description
num_docs	integer	The number of documents in the batch.
batch_id	string	The batch identifier.

parsed	boolean	Indicates whether the documents in the batch have been parsed.
description	string	The descriptive text you have given the data source.
finish_time	integer	The time at which the batch process was finished.
start_time	integer	The time at which the batch process started.
parsed_docs	integer	The number of parsed documents in the batch.
ds_id	string	The data source identifier.
crawler	string	The crawler controller type for the batch.
collection	string	The name of the collection containing the batch.

Response Codes

200: OK

Examples

Get a list of all batches:

Input

```
curl http://localhost:8888/api/collections/collection1/batches
```

Output

```
[
{
 "num_docs":0,
 "batch_id":"00000000-0000-0000-0000-057edb4a3fde",
 "parsed":true,
 "description":"CNN",
 "finish_time":0,
 "start_time":1313524585449,
 "parsed_docs":190,
 "ds_id":"39",
 "collection":"collection1",
 "crawler":"lucid.aperture"
},
{
 "num_docs":0,
 "batch_id":"00000000-0000-0000-0000-05a6aa4c26e8",
 "parsed":false,
 "description":"HDFSTestSite",
 "finish_time":0,
 "start_time":1313524756426,
 "parsed_docs":0,
 "ds_id":"57",
 "collection":"collection1",
 "crawler":"lucid.fs"},
{
 "num_docs":0,
 "batch_id":"00000000-0000-0000-0000-057e60bc7733",
 "parsed":false,
 "description":"XMLDS",
 "finish_time":0,
 "start_time":1313524583393,
 "parsed_docs":0,
 "ds_id":"38",
 "collection":"collection1",
 "crawler":"lucid.solrxml"
}
]
```

Delete All Existing Batches

`DELETE api/collections/collection/batches`

Input

Path Parameters

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

None.

Response Codes

204 No Content

Examples

Delete all existing batches:

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/batches
```

Output

None.

List All Batches For A Specific Crawler Controller

`GET api/collections/collection/batches/crawler`

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.

Query Parameters

None.

Output

Output Content

A JSON Array with these parameters:

Key	Type	Description
num_docs	integer	The number of documents in the batch.
batch_id	string	The batch identifier.
parsed	boolean	Indicates whether the documents in the batch have been parsed.
description	string	The descriptive text you have given the data source.
finish_time	integer	The time at which the batch process was finished.
start_time	integer	The time at which the batch process started.
parsed_docs	integer	The number of parsed documents in the batch.
ds_id	string	The data source identifier.
crawler	string	The crawler controller type for the batch.
collection	string	The name of the collection containing the batch.

Response Codes

200 OK

Examples

List batches for the `lucid.aperture` controller:

Input

```
curl 'http://localhost:8888/api/collections/collection1/batches/lucid.aperture'
```

Output

```
[
{
 "num_docs":0,
 "batch_id":"00000000-0000-0000-0000-012e3da48bde",
 "parsed":true,
 "description":"CNN",
 "finish_time":0,
 "start_time":1313519841161,
 "parsed_docs":120,
 "ds_id":"12",
 "collection":"collection1",
 "crawler":"lucid.aperture"
},
{
 "num_docs":0,
 "batch_id":"00000000-0000-0000-0000-05a4574bf361",
 "parsed":true,
 "description":"FileSystemDS",
 "finish_time":0,
 "start_time":1313524746443,
 "parsed_docs":1068,
 "ds_id":"51",
 "collection":"collection1",
 "crawler":"lucid.aperture"
}
]
```

Delete All Batches For A Specific Crawler Controller

`DELETE api/collections/collection/batches/crawler`

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.

Query Parameters

None.

Output

Output Content

None.

Response Codes

204 No Content

Examples

Delete all batches for the `lucid.aperture` controller:

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/batches/lucid.aperture
```

Output

None.

List All Batch Jobs For A Specific Crawler Controller

`GET api/collections/collection/batches/crawler/job`

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.

Query Parameters

None.

Output

Output Content

A JSON Array with the following parameters:

Key	Type	Description
id	string	The data source identifier
crawl_started	timestamp	The timestamp from the start of the crawling process.
num_total	integer	The number of documents included in the job.

num_unchanged	integer	The number of unchanged documents included in the job.
crawl_state	string	The current state of the crawling process.
crawl_finished	timestamp	The timestamp from the finish of the crawling process.
job_id	string	The job identifier.
batch_job	boolean	Indicates whether the job is a batch job or not.
num_updated	integer	The number of updated documents in the job.
num_new	integer	The number of new documents in the job.
num_failed	integer	The number of documents that the crawler failed to process in the job.
num_deleted	integer	The number of deleted documents in the job.

Response Codes

200 OK

Examples

List batch jobs for the `lucid.aperture` controller:

Input

```
curl http://localhost:8888/api/collections/collection1/batches/lucid.aperture/job
```

Output

```
[{
"id":"72",
"crawl_started":"2011-08-16T20:19:56+0000",
"num_total":2,
"num_unchanged":0,
"crawl_state":"FINISHED",
"crawl_stopped":"2011-08-16T20:19:56+0000",
"job_id":"00000000-0000-0000-0000-06c639b86e43",
"batch_job":true,
"num_updated":0,
"num_new":2,
"num_failed":0,
"num_deleted":0
}]
```

Define A Batch Job For A Specific Crawler Controller

```
PUT api/collections/collection/batches/crawler/job
```

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.

Query Parameters

None.

Input Content

A JSON array with these parameters:

Key	Type	Required	Default	Description
batch_id	string	yes	null	The batch identifier, obtained from the batch listing.
collection	string	no	see description	The name of the original collection containing the batch. If this parameter is not included, LucidWorks uses the current collection.
crawler	string	no	see description	The original crawler controller type for the batch. If this parameter is not included, LucidWorks uses the current crawler controller.

ds_id	string	no	null	Defines a data source to use as a template. If you specify a template data source, it will overwrite the field mappings and batch configuration values (parsing, indexing, and caching) for the original data source, as well as the data source identifier. For example, if you are defining a batch job for data source 5, and you set the `ds_id` value to `4`, the new batch job will use the field mappings and batch configuration from data source 4 regardless of the configuration you have defined for data source 5. It will also display `id:4` when you list your batch jobs. If this parameter is not included, LucidWorks uses the original data source field mappings, batch configuration, and identifier.
new_collection	string	no	null	Override for the target collection name, regardless of the batch or data source collection names.
parse	boolean	no	false	If true, LucidWorks parses the batch again, regardless of whether it has already been parsed. If false, the batch is parsed only when `index==true` and the batch has not been parsed.
index	boolean	no	true	If true, LucidWorks submits the parsed documents for indexing in the target collection. If false, LucidWorks does not index the documents.

Output

Output Content

None.

Response Codes

200 OK

Examples

Define a batch job for the `lucid.aperture` controller:

Input

```
curl -X PUT -H 'Content-type: application/json' -d '
{
  "batch_id":"00000000-0000-0000-1243-4df8a1b27888",
  "collection":"collection1",
  "crawler":"lucid.aperture",
  "ds_id":"4",
  "parse":true,
  "index":true
}' http://localhost:8888/api/collections/collection1/batches/lucid.aperture/job
```

Output

None.

Start a Batch Processing Job

`PUT api/collections/collection/batches/crawler/job/batch_id`

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.
batch_id	The batch identifier.

Query Parameters

None.

Output

Output Content

None.

Response Codes

200 OK

Examples

Start a batch processing job:

Input

```
curl -X PUT -H 'Content-type: application/json'
-d '
{
"batch_id":"00000000-0000-0000-1243-344f2becaaa0",
"ds_id":"4",
"collection":"collection1"
}' http://localhost:8888/api/collections/collection1/batches/lucid.aperture/job
```

Output

None.

Get The Status Of A Running Batch Processing Job

`GET api/collections/collection/batches/crawler/job/batch_id`

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.
batch_id	The batch identifier.

Query Parameters

None.

Output

Output Content

A JSON Array with the following parameters:

Key	Type	Description
id	string	The data source identifier
crawl_started	timestamp	The timestamp from the start of the crawling process.
num_total	integer	The number of documents included in the job.
num_unchanged	integer	The number of unchanged documents included in the job.
crawl_state	string	The current state of the crawling process.

crawl_finished	timestamp	The timestamp from the finish of the crawling process.
job_id	string	The job identifier.
batch_job	boolean	Indicates whether the job is a batch job or not.
num_updated	integer	The number of updated documents in the job.
num_new	integer	The number of new documents in the job.
num_failed	integer	The number of documents that the crawler failed to process in the job.
num_deleted	integer	The number of deleted documents in the job.

Response Codes

200 OK

Examples

Get the status of a running batch processing job:

Input

```
curl http://localhost:8888/api/collections/collection1/batches/
lucid.aperture/job/00000000-0000-0000-0000-06c639b86e43
```

Output

```
[
{"id":"72",
"crawl_started":"2011-08-16T20:19:56+0000",
"num_total":2,
"num_unchanged":0,
"crawl_state":"FINISHED",
"crawl_stopped":"2011-08-16T20:19:56+0000",
"job_id":"00000000-0000-0000-0000-06c639b86e43",
"batch_job":true,
"num_updated":0,
"num_new":2,
"num_failed":0,
"num_deleted":0}
]
```

Running batch jobs also appear in the list of all crawl jobs, but their identifiers correspond to `batch_id` and not to a data source ID.

Stop A Running Batch Processing Job

`DELETE api/collections/collection/batches/crawler/job/batch_id`

Input

Path Parameters

Key	Description
collection	The collection name.
crawler	The name of the crawler controller, such as `lucid.aperture`.
batch_id	The batch identifier.

Query Parameters

None.

Output

Output Content

None.

Response Codes

204 No Content.

Examples

Stop a running batch processing job:

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/batches/
lucid.aperture/job/00000000-0000-0000-0000-06c639b86e43
```

Output

None.

JDBC Drivers

This functionality is **not available** with LucidWorks Search on AWS or Azure

LucidWorks Search allows crawling of databases using a JDBC driver compatible with your RDBMS. However,due to licensing constraints, LucidWorks does not ship with a suite of compatible drivers and they must be added to LucidWorks manually. The JDBC Drivers API allows you to upload drivers to LucidWorks Search. Drivers used with LucidWorks Search must be compliant with the JDBC 3 or JDBC 4 specification.

API Entry Points

`/api/collections/collection/jdbcdrivers`: get a list of JDBC drivers or upload a new one
`/api/collections/collection/jdbcdrivers/filename`: update, delete, or get file contents
`/api/collections/collection/jdbcdrivers/classes`: get a list of JDBC 4.0 compliant drivers

Get a List of JDBC Drivers

```
GET /api/collections/collection/jdbcdrivers
```

Input

Path Parameters

Key	Description
collection	The collection name

Query Parameters

None.

Input Content

None.

Output

Output Content

Key	Type	Description
filename	string	The driver filenames in a list

Return Codes

None.

Examples

Get a list of all the driver jar files installed in the system.

Input

```
curl 'http://localhost:8888/api/collections/collection1/jdbcdrivers'
```

Output

```
[
  "mysql.jar",
  "postgresql.jar"
]
```

Upload a New JDBC driver

`POST /api/collections/collection/jdbcdrivers`

You need to upload the JDBC driver using multipart/form-data file upload.

Input

Path Parameters

Key	Description
collection	The collection name

Query Parameters

None.

Input Content

file=*driver filename*

Output

Output Content

None.

Return Codes

204: (no content = success)
400: Bad Request (if form submit type is not multipart; if there is no "file" element in HTTP request; if file stream is empty)
409: Conflict (same file/driver already exists)

Examples

Upload the mysql.jar file to the system in order to support MySQL.

Input

```
curl -F file=@/path/to/mysql.jar
http://localhost:8888/api/collections/collection1/jdbcdrivers
```

Output

None.

Delete a driver

`DELETE /api/collections/collection/jdbcdrivers/filename`

Input

Path Parameters

Key	Description
collection	The collection name
filename	The driver filename

Query Parameters

None.

Input Content

None.

Output

Output Content

Key	Type	Description

Return Codes

204: (No content = success)

409: Not Found

Examples

Remove MySQL support.

Input

```
curl -X DELETE
'http://localhost:8888/api/collections/collection1/jdbcdrivers/mysql.jar'
```

Output
None.

Get a List of JDBC 4.0 Compliant Drivers

`GET /api/collections/collection/jdbcdrivers/classes`

Input

Path Parameters

Key	Description
collection	The collection name

Query Parameters

None.

Input Content

None.

Output

Output Content

None.

Return Codes

None.

Examples

Get a list of the JDBC classes available to datasources.

Input

```
curl 'http://localhost:8888/api/collections/collection1/jdbcdrivers/classes'
```

Output

```
[
  "com.mysql.jdbc.Driver",
  "oracle.jdbc.OracleDriver"
]
```

FieldTypes

The FieldType API allows creating, updating and deleting field types that are used by LucidWorks Search to define how text found in a field should be processed during indexing. For example, dates should be processed differently from prices as they are generally structured differently in documents. The "date" field would use a field type appropriate for dates, while the "price" field would use a field type appropriate for prices. Many field types are included with LucidWorks and in some cases new ones do not need to be created. However, if the existing list of field types is insufficient for your implementation, this API will allow you to create new field types without manually editing the `schema.xml` configuration file. Alternately, however, the LucidWorks Admin UI includes a Field Types screen to create, update and delete field types.

Field Types Require Advanced Knowledge of Solr's Schema

LucidWorks uses Solr's `schema.xml` to define field types and fields, and all the functionality contained in Solr is available within LucidWorks Search. That said, FieldType configuration is for advanced users. This API is intended for those who prefer not to edit the `schema.xml` file by hand but who would be entirely comfortable doing so if required.

About Field Type Properties

There are only two properties common to all field types: "class" and "name". Other properties will vary according to the class. Some may have a map of analyzers which may include char_filters, tokenizers, and/or token_filters. Property names (for the class, name, analyzers, etc.) are all strings, even if they are numeric or boolean. They follow the naming convention found in `schema.xml`, which is to say that while most of the properties in LucidWorks Search follow a "under_score" naming convention, field type properties will generally be in "camelCase", meaning that the property names are identical to the attribute names as specified in a valid Solr `schema.xml` file.

API Entry Points

`/api/collections/collection/fieldtypes`: get a list of all field types
`/api/collections/collection/fieldtypes/fieldtype`: create, update, delete, or get details for a specific field type

Get a List of All Field Types

```
GET /api/collections/collection/fieldtypes
```

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection where this field type is available

Query Parameters

None

Output

Output Content

A JSON List of Maps mapping FieldType keys to values. There are only two properties that are common to all FieldTypes; the other properties vary by the class of the field type. All of the classes available in Solr are also available in LucidWorks.

Key	Type	Description
class	string	The implementing class for this field type
name	string	The name of the field type

Response Codes

200: OK

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/fieldtypes'
```

Output

The example below has been truncated for space (indicated by "..." at the beginning and end of the example - the full output would be much longer).

```
...
{
 "class":"solr.DateField",
 "name":"pdate",
 "omitNorms":"true",
 "sortMissingLast":"true"
},
{
 "class":"solr.TextField",
 "name":"text_ws",
 "positionIncrementGap":"100",
 "analyzers":{
  "default":{
   "char_filters":[],
   "tokenizer":{
    "class":"solr.WhitespaceTokenizerFactory"
   },
   "token_filters":[]
   }
  }
}
...
```

Get Details for a Specific Field Type

```
GET /api/collections/collection/fieldtypes/fieldtype
```

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection where this field type is available
fieldtype	The name of the field type

Query Parameters

None.

Output

Output Content

A JSON List of Maps mapping Collection keys to values. There are only two properties that are common to all FieldTypes; the other properties vary by the class of the field type.

Key	Type	Description
class	string	The implementing class for this field type
name	string	The name of the field type

Response Codes

List valid response codes and meaning

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/fieldtypes/text_en
```

Output

```
{
   "analyzers" : {
      "index" : {
         "token_filters" : [
            {
                "splitOnCaseChange" : "1",
                "catenateWords" : "1",
                "catenateAll" : "0",
                "generateNumberParts" : "1",
                "class" : "solr.WordDelimiterFilterFactory",
                "catenateNumbers" : "1",
                "generateWordParts" : "1"
             },
             {
                "class" : "solr.LowerCaseFilterFactory"
             },
             {
                "class" : "solr.ASCIIFoldingFilterFactory"
             },
             {
                "class" : "com.lucid.analysis.LucidPluralStemFilterFactory",
                "rules" : "LucidStemRules_en.txt"
             }
          ],
          "char_filters" : [],
          "tokenizer" : {
             "class" : "solr.WhitespaceTokenizerFactory"
          }
       },
       "query" : {
          "token_filters" : [
             {
                "ignoreCase" : "true",
                "synonyms" : "synonyms.txt",
                "expand" : "true",
                "class" : "solr.SynonymFilterFactory"
             },
             {
                "ignoreCase" : "true",
                "class" : "solr.StopFilterFactory",
                "words" : "stopwords.txt"
             },
             {
                "splitOnCaseChange" : "1",
                "catenateWords" : "0",
                "catenateAll" : "0",
                "generateNumberParts" : "1",
                "class" : "solr.WordDelimiterFilterFactory",
                "catenateNumbers" : "0",
                "generateWordParts" : "1"
             },
             {
                "class" : "solr.LowerCaseFilterFactory"
             },
             {
                "class" : "solr.ASCIIFoldingFilterFactory"
             },
             {
                "class" : "com.lucid.analysis.LucidPluralStemFilterFactory",
                "rules" : "LucidStemRules_en.txt"
             }
          ],
          "char_filters" : [],
          "tokenizer" : {
             "class" : "solr.WhitespaceTokenizerFactory"
          }
       }
    },
    "positionIncrementGap" : "100",
    "name" : "text_en",
    "class" : "solr.TextField"
}
```

Create a Field Type

`POST /api/collections/collection/fieldtypes`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection where this field type will be available

Query Parameters

None.

Input Content

See the note about properties at the top of the page.

Output

Output Content

A JSON List of Maps mapping Collection keys to values.

Key	Type	Description
class	string	The implementing class for this field type

name	string	The name of this field type

Other properties will also be listed if set during creation.

Response Codes

422: FieldType structure was not syntactically correct

Examples

Input

Simple Example

```
curl -H 'Content-type: application/json' -d
  '{
    "class":"solr.TextField",
     "name":"newfieldtype"
   }' 'http://localhost:8888/api/collections/collection1/fieldtypes'
```

Example with Analyzers

```
curl -H 'Content-type: application/json' -d
{
   "name" : "test_field",
   "class" : "solr.TextField",
   "analyzers" : {
      "default" : {
         "token_filters" : [],
         "char_filters" : [],
         "tokenizer" : {
            "class" : "solr.WhitespaceTokenizerFactory"
         }
      }
   },
   "positionIncrementGap" : "100"
}' 'http://localhost:8888/api/collections/collection1/fieldtypes'
```

Output

Simple Example

```
{
 "class":"solr.TextField",
 "name":"newfieldtype"
}
```

Example with Analyzers

```
{
 "class":"solr.TextField",
 "name":"test_field",
 "positionIncrementGap":"100",
 "analyzers":{
      "default":{
          "char_filters":[],
          "tokenizer":{
             "class":"solr.WhitespaceTokenizerFactory"
           },
          "token_filters":[]
       }
   }
}
```

Update Details for a Specific Field Type

```
PUT /api/collections/collection/fieldtypes/fieldtype
```

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name
fieldtype	The field type name

Query Parameters

None.

Input Content
See the note about properties at the top of the page.

Properties can be removed from an existing field type by specifying a null value in the PUT request. The `analyzers` attribute is considered an individual attribute, so any changes to any part of the sections within the `analyzers` attribute require sending the entire `analyzers` attribute.

Output

Output Content

None.

Response Codes

200: OK

422: Unprocessable Entity

405: Method Not Allowed

Examples

Input

```
curl -X PUT -H 'Content-type: application/json' -d
  '{"positionIncrementGap":"50"}'
  'http://localhost:8888/api/collections/collection1/fieldtypes/text_en'
```

Delete a Specific Field Type

`DELETE /api/collections/collection/fieldtypes/fieldtype`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection where this field type is available
fieldtype	The fieldtype name

Query Parameters

None.

Output

Output Content

None.

Response Codes

200: OK

Examples

Input

```
curl -X DELETE -H 'Content-type: application/json'
  'http://localhost:8888/api/collections/collection1/fieldtypes/payloads'
```

Fields

The Fields API allows for accessing, modifying, or adding field definitions to a Collection schema.

As of LucidWorks Search v2.1, this API will return only static fields by default, and not fields that have been created because of a dynamic field declaration. To view fields created by a dynamic field rule, use a new parameter for this API (`include_dynamic=true`), which will also allow changing a dynamic field to a static field if used in an update request. To view dynamic field declarations, use the Dynamic Fields API.

Some fields included with LucidWorks Search by default are required for proper functioning of the application. Some, however, are only needed for specific features and could be removed or modified if necessary. For a full description of the LucidWorks Search default fields and their purpose, see the section Customizing the Field Schema.

API Entry Points

`/api/collections/collection/fields`: get all fields and their attributes for a collection or create a new field.

`/api/collections/collection/fields/name`: update, delete, or get details for a particular field.

Get a List of Fields and Attributes for a Collection

`GET /api/collections/collection/fields`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

`include_dynamic=true` will include dynamic fields in the response.

Output

Output Content

Key	Type	Description
name	string	The name of the field. Field names are case sensitive. Currently a field name must consist of only A-Z a-z 0-9 - _
copy_fields	list	A list of field names that this field will be copied to. It is not possible to copy a field to the `id` field because it is by default the unique key for the index.
default_boost	float	How much to boost the field when it is not explicitly included in a user's query. The key `search_by_default` must be true to change this setting from 1.0: the settings value reverts to 1.0 when `search_by_default` is set to false.
default_value	string	A default text value for the field. The text specified with this key will be entered if the field is empty in the document.
dynamic_base	string	If non null, this indicates the name of the dynamic field in the schema that is the basis for this fields existence in the collection. Fields with a value in this attribute will not be returned unless `include_dynamic=true` is added to the request.
editable	boolean	Defines if an external client can edit the field; this cannot be changed.
facet	boolean	Set to **true** if the lucid request handler will facet on this field. This setting enables a field's terms (words for text types, or exact value for "string" type) to be returned to the search client. A field must be indexed to be facetable. This setting can be changed without reindexing, as it is used at query time only.

field_type	string	A valid field type defined in `schema.xml`, which can be created or modified with the FieldType API. The field type setting controls how a field is analyzed. There are many options available, and more can be added by adding a new plugin to the `schema.xml` file. It is crucial to understand the underlying values for a field in order to correctly set its type. For full text fields, such as "title", "body", or "description", a text field type is generally the desired setting so individual words in the text are searchable. There are various text field types, most of which are language-specific. However, when a text field value is to be taken literally as-is (exact match only, or for faceting), the "string" type is likely the right choice. There are also types for numeric data, including double, float, integer, and long (and variants of each suitable for sorting: sdouble, sfloat, sint, and slong). The date field accepts dates in the form "1995-12-31T23:59:59.999Z", with the fractional seconds optional, and trailing "Z" mandatory. If you change a field type, we strongly recommend reindexing.
highlight	boolean	Set to **true** if the `/lucid` request handler will highlight snippets of the stored field value when they are returned to the search client. A field must be stored to be highlighted. This setting can be changed without reindexing, as it is used at query time only.
include_in_results	boolean	Set to **true** if the lucid request handler will include this field in its returned results. A field must be stored to be included in results. This setting can be changed without reindexing, as it is used at query time only.
index_for_autocomplete	boolean	Set to **true** if this field will be used as a source for autocomplete. This allows terms from this field to be used in creation of an auto-complete index that will be created by default at the time of indexing. All fields selected for use in auto-complete are combined into a single "autocomplete" field for use in search suggestions. If you change this setting, we recommend that you recreate the auto-complete index as described in Auto-Complete of User Queries.

index_for_spellcheck	boolean	Set to **true** if this field will be used as a source for spellchecking. This allows terms from this field to be used in the creation of a spell check index. All fields selected for use in spell checking are combined into a single "spell" field for use in search suggestions.
indexed	boolean	Set to **true** if this field will be indexed for full text search. An indexed field is searchable on the words (or exact value) as determined by the field type. Unindexed fields are useful to provide the search client with metadata for display. For example, URL may not be a valuable search term, but it is very valuable information to show users in their results list. For performance reasons, a best practice is to index as few fields as necessary to still give users a satisfactory search experience. If you change this setting, you must reindex all documents.
multi_valued	boolean	Set to **true** if this field will contain multiple values from a document. Enable this if the document could have multiple values for a field, such as multiple categories or authors. We recommend that you reindex all documents after changing this setting.
num_facets	integer	Shows the number of facets that will be displayed if `facet` is true.
omit_tf	boolean	The `omit_tf` attribute sets Solr's `omitTermFreqAndPositions` attribute in the schema. If **true**, term frequency and position information will not be indexed. Enable this if the number of times a term occurs in a document (term frequency) and the proximity of a term to other terms (position) should not be stored. This may be useful for fields that are indexed but not used for searching. This option should not be enabled for text fields (for example, field type text_en) since it would prevent the proper operation of phrase queries and other proximity operators such as NEAR which depend on position information. This attribute works in conjunction with the omit_positions attribute; see the description of that attribute for valid combinations of the attributes.

omit_positions	boolean	The `omit_positions` attribute sets Solr's {{omitPositions attribute in the schema. If **true**, term position information will not be indexed. Enable this if the proximity of a term to other terms should not be stored. This attribute works with the `omit_tf` attribute in that it would be possible to remove information about term frequency while retaining proximity information. There are three possible valid combinations of `omit_tf` and `omit_positions`: • `omit_tf` is true and `omit_positions` is true: This would not store any term frequency or position information for the field • `omit_tf` is false and `omit_positions` is true: This would not store term positions information but would store term frequency • `omit_tf` is false and `omit_positions` is false: This would store term positions and term frequency
query_time_stopword_handling	boolean	Set to **true** if the Lucid query parser will intelligently remove stop words at query time. This will require LucidWorks Search to apply the stop word list to queries that use this specific field. This does not enable stop words across the board, only to queries on this field (which may be most useful for 'body' fields, for example).
search_by_default	boolean	Set to **true** if this field will be copied to the default search field. This requires that all queries search this field when the user has not specifically defined a field in a query.
short_field_boost	string	Valid values are: **none** to not boost terms found in short fields at all, **moderate** to boost moderately with LucidWorks `LucidSimilarityFactory` implementation, or **high** to use the standard Lucene boost implementation. This relevancy boost compensates for text in short documents that have fewer opportunities for text matches and may otherwise rank lower in results than they should. Use 'moderate' for typical text fields such as the abstract or body of an article. Use 'high' for very short fields like title or keywords. Use 'none' for non-text fields. We strongly recommend that you follow changes to the short field boost with a full reindex.

stored	boolean	Set to **true** if the original un-analyzed text will be stored. A field can be stored independently of indexing, and made available in the results sent to to a search client. Re-indexing is not necessary when changing the stored field flag, though fields in documents will remain as they were when they were originally indexed until they are reindexed.
synonym_expansion	boolean	Set to **true** if the Lucid query parser should expand synonyms at query time.
term_vectors	boolean	This attribute is for expert use only with Solr's TermVectorComponent. It may help you achieve better highlighting and MoreLikeThis performance at the expense of a larger index. For more information, see http://wiki.apache.org/solr/FieldOptionsByUseCase.
use_for_deduplication	boolean	Set to **true** if the contents of this field will be used when doing document de-duplication.
use_in_find_similar	boolean	Set to **true** if this field will be used with the default More Like This request handler and be taken into consideration in find-similar/more-like-this computations. The field must be indexed for it to be used for find-similar. This setting can be changed without reindexing, as it is used at query time only.

Return Codes

200: OK

404: Not Found

Examples

Get a list of all fields for the collection "social":

Input

```
curl 'http://localhost:8888/api/collections/social/fields'
```

Output

```
[
 {
  "name": "fileSize",
  "default_boost": 1.0,
  "term_vectors": false,
  "default_value": null,
  "index_for_autocomplete": false,
  "use_for_deduplication": false,
  "highlight": false,
  "multi_valued": true,
  "stored": true,
  "indexed": true,
  "search_by_default": false,
  "facet": false,
  "editable": true,
  "index_for_spellcheck": false,
  "synonym_expansion": false,
  "short_field_boost": "high",
  "include_in_results": false,
  "use_in_find_similar": false,
  "query_time_stopword_handling": false,
  "field_type": "text_en",
  "omit_tf": true,
  "copy_fields":[],
  "dynamic_base":null
 },
 {
  "name": "email",
  "default_boost": 1.0,
  "term_vectors": false,
  "default_value": null,
  "index_for_autocomplete": false,
  "use_for_deduplication": false,
  "highlight": false,
  "multi_valued": true,
  "stored": true,
  "indexed": true,
  "search_by_default": false,
  "facet": false,
  "editable": true,
  "index_for_spellcheck": false,
  "synonym_expansion": false,
  "short_field_boost": "high",
  "include_in_results": false,
  "use_in_find_similar": false,
  "query_time_stopword_handling": false,
  "field_type": "text_en",
  "omit_tf": true,
  "copy_fields":[],
  "dynamic_base":null
 }
]
```

Include dynamic fields in the response:

Input

```
curl http://localhost:8888/api/collections/social/fields?include_dynamic=true
```

Create a New Field

`POST /api/collections/collection/fields`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Input Content

JSON block with one or more field attribute key/value pairs.

When creating a new field, the default values for some attributes depends on the properties of the `field_type` specified at the time the field is created. These exceptions are noted below.

Key	Type	Required	Default	Description
name	string	Yes	No default	The name of the field. Field nar sensitive. Currently a field nam only: A-Z a-z 0-9 - _
copy_fields	list <string>	No	null	A list of field names that this fie to. It is not possible to copy a f because it is by default the unic index.
default_boost	float	No	1.0	How much to boost the field wh explicitly included in a user's qu `search_by_default` must be true setting from 1.0: the settings v when `search_by_default` is set t

default_value	string	No	null	A default text value for the field with this key will be entered if t the document.
facet	boolean	No	false	Set to **true** if the `/lucid` reques on this field. Enables a field's te text types, or exact value for "s returned to the search client. A indexed to be facetable. This se changed without reindexing, as time only.
field_type	string	Yes	No default	A valid field type defined in `sch` type setting controls how a field are many options available, and added by adding a new plugin t file. It is crucial to understand t values for a field in order to cor For full text fields, such as "title "description", a text field type i desired setting so individual wo searchable. There are various t most of which are language-spe when a text field value is to be (exact match only, or for faceti type is likely the right choice. There are also types for numeri double, float, integer, and long each suitable for sorting: sdoub slong). The date field accepts d "1995-12-31T23:59:59.999Z", seconds optional, and trailing " If you change a field type, we s reindexing.
highlight	boolean	No	false	Set to **true** if the lucid request highlight snippets of the stored they are returned to the search be stored to be highlighted. Thi changed without reindexing, as time only.

include_in_results	boolean	No	false	Set to **true** if the lucid request include this field in its returned must be stored to be included i setting can be changed without used at query time only.
index_for_autocomplete	boolean	No	false	Set to **true** if this field should b for autocomplete. This allows te to be used in creation of an aut that will be created by default a indexing. All fields selected for auto-complete are combined in "autocomplete" field for use in s If you change this setting, we r you recreate the auto-complete in Auto-Complete of User Queri
index_for_spellcheck	boolean	No	false	Set to **true** if this field should b for spellchecking. This allows te to be used in the creation of a s All fields selected for use in spe combined into a single "spell" fi search suggestions.
indexed	boolean	No	Inherited from the `field_type`	Set to **true** if this field should b text search. An indexed field is words (or exact value) as deter type. Unindexed fields are usef search client with metadata for example, URL may not be a val but it is very valuable informati their results list. For performan practice is to index as few fields still give users a satisfactory se you change this setting, you mu documents.
multi_valued	boolean	No	Inherited from the `field_type`	Set to **true** if this field will cont from a document. Enable this if could have multiple values for a multiple categories or authors. that you reindex all documents setting.
num_facets	integer	No	5	Set to the number of facets tha displayed if the `facet` attribute

omit_tf	boolean	No	Inherited from the `field_type`	The `omit_tf` attribute sets Solr's `omitTermFreqAndPositions` attrib If **true**, term frequency and po: will not be indexed. Enable this times a term occurs in a docum frequency) and the proximity o terms (position) should not be : useful for fields that are indexe searching. This option should n text fields (for example, field ty would prevent the proper opera queries and other proximity ope NEAR which depend on position attribute works in conjunction v omit_positions attribute; see th that attribute for valid combina attributes.
omit_positions	boolean	No	Inherited from the `field_type`	The `omit_positions` attribute se `omitPositions` attribute in the s term position information will n Enable this if the proximity of a terms should not be stored. Thi with the `omit_tf` attribute in tha possible to remove information frequency while retaining proxi There are three possible valid c `omit_tf` and `omit_positions`: • `omit_tf` is true and `omit_` This would not store any position information for tl • `omit_tf` is false and `omit_` This would not store term information but would sto • `omit_tf` is false and `omit_` This would store term pos frequency

query_time_stopword_handling	boolean	No	false	Set to **true** if the Lucid query p intelligently remove stop words will require LucidWorks to apply to queries that use this specific enable stop words across the b queries on this field (which may 'body' fields, for example).
search_by_default	boolean	No	false	Set to **true** if this field should b default search field. This requir search this field when the user defined a field in a query.
short_field_boost	string	No	high	Valid values are: **none** to omit **moderate** to boost moderately Sweet Spot Similarity implemen the standard Lucene boost impl relevancy boost compensates f documents that have fewer opp matches and may otherwise ra than they should. Use 'moderat fields such as the abstract or b Use 'high' for very short fields l keywords. Use 'none' for non-te strongly recommend that you f the short field boost with a full
stored	boolean	No	Inherited from the `field_type`	Set to **true** if the original unan be stored. A field can be stored indexing, and made available in to a search client. Reindexing is when changing the stored field in documents will remain as the were originally indexed until th
synonym_expansion	boolean	No	false	Set to **true** if the lucid query pa synonyms at query time.
term_vectors	boolean	No	Inherited from the `field_type`	This attribute is for expert use TermVectorComponent. It may better highlighting and MoreLik at the expense of a larger inde information, see http://wiki.apache.org/solr/Fiel .

use_for_deduplication	boolean	No	false	Set to **true** if the contents of th used when doing document de-
use_in_find_similar	boolean	No	false	Set to **true** if this field should b default MoreLikeThis request ha into consideration in "find simila The field must be indexed for it "find similar". This setting can b reindexing, as it is used at quer

Field Configuration for Synonyms

Fields must be properly configured for synonyms to work properly. If you expect synonyms to operate on a specific field, the attributes `search_by_default` and `synonym_expansion` must be enabled or you may experience situations where search results do not include all documents which contain the synonym terms. To achieve the broadest application of synonym matching, these settings are particularly important for the "text_all" field, which is configured this way by default. Unless you give a specific field in your query, LucidWorks Search will query for synonym terms only in those fields that are enabled for default search and enabled for synonym expansion.

Output

Output Content

JSON representation of created resource.

Return Codes

201: Created

422: Unprocessable Entity

This error may have several different conditions:

- Name must be specified
- You must specify a field_type for the field
- An explicit field already exists with the field name

Examples

Input

```
curl -H 'Content-type: application/json' -d '
{
  "name": "my_new_field",
  "default_value": "lucid rocks",
  "multi_valued": true,
  "stored": true,
  "indexed": true,
  "facet": true,
  "index_for_spellcheck": true,
  "synonym_expansion": true,
  "field_type": "text_en",
  "copy_fields": [
      "text_medium",
      "text_all"
  ]
}' 'http://localhost:8888/api/collections/collection1/fields'
```

Output

```
{
    "default_boost": 1.0,
    "field_type": "text_en",
    "facet": true,
    "indexed": true,
    "short_field_boost": "high",
    "term_vectors": false,
    "include_in_results": false,
    "stored": true,
    "omit_tf": false,
    "highlight": false,
    "editable": true,
    "search_by_default": false,
    "multi_valued": true,
    "default_value": "lucid rocks",
    "use_for_deduplication": false,
    "name": "my_new_field",
    "synonym_expansion": true,
    "index_for_spellcheck": true,
    "index_for_autocomplete": false,
    "query_time_stopword_handling": false,
    "copy_fields": [
        "text_medium",
        "text_all",
        "spell"
    ],
    "use_in_find_similar": false
}
```

Get Attributes for a Field

`GET /api/collections/collection/fields/name`

Input

Path Parameters

Key	Description
collection	The collection name.
name	The field name.

Query Parameters

`include_dynamic=true` will return attributes for a dynamic field.

Input Content

Output

Output Content

A JSON map of keys to values. For a list of keys, see GET: Output Content.

Return Codes

204: No Content

404: Not Found

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/fields/my_new_field'
```

Output

```
{
    "default_boost": 1.0,
    "field_type": "text_en",
    "facet": true,
    "indexed": true,
    "short_field_boost": "high",
    "term_vectors": false,
    "include_in_results": false,
    "stored": true,
    "omit_tf": false,
    "highlight": false,
    "editable": true,
    "search_by_default": false,
    "multi_valued": true,
    "default_value": "lucid rocks",
    "use_for_deduplication": false,
    "name": "my_new_field",
    "synonym_expansion": true,
    "index_for_spellcheck": true,
    "index_for_autocomplete": false,
    "query_time_stopword_handling": false,
    "copy_fields": [
        "text_medium",
        "text_all",
        "spell"
    ],
    "use_in_find_similar": false
}
```

Get details for a dynamic field:

Input

```
curl
'http://localhost:8888/api/collections/collection1/fields/my_new_field?include_dynamic=tr
```

Update a Field

`PUT /api/collections/collection/fields/name`

Input

Path Parameters

Key	Description
collection	The collection name.

name	The field name.

Query Parameters

`include_dynamic=true` is required to update a dynamic field, and will convert it to a static field.

Input Content

JSON block with one or more key to value mappings.

Any keys you don't edit will keep their existing values, but updating a specific key will overwrite all prior settings for that key. For example, if updating the `copy_field` attribute, any fields previously named as copy field destinations will be overwritten by the new set of copy fields defined.

For a list of keys, see POST: Input Content

Output

Output Content

None

Return Codes

204: No Content

404: Not Found

Examples

Edit the "my_new_field" field so that it's no longer multi-valued. Also change the default value to "lucid really rocks" and remove it from consideration for spellcheck:

Input

```
curl -X PUT -H 'Content-type: application/json' -d '
{
  "default_value": "lucid really rocks",
  "multi_valued": false,
  "index_for_spellcheck": false
}' 'http://localhost:8888/api/collections/collection1/fields/my_new_field'
```

Output

None.

Delete a Field

`DELETE /api/collections/collection/fields/name`

Note that deleting a field only removes it as an option for new documents; existing documents will retain this field, even after it's been deleted. Also, listing all fields in the collection will still show the field after it's been deleted.

Input

Path Parameters

Key	Description
collection	The collection name.
name	The field name.

Output

Output Content

None.

Return Codes

204: No Content

404: Not Found

Examples

Delete the "my_new_field" field.

Input

```
curl -X DELETE 'http://localhost:8888/api/collections/collection1/fields/my_new_field'
```

Output

None.

Related Topics

- Customizing the Field Schema
- Reindexing Content
- FieldTypes

Dynamic Fields

Dynamic fields are those which are not explicitly defined, but are created during indexing based on some criteria, such as a prefix or suffix on the field name. For example, a data set may have a number of fields which end in "_b". Using dynamicField functionality in Solr, it's possible to add the content of those fields to the index without having to specify each one of them in the `schema.xml` file. The Dynamic Fields API allows for accessing, modifying, or adding dynamic field definitions to a Collection schema.

Dynamic fields cannot be used for faceting, highlighting, de-duplication, or MoreLikeThis, unlike explicit fields. However, a dynamic field can be converted to a explicit field with the Fields API, at which point those attributes can be enabled and the field used for those features.

By default, LucidWorks Search includes a dynamic field declaration of `*`. This rule allows the LucidWorks Search crawlers to add fields as needed while processing crawled documents. If not using the crawlers, or are sure of how documents will be parsed by the crawlers, this rule could be removed. See also Customizing the Field Schema for more information on default dynamic rules.

API Entry Points

`/api/collections/collection/dynamicfields`: get all dynamic fields and their attributes for a collection or create a new dynamic field.

`/api/collections/collection/dynamicfields/name`: update, delete, or get details for a particular dynamic field.

Get a List of Dynamic Fields and Attributes for a Collection

```
GET /api/collections/collection/dynamicfields
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

Key	Type	Description
name	string	The name of the dynamic field. Dynamic field names are case sensitive. Currently a field name must consist of only A-Z, a-z, 0-9, - or _, and either begin or end (but not both) with *. Examples of legal names include `attr_*`, `*_t` or `*`.
copy_fields	list <string>	A list of field names that this field will be copied to.
field_type	string	A valid field type defined in `schema.xml`, which can be created or modified with the FieldType API. The field type setting controls how a field is analyzed. There are many options available, and more can be added by adding a new plugin to the `schema.xml` file. It is crucial to understand the underlying values for a field in order to correctly set its type. For full text fields, a text field type is generally the desired setting so individual words in the text are searchable. There are various text field types, most of which are language-specific. However, when a text field value is to be taken literally as-is (exact match only, or for faceting), the "string" type is likely the right choice. There are also types for numeric data, including double, float, integer, and long (and variants of each suitable for sorting: sdouble, sfloat, sint, and slong). The date field accepts dates in the form "1995-12-31T23:59:59.999Z", with the fractional seconds optional, and trailing "Z" mandatory. If you change a field type, we strongly recommend reindexing.
index_for_autocomplete	boolean	Set to **true** if these fields will be used as a source for autocomplete. This allows terms from these fields to be used in creation of an auto-complete index that will be created by default at the time of indexing. All fields selected for use in auto-complete are combined into a single "autocomplete" field for use in search suggestions. If you change this setting, we recommend that you recreate the auto-complete index as described in Auto-Complete of User Queries.

index_for_spellcheck	boolean	Set to **true** if these fields will be used as a source for spellchecking. This allows terms from these fields to be used in the creation of a spell check index. All fields selected for use in spell checking are combined into a single "spell" field for use in search suggestions.
indexed	boolean	Set to **true** if these fields will be indexed for full text search. An indexed field is searchable on the words (or exact value) as determined by the field type. Unindexed fields are useful to provide the search client with metadata for display. For example, URL may not be a valuable search term, but it is very valuable information to show users in their results list. For performance reasons, a best practice is to index as few fields as necessary to still give users a satisfactory search experience. If you change this setting, you must reindex all documents.
multi_valued	boolean	Set to **true** if these fields will be a 'multi_valued' field. Enable this if the document could have multiple values for a field, such as multiple categories or authors. We recommend that you reindex all documents after changing this setting.
omit_tf	boolean	The `omit_tf` attribute sets Solr's `omitTermFreqAndPositions` attribute in the schema. If **true**, term frequency and position information will not be indexed. Set to true this if the number of times a term occurs in a document (term frequency) and the proximity of a term to other terms (position) should NOT be stored. This may be useful for fields that are indexed but not used for searching. This option should not be enabled for text fields (for example, field type `text_en`) since it would prevent the proper operation of phrase queries and other proximity operators such as NEAR which depend on position information. This attribute works in conjunction with the `omit_positions` attribute; see the description of that attribute for valid combinations of the attributes.

omit_positions	boolean	The `omit_positions` attribute sets Solr's `omitPositions` attribute in the schema. If **true**, term position information will not be indexed. Set to true this if the proximity of a term to other terms should NOT be stored. This attribute works with the `omit_tf` attribute in that it would be possible to remove information about term frequency while retaining proximity information. There are three possible valid combinations of `omit_tf` and `omit_positions`: • `omit_tf` is true and `omit_positions` is true: This would not store any term frequency or position information for the field • `omit_tf` is false and `omit_positions` is true: This would not store term positions information but would store term frequency • `omit_tf` is false and `omit_positions` is false: This would store term positions and term frequency
stored	boolean	Set to **true** if the original unanalyzed text will be stored. The fields can be stored independently of indexing, and made available in the results sent to to a search client. Reindexing is not necessary when changing the stored field flag, though fields in documents will remain as they were when they were originally indexed until they are reindexed.
term_vectors	boolean	This attribute is for expert use only with Solr's TermVectorComponent. It may help you achieve better highlighting and MoreLikeThis performance at the expense of a larger index. For more information, see http://wiki.apache.org/solr/FieldOptionsByUseCase.

Return Codes

200: OK

404: Not Found

Examples

Get a list of all dynamic fields for the default LucidWorks collection "collection1":

Input

```
curl http://localhost:8888/api/collections/collection1/dynamicfields
```

Output

```
[
{
 "field_type":"string",
 "multi_valued":true,
 "indexed":true,
 "name":"attr_*",
 "term_vectors":false,
 "index_for_spellcheck":false,
 "index_for_autocomplete":false,
 "omit_tf":true,
 "stored":true,
 "copy_fields":[ ],
 "omit_positions":true
},
{
 "field_type":"date",
 "multi_valued":false,
 "indexed":true,
 "name":"*_dt",
 "term_vectors":false,
 "index_for_spellcheck":false,
 "index_for_autocomplete":false,
 "omit_tf":true,
 "stored":true,
 "copy_fields":[ ],
 "omit_positions":true
}
]
```

Create a New Dynamic Field

```
POST /api/collections/collection/dynamicfields
```

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Input Content

JSON block with one or more field attribute key/value pairs.

When creating a new dynamic field rule, the default values of some attributes are inherited from the `field_type` specified when the rule is created. These exceptions are noted below.

Key	Type	Required	Default	Description
name	string	Yes	No default	The name of the dynamic field. Dynam names are case sensitive. Currently a f must consist of only A-Z, a-z, 0-9, - or *either begin or end (but not both) with* *Examples of legal names include* `attr`,
copy_fields	list <string>	No	null	A list of field names that this field will l to.
field_type	string	Yes	No default	A valid field type defined in `schema.xml` type setting controls how a field is ana are many options available, and more added by adding a new plugin to the `s` file. It is crucial to understand the und values for a field in order to correctly s For full text fields, such as "title", "bod "description", a text field type is gener desired setting so individual words in t searchable. There are various text field most of which are language-specific. H when a text field value is to be taken li (exact match only, or for faceting), the type is likely the right choice. There ar for numeric data, including double, floa and long (and variants of each suitable sdouble, sfloat, sint, and slong). The d accepts dates in the form "1995-12-31T23:59:59.999Z", with th seconds optional, and trailing "Z" mand New field types can be created with th API. If you change a field type, we strongly reindexing.

index_for_autocomplete	boolean	No	false	Set to **true** if these fields should be us source for autocomplete. This allows te these fields to be used in creation of ar auto-complete index that will be create at the time of indexing. All fields select in auto-complete are combined into a s "autocomplete" field for use in search s If you change this setting, we recomm you recreate the auto-complete index a in Auto-Complete of User Queries.
index_for_spellcheck	boolean	No	false	Set to **true** if these fields should be us source for spellchecking. This allows te these fields to be used in the creation c check index. All fields selected for use checking are combined into a single "s for use in search suggestions.
indexed	boolean	No	Inherited from the `field_type`	Set to **true** if these fields should be inc full text search. Indexed fields are sear the words (or exact value) as determir field type. Unindexed fields are useful t the search client with metadata for dis example, URL may not be a valuable s but it is very valuable information to sh their results list. For performance reas practice is to index as few fields as nec still give users a satisfactory search ex you change this setting, you must rein documents.
multi_valued	boolean	No	Inherited from the `field_type`	Set to **true** if these fields should be a 'multi_valued' field. Enable this if the d could have multiple values for a field, s multiple categories or authors. We rec that you reindex all documents after cl setting.

omit_tf	boolean	No	Inherited from the `field_type`	The `omit_tf` attribute sets Solr's `omitTermFreqAndPositions` attribute in t for this collection. If **true**, term freque position information will not be indexe true if the number of times a term occu document (term frequency) and the pr term to other terms (position) should N stored. This may be useful for fields th indexed but not used for searching. Th should not be enabled for text fields (f field type `text_en`) since it would preve proper operation of phrase queries and proximity operators such as NEAR whic on position information.
omit_positions	boolean	No	Inherited from the `field_type`	The `omit_positions` attribute sets Solr' `omitPositions` attribute in the schema collection. If **true**, term position inforn not be indexed. Enable this if the proxi term to other terms should not be stor attribute works with the `omit_tf` attribu would be possible to remove informatio term frequency while retaining proximi information. There are three possible v combinations of `omit_tf` and `omit_posi` • `omit_tf` is true and `omit_positio` This would not store any term fr position information for the field • `omit_tf` is false and `omit_positic` This would not store term positi information but would store term • `omit_tf` is false and `omit_positic` This would store term positions a frequency
stored	boolean	No	true	Set to **true** if the original unanalyzed t be stored. Fields can be stored indeper indexing, and made available in the re to a search client. Reindexing is not ne when changing the stored field flag, th in documents will remain as they were were originally indexed until they are r

term_vectors	boolean	No	Inherited from the `field_type`	This attribute is for expert use only wit TermVectorComponent. It may help yo better highlighting and MoreLikeThis pe at the expense of a larger index. For m information, see http://wiki.apache.org/solr/FieldOptior .

Output

Output Content

JSON representation of the created dynamic field.

Return Codes

201: Created

422: Unprocessable Entity

This error may have several different conditions:

- Name must be specified
- You must specify a field_type for the field
- A field already exists with the same name

Examples

Input

```
curl -H 'Content-type: application/json' -d
   {
 "name":"*_sh",
 "indexed":true,
 "stored":true,
 "field_type":"text_en"
   }' http://localhost:8888/api/collections/collection1/dynamicfields
```

Output

```
{
 "field_type":"text_en",
 "multi_valued":false,
 "indexed":true,
 "name":"*_sh",
 "term_vectors":false,
 "index_for_spellcheck":false,
 "index_for_autocomplete":false,
 "omit_tf":false,
 "stored":true,
 "copy_fields":[ ],
 "omit_positions":false
}
```

Get Attributes for a Dynamic Field

`GET /api/collections/collection/dynamicfields/name`

Input

Path Parameters

Key	Description
collection	The collection name.
name	The dynamic field name.

Query Parameters

None.

Input Content

None.

Output

Output Content

A JSON map of keys to values. For a list of keys, see GET: Output Content.

Return Codes

204: No Content

404: Not Found

Examples

Input

```
curl http://localhost:8888/api/collections/collection1/dynamicfields/attr_*
```

Output

```
{
 "field_type":"string",
 "multi_valued":true,
 "indexed":true,
 "name":"attr_*",
 "term_vectors":false,
 "index_for_spellcheck":false,
 "index_for_autocomplete":false,
 "omit_tf":true,
 "stored":true,
 "copy_fields":[ ],
 "omit_positions":true
}
```

Update a Dynamic Field

`PUT /api/collections/collection/dynamicfields/name`

Input

Path Parameters

Key	Description
collection	The collection name.
name	The dynamic field name.

Query Parameters

None.

Input Content

JSON block with one or more key to value mappings. Any keys you don't edit will keep their existing values. For a list of keys, see POST: Input Content

Output

Output Content

None.

Return Codes

204: No Content

404: Not Found

Examples

Edit the "attr_*" dynamic field so that it is multi-valued:

Input

```
curl -X PUT -H 'Content-type: application/json'
   -d '{"multi_valued":true}'
   http://localhost:8888/api/collections/collection1/dynamicfields/attr_*
```

Output

None. (Check the dynamic field properties to confirm changes.)

Delete a Field

`DELETE /api/collections/collection/dynamicfields/name`

Note that deleting a field only removes it as an option for new documents; existing documents will retain this field, even after it's been deleted. Also, listing all fields in the collection will still show the field after it's been deleted.

Input

Path Parameters

Key	Description
collection	The collection name.
name	The dynamic field name.

Query Parameters

none

Input content

None

Output

Output Content

None

Return Codes

204: No Content

404: Not Found

Examples

Delete the "*_sh" dynamic field.

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/dynamicfields/*_sh
```

Output

None.

Filtering Results

The Filtering API allows you to configure instances of filtering-related search components, such as those used to filter search results by Access Control Lists. A filter can then be used to filter search results according to the Users group membership within Active Directory.

> ⚠ The Filtering API creates a `searchComponent` in the collection's `solrconfig.xml` file.
>
> After the `searchComponent` has been created, it should be added to the `/lucid` request handler with the Search Handler Components API.

API Entry Points

`/api/collections/collection/filtering/`: list all filtering instances.

`/api/collections/collection/filtering/instance-name`: get, update, and delete configuration details about a search component instance.

List Existing Filtering Components

```
GET /api/collections/collection/filtering
```

Input

Path Parameters

None.

Query Contents

None.

Output

Output Contents

A list of JSON hashes of instances configuration:

Key	Description
filterer.class	class name that generates the effective filter queries based on tags. Use `com.lucid.security.WindowsACLQueryFilterer`
filterer.config	hash containing configuration for filterer instance, for `com.lucid.security.WindowsACLQueryFilterer`, the following keys are supported: `fallback_query`: Specifies the query to use when AD provided no sid information for the user, by default this is -*:* `should_clause`: An optional should clause that is used in the top level boolean query that is constructed based on the sid information provided by the Active directory for example if you want to allow all non SMB content to be available to everybody you could use "`*:* -data_source_type:smb`"
provider.class	class name that reads user groups from Windows ActiveDirectory and builds access tags based on those. Use `com.lucid.security.ad.ADACLTagProvider`
provider.config	hash containing configuration for provider instance, for `com.lucid.security.ad.ADACLTagProvider`, the following keys are supported: `java.naming.provider.url`: specifies the ActiveDirectory LDAP URL `java.naming.security.principal`: the user ID for accessing ActiveDirectory `java.naming.security.credentials`: password for the user account accessing ActiveDirectory `userFilter`: specifies the LDAP search filter for the user in ActiveDirectory. By default, the `userFilter` is `(&(objectclass=user)(userPrincipalName={0}))`. `groupFilter`: specifies the LDAP search filter for the group in ActiveDirectory. By default the `groupFilter` is `(&(objectclass=group))`.

Examples

List filterer component confiugurations in the `social` collection.

Input

```
curl 'http://localhost:8888/api/collections/social/filtering'
```

Output

```
{
  "ad": {
    "provider.class":"com.lucid.security.ad.ADACLTagProvider",
    "filterer.class":"com.lucid.security.WindowsACLQueryFilterer",
    "provider.config":{
       "java.naming.provider.url": "ldap://pdc.domain/",
       "java.naming.security.principal": "ad-user@domain",
    },
    "filterer.config":{}
  }
}
```

Get the Configuration of a Single Filtering Component

`GET /api/collections/collection/filtering/instance-name`

Input

Path Parameters

Key	Description
collection	The collection name
instance	name of the search component instance

Output

Output Contents

JSON hash of instance configuration with the following keys:

Key	Description
filterer.class	class name that generates the effective filter queries based on tags. Use `com.lucid.security.WindowsACLQueryFilterer`

filterer.config	hash containing configuration for filterer instance, for `com.lucid.security.WindowsACLQueryFilterer`, the following keys are supported: `fallback_query`: Specifies the query to use when AD provided no sid information for the user, by default this is -*:* `should_clause`: An optional should clause that is used in the top level boolean query that is constructed based on the sid information provided by the Active directory for example if you want to allow all non SMB content to be available to everybody you could use "`*:* -data_source_type:smb`"
provider.class	class name that reads user groups from Windows ActiveDirectory and builds access tags based on those. Use `com.lucid.security.ad.ADACLTagProvider`
provider.config	hash containing configuration for provider instance, for `com.lucid.security.ad.ADACLTagProvider`, the following keys are supported: `java.naming.provider.url`: specifies the ActiveDirectory LDAP URL `java.naming.security.principal`: the user ID for accessing ActiveDirectory `java.naming.security.credentials`: password for the user account accessing ActiveDirectory `userFilter`: specifies the LDAP search filter for the user in ActiveDirectory. By default, the `userFilter` is `(&(objectclass=user)(userPrincipalName={0}))`. `groupFilter`: specifies the LDAP search filter for the group in ActiveDirectory. By default the `groupFilter` is `(&(objectclass=group))`.

Response Codes

200: Success OK

Examples

Get the filterer configuration for the `ad` instance in the `social` collection.

Input

```
curl 'http://localhost:8888/api/collections/social/filtering/ad'
```

Output

```
{
 "filterer.class": "com.lucid.security.WindowsACLQueryFilterer",
 "provider.class": "com.lucid.security.ad.ADACLTagProvider",
 "provider.config":
  {
   "java.naming.provider.url": "ldap://pdc.domain/",
   "java.naming.security.principal": "ad-user@domain",
  }
}'
```

Create a New Filtering Component

`POST /api/collections/collection/filtering`

Input

Path Parameters

Key	Description
collection	The collection name
instance	name of the search component instance

Input Content

Hash of configuration values to set. See Get Output Contents for supported values.

Output

Response Codes

201: created
422: If there was a problem in creating a configuration

Examples

Set the filterer configuration for the `ad` instance in the `social` collection.

```
curl -v -X POST http://localhost:8888/api/collections/social/filtering/ad
 -H "Accept: application/json"
 -H "Content-Type: application/json"
 -d '{
      "filterer.class": "com.lucid.security.WindowsACLQueryFilterer",
      "provider.class": "com.lucid.security.ad.ADACLTagProvider",
      "provider.config":
        {
         "java.naming.provider.url": "ldap://10.0.0.50/",
         "java.naming.security.principal": "user@dc.domain.example",
         "java.naming.security.credentials": "password"
        }
      }'
```

Update a Filtering Component Instance

`PUT /api/collections/collection/filtering/instance`

Input

Path Parameters

Key	Description
collection	The collection name
instance	name of the search component instance

Input Content

Hash of configuration values to set. See Get Output Contents for supported values.

Output

Response Codes

204: success no content

Examples

Set the filterer configuration for the `ad` instance in the `social` collection.

```
curl -v -X PUT http://localhost:8888/api/collections/social/filtering/ad
 -H "Accept: application/json"
 -H "Content-Type: application/json"
 -d '{
      "filterer.class": "com.lucid.security.WindowsACLQueryFilterer",
      "provider.class": "com.lucid.security.ad.ADACLTagProvider",
      "provider.config":
        {
         "java.naming.provider.url": "ldap://10.0.0.50/",
         "java.naming.security.principal": "user@dc.domain.example",
         "java.naming.security.credentials": "password"
        }
      }'
```

Delete a Filtering Component

`DELETE /api/collections/collection/filtering/instance`

Input

Path Parameters

Key	Description
collection	The collection name
instance	name of the search component instance

Output

Response Codes

204: success no content
404: when deleting non existing instance

Search Handler Components

The Search Handler Component API allows you to configure the list of active search components for each particular search handler.

One example of when you'd need to use this API is when you create a new filter with the Filtering API to support limiting results to only authorized users as defined by Windows Access Control Lists. Once the filter is created, you must add the `searchComponent` name to the `/lucid` request handler with this API.

An understanding of Solr's search component functionality is helpful when using this API.

- API Entry Point

- List Search Components for a Search Handler
- Update Search Components for Search Handler

API Entry Point

`/api/collections/collection/components/list-name?handlerName=/handlerName`: List or update search components for a search handler.

List Search Components for a Search Handler

`GET /api/collections/collection/components/list-name?handlerName=/handlerName`

Input

Path Parameters

Key	Description
collection	The collection name.
list-name	The Solr search component list name: **all**, **first**, or **last**. The **all** list, if present, contains all the search components for your search handler. The **first** list, if present, appends the additional search components of your search handler to the beginning of the default list the handler inherits from its parent class. The **last** list, if present, appends the additional search components of your search handler to the end of the default list the handler inherits from its parent class. You cannot create new lists using this API; you can only work with lists that already exist for your search handler.

Query Parameters

Key	Description
handlerName	The name of search handler; use /lucid for the standard Lucid query request handler.

Output

Output Content

JSON array of current search components.

Response Codes

200: Success OK
404: if the list does not exist

Examples

Get the search components for `/lucid handler` in the `social` collection.

Input

```
curl 'http://localhost:8888/api/collections/social/components/all?handlerName=/lucid'
```

Output

```
["rolefiltering","query","mlt","stats","feedback","highlight","facet","spellcheck","debug"
```

Update Search Components for Search Handler

`PUT /api/collections/collection/components/list-name?handlerName=/handlerName`

Input

Input Content

JSON array with all of the components.

Path Parameters

Key	Description
collection	The collection name.
list-name	The Solr search component list name: **all**, **first**, or **last**. The **all** list, if present, contains all the search components for your search handler. The **first** list, if present, appends the additional search components of your search handler to the beginning of the default list the handler inherits from its parent class. The **last** list, if present, appends the additional search components of your search handler to the end of the default list the handler inherits from its parent class. You cannot create new lists using this API; you can only work with lists that already exist for your search handler.

Query Parameters

Key	Description
handlerName	The name of the search handler; use /lucid for the standard Lucid query request handler.

Output

Output Content

None.

Response Codes

200: Success OK

Examples

Set the list of search components for `/lucid` handler in the `social` collection.

Input

```
curl -v -X PUT
http://localhost:8888/api/collections/social/components/all?handlerName=/lucid
 -H "Accept: application/json"
 -H "Content-Type: application/json"
 -d
'["adfiltering","query","mlt","stats","feedback","highlight","facet","spellcheck","debug"]
```

Output
None.

Settings

The Settings API allows for accessing and modifying settings for a given collection. Note that some of the settings listed below cannot be changed by customers with LucidWorks Search hosted on AWS or Azure.

- API Entry Points
- Get All Settings for a Collection
- Get a Particular Setting
- Update Settings

API Entry Points

`/api/collections/collection/settings`: get all settings for a collection or update settings.

`/api/collections/collection/settings/name`: get a particular setting

Get All Settings for a Collection

`GET /api/collections/collection/settings`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

Key	Type	Description
auto_complete	boolean	Is **true** if auto-complete is enabled for use in the LucidWor Search default search interface. Note that this also require setting the auto-complete activity to run at regular interva For more information, see Auto-Complete of User Queries.
boosts	Solr function query	Defines the boost to apply to each query. The default boos the Lucid Query Parser prefers more recent documents.
boost_recent	boolean	Is **true** if the lucid request handler should boost recent documents.
click_enabled	boolean	Is **true** if Click Scoring is enabled. If enabling this feature NearRealTime (NRT) search (the `update_handler_autosoftcommit_*` parameters discussed be please refer to the Click Scoring Relevance Framework sec for more information about how Click Scoring and NRT imp document updates. This feature is available in **LucidWork Search on-premise only**.
click_boost_data	string	The path to Click Scoring boost data. This feature is availa **LucidWorks Search on-premise only**.
click_boost_field	string	The field name prefix used by Click fields. This feature is available in **LucidWorks Search on-premise only**.
click_index_location	string	The path to Click boost index (LucidWorks Search on-prem only).
de_duplication	string	The valid values are: **off**, do not de-duplicate; **overwrite** duplicate documents; **tag** duplicated with a unique signatu Note that de-duplication does not work properly in SolrClo mode.
default_sort	string	Default sort method - valid values are: **relevance**, **date**, **random**.
display_facets	boolean	Is **true** if the LucidWorks Search default search interface s display facets.

display_fields	string	Defines the fields to use for display of results to users. Prir used to add pseudo-fields to documents, but could be usec "real" fields also. This parameter only applies when using t `lucid` handler (query parser).

elevations	JSON map	Defines the documents to be elevated or excluded from res for a specific query. It uses Solr's QueryElevationCompone which is enabled by default in LucidWorks Search. This API an interface to manage the `elevate.xml` file, which stores t elevation definitions that are used for queries. The elevatic file is located in the `conf` directory for each collection (`$LWE_HOME/conf/solr/cores/collection/conf`).
		
		The structure of the `elevate.xml` file is an XML file defining query and the IDs of the documents that are to be elevate excluded. The API uses a JSON map to write to this file wit structure of:
		
		`{"elevations":`
		`   {"query":`
		`      [{"doc":"docID","exclude":false}]`
		`   }`
		`}`
		
		It is also possible to define elevations or exclusions using t built-in Search UI, which includes a "pin" or a "minus" nex every result to allow you to add it to the elevations list as a required document or an excluded document.
		
		The list of documents included or excluded is not synchron with the index, which means that if a document is listed in `elevate.xml` file, and then is removed from the index, it do not get removed from the file.

main_index_ lock_type	string	Defines which Lucene LockFactory to use. When applying changes to an index the IndexWriter requires a lock on the directory. The options are: • **single**: using the SingleInstanceLockFactory. This is suggested for a read-only index or when there is no possibility of another process trying to modify the in • **native**: using the NativeFSLockFactory. This uses th native file locking. It should not be used when there multiple Solr web applications in the same JVM that attempting to share a single index. • **simple**: using the SimpleFSLockFactory. This uses a lock file for locking. More information is available in the Lucene Wiki: http://wiki.apache.org/lucene-java/AvailableLockFac .
main_index_ max_buffered_docs	integer	Allows setting the `maxBufferedDocs` parameter in the `solrconfig.xml` file for the collection, which sets the numb document updates to buffer in memory before they are flu to disk and added to the current index segment. It is gene preferred to use the `main_index_ram_buffer_size_mb`, but if settings are defined, a flush will occur when either limit is reached.
main_index_ max_merge_docs	integer	Allows setting the `maxMergeDocs` parameter in the `solrconfig.xml` file for the collection, which sets the maxin number of documents for a single segment. Once this limit reached, the segment is closed and a new one is created. segment merge, as defined by `main_index_merge_factor` m also occur at this time.
main_index_ merge_factor	integer	Allows setting the `mergeFactor` parameter in the `solrconfi` file for the collection, which defines how many segments t index is allowed to have before they are coalesced into on segment. When the index is updated, the new data is add the most recently opened segment. When that segment is a new segment is created and subsequent updates are pla there (defining when a segment is full is done with the `main_index_max_buffered_docs` and `main_index_ram_buffer_size_mb` settings). When the the `main_index_merge_factor` is reached, the segments are me into a single larger segment. See the section on `mergeFact` the Solr Reference Guide for more information.

main_index_ ram_buffer_size_mb	integer	Allows setting the `ramBufferSizeMb` parameter in the `solrconfig.xml` file for the collection, which sets the amour memory space (in megabytes) document updates can use before they are flushed to the current index segment. This setting is generally preferable to `main_index_max_buffered_` but if both settings are defined, a flush will occur when eitl limit is reached.
main_index_ term_index_interval	integer	Allows setting the `TermIndexInterval` for the index and determines the amount of computation required per query term, regardless of the number of documents. This allows level of control over the time query processing takes. Larg values cause less memory to be used by the IndexReader, slows random-access to terms. Smaller values cause more memory to be used by the IndexReader, but will speed random-access to terms. A large index with user-entered queries may benefit from a larger `main_index_term_index_interval` because query processing dominated by frequency and positional data processing an by term lookup. A system that experiences a great deal of wildcard queries may benefit from a smaller value for this setting.
main_index_ use_compound_file	boolean	Allows you to set the `UseCompoundFile` parameter in the `solrconfig.xml` file for the collection. Setting this to **true** combines the multiple index files on disk to a single file. Tl setting would help avoid hitting an open file limit on those systems which restrict the number of open files allowed pe process. See the section on `UseCompoundFile` in the Solr Reference Guide for more information.
main_index_ write_lock_timeout	integer	Defines the maximum time to wait for a write lock.
query_parser	string	Which query parser the lucid search request handler will u: valid values are: **lucid**, **dismax**, **extended dismax**, **luce**
query_time_stopwords	boolean	Is **true** if stopwords will be removed at query time.
query_time_synonyms	boolean	Is **true** if synonyms should be added to queries. This will c be used if the 'lucid' query parser is selected as the defaul used in the query request.
search_server_list	list:string	A list of Solr core URLs that the lucid request handler will u for distributed search - pass an empty list to disable distrib search.

show_similar	boolean	Is **true** if a "Find Similar" link should be displayed next to user's search results.
spellcheck	boolean	Is **true** if the LucidWorks Search default search interface s suggest spelling corrections.
stopword_list	list:string	A list of stopwords that will be used if 'query_time_stopwo is enabled.
synonym_list	list:string	A list of synonym rules that will be used if 'query_time_synonyms' is enabled.
unknown_type_handling	string	A valid field type from the core's schema to use for unrecognized fields - default is **text_en**.
unsupervised_feedback	boolean	Is **true** if unsupervised feedback is enabled
unsupervised_ feedback_emphasis	string	Defines if unsupervised feedback should emphasize "**relev** " which does an "AND" of the original query which neither includes nor excludes additional documents, or "**recall**" wh does an "OR" of the original query which permits the feedb terms to expand the set of documents matched - default is **relevancy**".
update_handler_ autocommit_max_docs	integer	Allows setting the `maxDocs` parameter for `autocommit` definit in the `solrconfig.xml` file for the collection. This setting de the number of documents to queue before pushing them to index. It works in conjunction with the `update_handler_autocommit_max_time` parameter in that if e limit is reached, the pending updates will be pushed to the index.
update_handler_ autocommit_max_time	integer	Allows setting the `maxTime` parameter for `autocommit` definit in the `solrconfig.xml` file for the collection. This setting de the number of milliseconds to wait before pushing docume to the index. It works in conjunction with the `update_handler_autocommit_max_docs` parameter in that if e limit is reached, the pending updates will be pushed to the index.
update_handler_ autocommit_open_searcher	boolean	Provides the option to not open a searcher on hard commit This may be useful to minimize the size of transaction logs keep track of uncommitted updates. The default is **true**, change this to **false** to not open a searcher.

update_handler_ autosoftcommit_max_docs	integer	Allows setting the `maxDocs` parameter for `autosoftcommit` definitions in the `solrconfig.xml` file for the collection. "Sof commits are used in Solr's Near RealTime search. This sett defines the number of documents to queue before pushing them to the index. It works in conjunction with the {{update_handler_ autosoftcommit_max_time}} parameter in that if either lir reached, the documents will be pushed to the index.
update_handler_ autosoftcommit_max_time	integer	Allows setting the `maxTime` parameter for `autosoftcommit` definitions in the `solrconfig.xml` file for the collection. "Sof commits are used in Solr's Near RealTime search. This sett defines the number of milliseconds to wait before pushing documents to the index. It works in conjunction with the `update_handler_autosoftcommit_max_docs` parameter in that either limit is reached, the documents will be pushed to th index.
update_server_list	complex	A map that contains two keys: 'server_list' and 'self'. 'server_list' is list:string of servers that the lucid update ch will use for distributed updates and 'self' should either be r *this* server will not receive updates, or it should be a string value containing *this* server address if *this* server will recei updates - pass an empty list of servers to disable distribut update.

Response Codes

200: OK

Examples

Get the existing settings for the collection:

Input

```
curl http://localhost:8888/api/collections/collection1/settings
```

Output

```
{
    "auto_complete": true,
    "boost_recent": true,
    "boosts": [
        "recip(rord(lastModified),1,1000,1000)"
    ],
    "click_boost_data": "click-data",
    "click_boost_field": "click",
    "click_enabled": false,
    "de_duplication": "off",
    "default_sort": "relevance",
    "display_facets": true,
    "display_fields": [
        "id","url","author","data_source_type","lastModified",
        "mimeType","pageCount","title"],
    "elevations": {},
    "main_index_lock_type": "native",
    "main_index_max_buffered_docs": -1,
    "main_index_max_merge_docs": 2147483647,
    "main_index_merge_factor": 10,
    "main_index_ram_buffer_size_mb": 64.0,
    "main_index_term_index_interval": 32,
    "main_index_use_compound_file": false,
    "main_index_write_lock_timeout": 1000,
    "query_parser": "lucid",
    "query_time_stopwords": true,
    "query_time_synonyms": true,
    "search_server_list": [],
    "show_similar": true,
    "spellcheck": true,
    "stopword_list": [
        "a","an","and","are","as","at","be","but","by","for","if","in","into",
        "is","it","no","not","of","on","or","s","such","t","that","the",
        "their","then","there","these","they","this","to","was","will","with"],
    "synonym_list": [
        "lawyer, attorney","one, 1","two, 2","three, 3","ten, 10",
        "hundred, 100","thousand, 1000","tv, television"],
    "unknown_type_handling": "text_en",
    "unsupervised_feedback": false,
    "unsupervised_feedback_emphasis": "relevancy",
    "update_handler_autocommit_max_docs": null,
    "update_handler_autocommit_max_time": 3600000,
    "update_handler_autocommit_open_searcher": true,
    "update_handler_autosoftcommit_max_docs": null,
    "update_handler_autosoftcommit_max_time": null,
    "update_server_list": null
}
```

Get a Particular Setting

`GET /api/collections/collection/settings/name`

Returns a map of settings to values for a given setting.

Input

Path Parameters

Key	Description
collection	The collection name.
name	The name of the setting to return.

Query Parameters

None.

Output

Return Codes

200: OK

Examples

Determine the default parser for the collection.

Input

```
curl 'http://localhost:8888/api/collections/collection1/settings/query_parser'
```

Output:

```
{
   "query_parser":"lucid",
}
```

Update Settings

`PUT /api/collections/collection/settings`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input Content

JSON block with values for keys to be updated.

Key	Type	Description
auto_complete	boolean	Is **true** if auto-complete is enabled for use in the LucidWor Search default search interface. Note that this also require setting the auto-complete activity to run at regular interva For more information, see Auto-Complete of User Queries.
boosts	Solr function query	Defines the boost to apply to each query. The default boos the Lucid Query Parser prefers more recent documents.
boost_recent	boolean	Is **true** if the lucid request handler should boost recent documents.
click_enabled	boolean	Is **true** if Click is enabled (LucidWorks Search on-premise only).
click_boost_data	string	The path to Click boost data (LucidWorks Search on-premi only).
click_boost_field	string	The field name prefix used by Click fields (LucidWorks Sea on-premise only).
click_index_location	string	The path to Click boost index (LucidWorks Search on-prem only).
de_duplication	string	The valid values are: **off**, do not de-duplicate; **overwrite** duplicate documents; **tag** duplicated with a unique signatu Note that de-duplication does not work properly in SolrClo mode.
default_sort	string	Default sort method - valid values are: **relevance**, **date**, **random**.
display_facets	boolean	Is **true** if the LucidWorks Search default search interface s display facets.

display_fields	string	Defines the fields to use for display of results to users. Prir used to add pseudo-fields to documents, but could be usec "real" fields also. This parameter only applies when using t `lucid` handler (query parser).

elevations	JSON map	Defines the documents to be elevated or excluded from res for a specific query. It uses Solr's QueryElevationCompone which is enabled by default in LucidWorks. This API is an interface to manage the `elevate.xml` file, which stores the elevation definitions that are used for queries. The elevatic file is located in the `conf` directory for each collection (`$LWE_HOME/conf/solr/cores/collection/conf`)
		The structure of the `elevate.xml` file is an XML file defining query and the IDs of the documents that are to be elevate excluded. The API uses a JSON map to write to this file wit structure of:
		`{"elevations":`
		`   {"query":`
		`      [{"doc":"docID", "exclude":true}]`
		`   }`
		`}`
		The `exclude` attribute allows using the QueryElevationComponent to explicitly omit certain docum for specific queries. The default is **false**, so if this attribute not defined for a document, the document will be elevated the top of the result set for the specified query. Multiple documents elevated for a single query are elevated in the they are listed in the `elevate.xml` file. Set `exclude` to **true** exclude the document from the specified query.
		
		It is also possible to define elevations or exclusions using t built-in Search UI, which includes a "pin" or a "minus" nex every result to allow you to add it to the elevations list as a required document or an excluded document.
		
		The list of documents included or excluded is not synchron with the index, which means that if a document is listed in `elevate.xml` file, and then is removed from the index, it do not get removed from the file.

main_index_ lock_type	string	Defines which Lucene LockFactory to use. When applying changes to an index the IndexWriter requires a lock on the directory. The options are: <ul><li>**single**: using the SingleInstanceLockFactory. This is suggested for a read-only index or when there is no possibility of another process trying to modify the in</li><li>**native**: using the NativeFSLockFactory. This uses th native file locking. It should not be used when there multiple Solr web applications in the same JVM that attempting to share a single index.</li><li>**simple**: using the SimpleFSLockFactory. This uses a lock file for locking. More information is available in the Lucene Wiki: http://wiki.apache.org/lucene-java/AvailableLockFac .</li></ul>
main_index_ max_buffered_docs	integer	Allows setting the `maxBufferedDocs` parameter in the `solrconfig.xml` file for the collection, which sets the numb document updates to buffer in memory before they are flu to disk and added to the current index segment. It is gene preferred to use the `main_index_ram_buffer_size_mb`, but if settings are defined, a flush will occur when either limit is reached.
main_index_ max_merge_docs	integer	Allows setting the `maxMergeDocs` parameter in the `solrconfig.xml` file for the collection, which sets the maxin number of documents for a single segment. Once this limit reached, the segment is closed and a new one is created. segment merge, as defined by `main_index_merge_factor` m also occur at this time.
main_index_ merge_factor	integer	Allows setting the `mergeFactor` parameter in the `solrconfig` file for the collection, which defines how many segments tl index is allowed to have before they are coalesced into one segment. When the index is updated, the new data is adde the most recently opened segment. When that segment is a new segment is created and subsequent updates are pla there (defining when a segment is full is done with the `main_index_max_buffered_docs` and `main_index_ram_buffer_size_mb` settings). When the the `main_index_merge_factor` is reached, the segments are mer into a single larger segment. See the section on `mergeFact` the Solr Reference Guide for more information.

main_index_ ram_buffer_size_mb	integer	Allows setting the `ramBufferSizeMb` parameter in the `solrconfig.xml` file for the collection, which sets the amour memory space (in megabytes) document updates can use before they are flushed to the current index segment. This setting is generally preferable to `main_index_max_buffered_` but if both settings are defined, a flush will occur when eitl limit is reached.
main_index_ term_index_interval	integer	Allows setting the `TermIndexInterval` for the index and determines the amount of computation required per query term, regardless of the number of documents. This allows level of control over the time query processing takes. Larg values cause less memory to be used by the IndexReader, slows random-access to terms. Smaller values cause more memory to be used by the IndexReader, but will speed random-access to terms. A large index with user-entered queries may benefit from a larger `main_index_term_index_interval` because query processing dominated by frequency and positional data processing an by term lookup. A system that experiences a great deal of wildcard queries may benefit from a smaller value for this setting.
main_index_ use_compound_file	boolean	Allows you to set the `UseCompoundFile` parameter in the `solrconfig.xml` file for the collection. Setting this to **true** combines the multiple index files on disk to a single file. Tl setting would help avoid hitting an open file limit on those systems which restrict the number of open files allowed pe process. See the section on `UseCompoundFile` in the Solr Reference Guide for more information.
main_index_ write_lock_timeout	integer	Defines the maximum time to wait for a write lock.
query_parser	string	Which query parser the lucid search request handler will u valid values are: **lucid**, **dismax**, **extended dismax**, **luce**
query_time_stopwords	boolean	Is **true** if stopwords will be removed at query time.
query_time_synonyms	boolean	Is **true** if synonyms should be added to queries. This will c be used if the 'lucid' query parser is selected as the defaul used in the query request.
search_server_list	list:string	A list of Solr core URLs that the lucid request handler will u for distributed search - pass an empty list to disable distrik search.

show_similar	boolean	Is **true** if a "Find Similar" link should be displayed next to user's search results.
spellcheck	boolean	Is **true** if the LucidWorks Search default search interface s suggest spelling corrections.
stopword_list	list:string	A list of stopwords that will be used if 'query_time_stopwo is enabled.
synonym_list	list:string	A list of synonym rules that will be used if 'query_time_synonyms' is enabled.
unsupervised_feedback	boolean	Is **true** if unsupervised feedback is enabled
unsupervised_ feedback_emphasis	string	Defines if unsupervised feedback should emphasize "**relev** " which does an "AND" of the original query which neither includes nor excludes additional documents, or "**recall**" wh does an "OR" of the original query which permits the feedb terms to expand the set of documents matched - default is **relevancy**".
unknown_type_handling	string	A valid field type from the core's schema to use for unrecognized fields - default is **text_en**.
update_handler_ autocommit_max_docs	integer	Allows setting the `maxDocs` parameter for `autocommit` definit in the `solrconfig.xml` file for the collection. This setting de the number of documents to queue before pushing them to index. It works in conjunction with the `update_handler_autocommit_max_time` parameter in that if e limit is reached, the pending updates will be pushed to the index.
update_handler_ autocommit_max_time	integer	Allows setting the `maxTime` parameter for `autocommit` definit in the `solrconfig.xml` file for the collection. This setting de the number of milliseconds to wait before pushing docume to the index. It works in conjunction with the `update_handler_autocommit_max_docs` parameter in that if e limit is reached, the pending updates will be pushed to the index.
update_handler_ autocommit_open_searcher	boolean	Provides the option to not open a searcher on hard commi This may be useful to minimize the size of transaction logs keep track of uncommitted updates. The default is **true**, change this to **false** to not open a searcher.

update_handler_ autosoftcommit_max_docs	integer	Allows setting the `maxDocs` parameter for `autosoftcommit` definitions in the `solrconfig.xml` file for the collection. "So commits are used in Solr's Near RealTime searching. This setting defines the number of documents to queue before pushing them to the index. It works in conjunction with th {{update_handler_ autosoftcommit_max_time}} parameter in that if either li reached, the documents will be pushed to the index.
update_handler_ autosoftcommit_max_time	integer	Allows setting the `maxDocs` parameter for `autosoftcommit` definitions in the `solrconfig.xml` file for the collection. "So commits are used in Solr's Near RealTime searching. This setting defines the number of milliseconds to wait before pushing documents to the index. It works in conjunction w the `update_handler_autosoftcommit_max_docs` parameter in if either limit is reached, the documents will be pushed to index.
update_server_list	complex	A map that contains two keys: 'server_list' and 'self'. 'server_list' is list:string of servers that the lucid update c will use for distributed updates and 'self' should either be *this* server will not receive updates, or it should be a strin value containing *this* server address if *this* server will recei updates - pass an empty list of servers to disable distribut update.

Output

Output Content

None.

Return Codes

204: No Content

Examples

Turn on spell-checking for the collection.

Input

```
curl -X PUT -H 'Content-type: application/json'
-d '{"spellcheck":true}'
http://localhost:8888/api/collections/collection1/settings
```

Output

None. Check properties to confirm changes.

Caches

The Caches API allows creating, viewing, updating and deleting cache information configured in the `solrconfig.xml` file for the collection.

Caches are used in Solr to store information from the index for faster responses if the index information is needed again. They are associated with a specific instance of a Solr IndexSearcher, and the cached information is valid as long as the IndexSearcher is valid. When a new IndexSearcher is opened, it is usually auto-warmed (pre-populated) with data from the previous cache. The settings on this page help define the size of caches and how they are used to auto-warm a new IndexSearcher.

These are advanced settings, and modifying them should be done with care.

API Entry Points

`/api/collections/collection/caches`: get details of all configured caches or create a new cache.
`/api/collections/collection/chaches/name`: update, delete, or get details for a specific cache.

Get a List of Caches and Attributes for a Collection

```
GET /api/collections/collection/caches
```

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

A JSON List of Maps mapping Collection keys to values.

Key	Type	Description
name	string	The name of the cache. There are four known caches that are used in most implementations, which correspond to known Solr caches. These are: • **field_value_cache**, which implements Solr's fieldValueCache. This cache is mostly used for faceting. If it is not explicitly set in `solrconfig.xml`, then it is generated automatically with default values (initial_size = 10, max_size=10000, and autowarm_count = 0). • **document_cache**, which implements Solr's documentCache. This cache stores Lucene document objects. • **filter_cache**, which implements Solr's filterCache. This cache stores unordered sets of all documents for a query, and is used by Solr for filter queries (and thus by LucidWorks for the Search Filter functionality). It can also be used for faceting, sorting, or other situations that require the full set of documents that match a query. • **query_result_cache**, which implements Solr's queryResultCache. This cache stores ordered sets of document IDs from previous searches. A custom cache implementation can also be created and used if a Solr plugin has been created for your application.
class	string	The cache implementation that's used for this cache. There are two available types, **solr.LRUCache** and **solr.FastLRUCache**. The other properties for the cache depend on the implementation that is used. See the following table for details of attributes that can be used for each cache implementation. The "LRU" part of the cache class name stands for "Least Recently Used". When an LRU cache fills up, the entry with the oldest last-accessed timestamp is removed to make room for the new entry. Frequently used items tend to remain cached, while less-used items will drop out of cache to be retrieved again later if they are needed. The FastLRUCache is meant to be lock-less, so is well-suited for caches that are hit several times in a request.
regenerator	string	A valid class name which specifies how to re-populate a new cache created by a new Searcher with old cache objects. Usually only declared for custom caches.

`solr.LRUCache` **Attributes**

Key	Type	Description
size	integer	The maximum number of entries in the cache.

initial_size	integer	The initial number of entries in the cache.
autowarm_count	integer	The number of entries to pre-populate from an old cache, or can be a percentage of the old cache. When a new Searcher is opened, it may be pre-populated with cached objects from the old Searcher. A best practice is to base the autowarm_count on how long it takes to autowarm a new Searcher.

`solr.FastLRUCache` **Attributes**

Key	**Type**	**Description**
size	integer	The maximum number of entries in the cache.
initial_size	integer	The initial number of entries in the cache.
autowarm_count	integer	The number of entries to pre-populate from an old cache, or can be a percentage of the old cache. When a new Searcher is opened, it may be pre-populated with cached objects from the old Searcher. A best practice is to base the autowarm_count on how long it takes to autowarm a new Searcher.
min_size	integer	When the cache hits it's `size` limit, this allows the cache to try to reduce to this value. The default is 0.9 * `size`.
acceptable_size	integer	When the cache removes old entries, it tries to achieve the `min_size`. If that is not possible, it tries to achieve `acceptable_size` instead. The default is 0.95 * `size`.
cleanup_thread	boolean	If set to **true**, cache cleanup will run in a dedicated separate thread. Very large cache sizes may perform better if they are in a dedicated thread.
show_items	integer	Used to debug what's contained in the cache. Will return the last N accessed items in the Solr Admin UI, using an MBeans Request Handler or JMX. A value of "-1" will display all items in the cache.

Response Codes

200: OK

Examples

Input

```
curl http://localhost:8888/api/collections/collection1/caches
```

Output

```
[
{
 "initial_size":"512",
 "name":"document_cache",
 "class":"solr.LRUCache",
 "autowarm_count":"0",
 "size":"512"
},
{
 "initial_size":"512",
 "name":"filter_cache",
 "class":"solr.LRUCache",
 "autowarm_count":"256",
 "size":"512"
},
{
 "initial_size":"512",
 "name":"query_result_cache",
 "class":"solr.LRUCache",
 "autowarm_count":"256",
 "size":"512"
},
{
 "acceptable_size":null,
 "initial_size":"10",
 "name":"field_value_cache",
 "class":"solr.FastLRUCache",
 "cleanup_thread":null,
 "autowarm_count":null,
 "show_items":"-1",
 "min_size":null,
 "size":"10000"
}
]
```

Create a Cache and Attributes for a Collection

`POST /api/collections/collection/caches`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.

Query Parameters

None.

Input Content

A JSON List of Maps mapping Collection keys to values.

Key	Type	Description
name	string	The name of the cache. There are four known caches that are used in most implementations, which correspond to known Solr caches. These are: • **field_value_cache**, which implements Solr's fieldValueCache. This cache is mostly used for faceting. If it is not explicitly set in `solrconfig.xml`, then it is generated automatically with default values (initial_size = 10, max_size=10000, and autowarm_count = 0). • **document_cache**, which implements Solr's documentCache. This cache stores Lucene document objects. • **filter_cache**, which implements Solr's filterCache. This cache stores unordered sets of all documents for a query, and is used by Solr for filter queries (and thus by LucidWorks for the Search Filter functionality). It can also be used for faceting, sorting, or other situations that require the full set of documents that match a query. • **query_result_cache**, which implements Solr's queryResultCache. This cache stores ordered sets of document IDs from previous searches. A custom cache implementation can also be created and used if a Solr plugin has been created for your application.
class	string	The cache implementation that's used for this cache. There are two available types, **solr.LRUCache** and **solr.FastLRUCache**. The other properties for the cache depend on the implementation that is used. See the following table for details of attributes that can be used for each cache implementation. The "LRU" part of the cache class name stands for "Least Recently Used". When an LRU cache fills up, the entry with the oldest last-accessed timestamp is removed to make room for the new entry. Frequently used items tend to remain cached, while less-used items will drop out of cache to be retrieved again later if they are needed. The FastLRUCache is meant to be lock-less, so is well-suited for caches that are hit several times in a request.
regenerator	string	A valid class name which specifies how to re-populate a new cache created by a new Searcher with old cache objects. Usually only declared for custom caches.

{`solr.LRUCache` **Attributes**

Key	Type	Description
size	integer	The maximum number of entries in the cache.
initial_size	integer	The initial number of entries in the cache.
autowarm_count	integer	The number of entries to pre-populate from an old cache, or can be a percentage of the old cache. When a new Searcher is opened, it may be pre-populated with cached objects from the old Searcher. A best practice is to base the autowarm_count on how long it takes to autowarm a new Searcher. If setting to a percentage, be sure to enclose the value in quotes, such as `"autowarm":"50%"`. Setting to a number of entries does not require quotes.

`solr.FastLRUCache` **Attributes**

Key	Type	Description
size	integer	The maximum number of entries in the cache.
initial_size	integer	The initial number of entries in the cache.
autowarm_count	integer	The number of entries to pre-populate from an old cache, or can be a percentage of the old cache. When a new Searcher is opened, it may be pre-populated with cached objects from the old Searcher. A best practice is to base the autowarm_count on how long it takes to autowarm a new Searcher. If setting to a percentage, be sure to enclose the value in quotes, such as `"autowarm":"50%"`. Setting to a number of entries does not require quotes.
min_size	integer	When the cache hits it's `size` limit, this allows the cache to try to reduce to this value. The default is 0.9 * `size`.
acceptable_size	integer	When the cache removes old entries, it tries to achieve the `min_size`. If that is not possible, it tries to achieve `acceptable_size` instead. The default is 0.95 * `size`.
cleanup_thread	boolean	If set to **true**, cache cleanup will run in a dedicated separate thread. Very large cache sizes may perform better if they are in a dedicated thread.
show_items	integer	Used to debug what's contained in the cache. Will return the last N accessed items in the Solr Admin UI, using an MBeans Request Handler or JMX. A value of "-1" will display all items in the cache.

Output

Output Content

None.

Response Codes

List valid response codes and meaning

Examples

Input

```
curl -H 'Content-type: application/json' -d
'{"name":"newcache","class":"solr.LRUCache","size":500}'
http://localhost:8888/api/collections/collection1/caches
```

Output

None.

Update Cache Attributes

`PUT /api/collections/collection/caches/name`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.
name	The cache name.

Query Parameters

None.

Input Content

A JSON List of Maps mapping Collection keys to values.

Key	Type	Description

name	string	The name of the cache. There are four known caches that are used in most implementations, which correspond to known Solr caches. These are: • **field_value_cache**, which implements Solr's fieldValueCache. This cache is mostly used for faceting. If it is not explicitly set in `solrconfig.xml`, then it is generated automatically with default values (initial_size = 10, max_size=10000, and autowarm_count = 0). • **document_cache**, which implements Solr's documentCache. This cache stores Lucene document objects. • **filter_cache**, which implements Solr's filterCache. This cache stores unordered sets of all documents for a query, and is used by Solr for filter queries (and thus by LucidWorks for the Search Filter functionality). It can also be used for faceting, sorting, or other situations that require the full set of documents that match a query. • **query_result_cache**, which implements Solr's queryResultCache. This cache stores ordered sets of document IDs from previous searches. A custom cache implementation can also be created and used if a Solr plugin has been created for your application.
class	string	The cache implementation that's used for this cache. There are two available types, **solr.LRUCache** and **solr.FastLRUCache**. The other properties for the cache depend on the implementation that is used. See the following table for details of attributes that can be used for each cache implementation. The "LRU" part of the cache class name stands for "Least Recently Used". When an LRU cache fills up, the entry with the oldest last-accessed timestamp is removed to make room for the new entry. Frequently used items tend to remain cached, while less-used items will drop out of cache to be retrieved again later if they are needed. The FastLRUCache is meant to be lock-less, so is well-suited for caches that are hit several times in a request.
regenerator	string	A valid class name which specifies how to re-populate a new cache created by a new Searcher with old cache objects. Not needed for most implementations.

`solr.LRUCache` **Attributes**

Key	**Type**	**Description**
size	integer	The maximum number of entries in the cache.
initial_size	integer	The initial number of entries in the cache.

autowarm_count	integer	The number of entries to pre-populate from an old cache, or can be a percentage of the old cache. When a new Searcher is opened, it may be pre-populated with cached objects from the old Searcher. A best practice is to base the autowarm_count on how long it takes to autowarm a new Searcher. If setting to a percentage, be sure to enclose the value in quotes, such as `"autowarm":"50%"`. Setting to a number of entries does not require quotes.

`solr.FastLRUCache` **Attributes**

Key	Type	Description
size	integer	The maximum number of entries in the cache.
initial_size	integer	The initial number of entries in the cache.
autowarm_count	integer	The number of entries to pre-populate from an old cache, or can be a percentage of the old cache. When a new Searcher is opened, it may be pre-populated with cached objects from the old Searcher. A best practice is to base the autowarm_count on how long it takes to autowarm a new Searcher. If setting to a percentage, be sure to enclose the value in quotes, such as `"autowarm":"50%"`. Setting to a number of entries does not require quotes.
min_size	integer	When the cache hits it's `size` limit, this allows the cache to try to reduce to this value. The default is 0.9 * `size`.
acceptable_size	integer	When the cache removes old entries, it tries to achieve the `min_size`. If that is not possible, it tries to achieve `acceptable_size` instead. The default is 0.95 * `size`.
cleanup_thread	boolean	If set to **true**, cache cleanup will run in a dedicated separate thread. Very large cache sizes may perform better if they are in a dedicated thread.
show_items	integer	Used to debug what's contained in the cache. Will return the last N accessed items in the Solr Admin UI, using an MBeans Request Handler or JMX. A value of "-1" will display all items in the cache.

Output

Output Content

None.

Response Codes

200: OK
415: Unsupported media type (if attribute value is not valid)
422: Unprocessable entity

Examples

Input

```
curl -X PUT -H 'Content-type: application/json' -d '{"autowarm_count":"50%"}'
http://localhost:8888/api/collections/collection1/caches/newcache
```

Output

None.

Delete a Cache

`GET /api/collections/collection/caches/name`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.
name	The cache name.

Query Parameters

None.

Output

Output Content

None.

Response Codes

None.

Examples

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/caches/newcache
```

Click Scoring

The Click Scoring API allows programmatic access to record click and query "events", which can be used when developing a custom search application to record user click behavior and use it to impact relevance scores following data processing.

This API does not enable or disable Click Scoring, nor kick off log processing. Those actions are covered by the Settings and Activities APIs, respectively. More details about Click Scoring are available in the section on the Click Scoring Relevance Framework.

API Entry Points

`/api/collection/collection/click`: get statistics about recent Click Scoring events (queries or clicks) or record events.

Get Recent Events

`GET /api/collections/collection/click`

Input

Path Parameters

Enter path parameters.

Key	Description
collection	The collection name.

Query Parameters

None.

Output

Output Content

Key	Type	Description
buffer_size	integer	The number of events in the buffer, if it is enabled.

buffering	boolean	Is **true** if buffering has been enabled. Buffering allows the Click Scoring process to store events in memory for later writing to the log files. It's meant to be used as a temporary measure and is most useful when log files are being moved during the Click Scoring analysis processes. When this is changed to **false** events in the buffer are flushed to the log files.
click_count	integer	The number of click events recorded.
logs_count	integer	The number of Click-related log files in use.
query_count	integer	The number of query events recorded.

Response Codes

200: OK

Examples

Input

```
curl http://localhost:8888/api/collections/collection1/click
```

Output

```
{
    "buffer_size": 0,
    "buffering": false,
    "click_count": 3,
    "logs_count": 1,
    "query_count": 4
}
```

Record User Queries and Clicks

The PUT request makes an entry into the `click-collection.log` file, which is where all queries and clicks are recorded for later processing.

`PUT /api/collections/collection/click`

Input

Path Parameters

Enter path parameters.

Key	Description

collection	The collection name.

Query Parameters

None.

Output

Output Content

Key	Type	Description
type	string	The type of event being recorded. A value of "**q**" indicates a query event; a value of "**c**" indicates a click event.
req	string	A unique request ID. It is used for both types of events.
q	string	Used for query events, the user's query.
qt	long	Used for query events, the time of the query, in milliseconds since epoch.
hits	integer	Used for query events, the number of hits for the query.
u	string	Optionally used for query events, the user ID.
ct	long	Used for click events, the time of the click, in milliseconds since epoch.
doc	string	Used for click events, the document ID that the user selected.
pos	integer	Used for click events, the position of the document in the result list.

Response Codes

200: Success

Examples

Input

Example Query Event:

```
curl -X PUT -H 'Content-type: application/json' -d '
{
   "type":"q",
   "req":"34509586770",
   "q":"ipod",
   "qt":1329157201,
   "hits":545
}' http://localhost:8888/api/collections/collection1/click
```

Example Click Event:

```
curl -X PUT -H 'Content-type: application/json' -d '
{
   "type":"c",
   "req":"34509598766",
   "ct":1329157350,
   "doc":"http://www.apple.com",
   "pos":6
}' http://localhost:8888/api/collections/collection1/click
```

Output

Corresponding entries in the `click-collection.log` file:

```
Q       1329157201      ipod     none~34509586770        545
C       1329157350      none~34509598766        http://www.apple.com    6
```

Roles

With LucidWorks Search, roles are used in conjunction with search filters to restrict the set of documents appearing in search results for users to a particular subset of documents in the index based on their username or LDAP-supplied group membership. This API provides a way to programmatically manage roles. Note that this functionality is also available in the Admin UI, where it is called "Search Filters".

By default, LucidWorks Search contains one role, called DEFAULT, that is allowed to search all documents and the default "admin" user is added to it. The included Search UI requires a role, but we have configured LucidWorks so if a user is not included in a role, they inherit DEFAULT

API Entry Points

`/api/collections/collection/roles`: retrieve existing roles or add new roles

`/api/collections/collection/roles/role`: retrieve, update, or remove roles

Retrieve Existing Roles

`GET /api/collections/collection/roles`

Returns a list of role maps. Each map containing the role name, a list of users, a list of groups the role maps to, and a list of filters to apply when a given role reads the index.

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Output

Output Content

Key	Type	Description
name	string	Name of the role.
groups	list:string	A list of groups for this role.
users	list:string	A list of users for this role.
filters	list:string	A list of filters for this role. Note that filters use Solr's filter query capability, which means they also use the default Lucene query parser instead of the include lucid parser.

Return Codes

200: OK

Examples

Get a list of the existing roles.

Input

```
curl 'http://localhost:8888/api/collections/collection1/roles'
```

Output

```
[
  {
    "groups": [
    ],
    "name": "DEFAULT",
    "filters": [
      "*:*"
    ],
    "users": [
      "admin"
    ]
  }
]
```

Create a New Role

`POST /api/collections/collection/roles`

Input

Path Parameters

Key	Description
collection	The collection name.

Query Parameters

None

Input Content

A JSON map with the following keys.

Key	Type	Description
name	string	Name of the role.
groups	list:string	A list of groups for this role.
users	list:string	A list of users for this role.
filters	list:string	A list of filters for this role.

Output

Output Content

Key	Type	Description
name	string	Name of the role.
groups	list:string	A list of groups for this role.
users	list:string	A list of users for this role.
filters	list:string	A list of filters for this role.

Return Codes

201: Created

Examples

Create a new role that only shows documents that have a `status` field of "public", and assign it to the `group1` and `group2` groups, and to `user1`.

Input

```
curl -H 'Content-type: application/json' -d '
{
  "name": "ONLY_PUBLIC",
  "groups": [
    "group1",
    "group2"
  ],
  "filters": [
    "status:public"
  ],
  "users": [
    "user1"
  ]
}' 'http://localhost:8888/api/collections/collection1/roles'
```

Output

```
{
  "name": "ONLY_PUBLIC",
  "groups": [
    "group1",
    "group2"
  ],
  "filters": [
    "status:public"
  ],
  "users": [
    "user1"
  ]
}
```

Retrieve an Existing Role

`GET /api/collections/collection/roles/role`

Input

Path Parameters

Key	Description
collection	The collection name.
role	The role name.

Query Parameters

None

Input Content

None.

Output

Output Content

Key	Type	Description
name	string	Name of the role.
groups	list:string	A list of groups for this role.
users	list:string	A list of users for this role.
filters	list:string	A list of filters for this role.

Response Codes

200: OK

404: Not Found

Examples

Get the details for the `ONLY_PUBLIC` role:

Input

```
curl 'http://localhost:8888/api/collections/collection1/roles/ONLY_PUBLIC'
```

Output

```
{
  "name": "ONLY_PUBLIC",
  "groups": [
    "group1",
    "group2"
  ],
  "filters": [
    "status:public"
  ],
  "users": [
    "user1"
  ]
}
```

Update a Role

`PUT /api/collections/collection/roles/role`

Input

Path Parameters

Key	Description
collection	The collection name.
role	The role name.

Query Parameters

None

Input Content

A JSON map with the following keys.

Key	Type	Description
groups	list:string	A list of groups for this role.
users	list:string	A list of users for this role.
filters	list:string	A list of filters for this role.

Output

Output Content

None.

Response Codes

204: No Content

404: Not Found

Examples

Remove `group1` from the `ONLY_PUBLIC` role.

Input

```
curl -X PUT -H 'Content-type: application/json' -d '
{
  "groups": [
    "group2"
  ]
}' 'http://localhost:8888/api/collections/collection1/roles/ONLY_PUBLIC'
```

Output

None. (Check properties to confirm changes.)

Delete a Role

`DELETE /api/collections/collection/roles/role`

Input

Path Parameters

Key	Description
collection	The collection name.
role	The role name.

Query Parameters

None

Output

Output Content

None.

Return Codes

204: No Content

404: Not Found

Examples

Delete the `ONLY_PUBLIC` role.

Input

```
curl -X DELETE  'http://localhost:8888/api/collections/collection1/roles/ONLY_PUBLIC'
```

Output

None.

Alerts API

Enterprise Alerts are a way for users to create a search and save it so LucidWorks notifies them when new content has been added to the index. The Alerts API provides a way to create, update, or list user alerts so that you can manage them programmatically.

Even if using the Alerts API to define alerts for users, the Application Server and Outgoing Mail Server must be configured in the System Settings screen of the Admin UI before alerts will successfully run and notify users.

- API Entry Points
- Create an Alert
- List all Alerts

- List Alerts for a User
- View Details of a Single Alert
- Run an Alert to Get New Results
- Update the Details of an Alert
- Delete an Alert

API Entry Points

`/api/alerts/`: create an alert, or get a list of all alerts, or list alerts for a specific user

`/api/alerts/id`: view details, update or delete a single alert

`/api/alerts/id/check`: run an alert to find new results

API Address

Unlike the other REST APIs with LucidWorks Search, the Alerts API uses the LWE-UI component. If you followed a default installation, this component runs at `http://localhost:8989/`. Modify this URL as needed to match the location of the LWE-UI component if it is not at the default address.

Create an Alert

```
POST /api/alerts
```

Input

Path Parameters

None.

Query Parameters

None.

Input Content

Key	Type	Required	Default	Description
name	string	Yes	null	A user-selected name for the alert.
collection	string	Yes	null	The collection to be searched by the alert.
username	string	Yes	null	The name of the user who will receive alerts.
query	string	Yes	null	The query string.

period	32 bit integer	No	null	Frequency to update the alert, in seconds. If period is not specified, the alert will only be checked when the user selects it to be checked. In this sense it becomes more of a "saved search".
email	string	No	null	The email address to send alert messages to. If a period is entered, an email address is required.
filters	JSON map	No	null	Filters are generally made from facets displayed as part of results, but can be made from any field that contains indexed content from documents.

Output

Output Content

Key	Type	Description
id	integer	A unique id for the alert
name	string	A user-selected name for the alert.
collection	string	The collection to be searched by the alert.
username	string	The name of the user who will receive alerts.
query	string	The query string.
checked_at	date	A timestamp of when the alert was last checked.
period	32 bit integer	Frequency the alert will be updated, in seconds. If period is not specified, the alert will only be checked when the user selects it to be checked. In this sense it becomes more of a "saved search".
email	string	The email address to send alert messages to. If a period is entered, an email address is required.
filters	JSON map	Filters are generally made from facets displayed as part of results, but can be made from any field that contains indexed content from documents.

Return Codes

201: Created

Examples

Create an alert that searches every 24 hours for documents with the word "solr".

Input

```
curl -iX POST -H 'Content-type: application/json' http://localhost:8989/api/alerts
-d '{"name":"Solr
documents","collection":"collection1","username":"admin","query":"solr",
"period":86400,"email":"test@test.com"}'
```

Output

```
{
  "id":3,
  "name":"Solr documents",
  "collection":"collection1",
  "username::"admin",
  "query":"solr",
  "checked_at":"2011-09-09T18:11:37Z"
}
```

List all Alerts

```
Get /api/alerts
```

Input

Path Parameters

None.

Query Parameters

None.

Input Content

None.

Output

Output Content

Key	Description
id	The unique ID of the alert.
collection	The collection that is queried for the alert.
username	The username for the owner of the alert.
query	The search string.

period	How frequently this alert will run (if set).
email	The email address that will be sent notifications.
checked_at	The timestamp of the last time the alert was run.
filters	The filters that will be added to the query string.

Return Codes

201: Created

Examples

Input

```
curl -iX GET -H 'Accept: application/json' http://localhost:8989/api/alerts
```

Output

```
{
   "id":1,
   "name":"Solr documents",
   "collection":"collection1",
   "username":"admin",
   "query":"solr",
   "period":86400,
   "email":"test@test.com",
   "checked_at":"2011-09-01T19:49:48Z"
}
```

List Alerts for a User

`GET /api/alerts?username=username`

Input

Path Parameters

None.

Query Parameters

Key	Description
username	The username associated with the alert.

Output

Output Content

Key	Description
id	The unique ID of the alert.
name	The user-defined name of the alert.
collection	The collection that is queried for the alert.
username	The username for the owner of the alert.
query	The search string.
period	How frequently this alert will run.
email	The email address that will be sent notifications.
checked_at	The timestamp of the last time the alert was run.
filters	The filters that will be added to the query string.

Return Codes

200: OK

Examples

List all alerts for the user with a username of "jdoe".

Input

```
curl -i http://localhost:8989/api/alerts?username=jdoe
```

Output

```
{
   "id":6,
   "name":"Solr PDF Docs"
   "collection":"collection1",
   "username":"jdoe",
   "query":"solr",
   "period":86400,
   "email":"test@test.com",
   "checked_at":"2011-09-01T19:49:48Z"
   "filters":
      {"mimeType":["application/pdf"]}
}
```

Note that this is a mutli-valued response.

View Details of a Single Alert

GET /api/alerts/id

Input

Path Parameters

Key	Description
id	The unique ID of the alert

Query Parameters

None.

Output

Output Content

Key	Description
id	The unique ID of the alert.
name	The user-defined name of the alert.
collection	The collection that is queried for the alert.
username	The username for the owner of the alert.
query	The search string.
period	How frequently this alert will run.
email	The email address that will be sent notifications.
checked_at	The timestamp of the last time the alert was run.
filters	The filters that will be added to the query string. This will only be shown if there are any filters defined.

Return Codes

200: OK

Examples

Get the details of alert number '1':

Input

```
curl -i http://localhost:8989/api/alerts/1
```

Output

```
{
   "id":1,
   "name":"Solr documents",
   "collection":"collection1",
   "username":"admin",
   "query":"solr",
   "period":86400,
   "email":"test@test.com",
   "checked_at":"2011-09-01T19:49:48Z"
}
```

Run an Alert to Get New Results

`PUT /api/alerts/id/check`

Input

Path Parameters

Key	Description
id	The unique ID of the alert.

Query Parameters

None.

Output

Output Content

Key	Description
id	The unique ID of the alert.
name	The user-defined name of the alert.
collection	The collection that is queried for the alert.
username	The username for the owner of the alert.
query	The search string.

period	How frequently this alert is run.
email	The email address that will be sent notifications.
checked_at	The timestamp of the last run.
filters	The filters that will be added to the query string. This will only be shown if there are any filters defined.
results	The result list, formatted in a JSON block with the responseHeader, response, highlighting, facet counts, and spell check.

Return Codes

200: OK

Examples

Get the results of alert number 1:

Input

```
curl -iX PUT http://localhost:8989/api/alerts/1/check
```

Output

The example below has been shortened - real output will be much longer depending on how many documents there are that match the query.

```
{
   "id":1,
   "name":"Solr documents",
   "collection":"collection1",
   "username":"admin",
   "query":"solr",
   "period":86400,
   "email":"test@test.com",
   "checked_at":"2011-09-01T20:17:19Z",
   "results":{
     "responseHeader":{
      ...
     },
     "response":{
      ...
     },
     "highlighting":{
      ...
     },
     "facet_counts":{
      ...
     },
     "spellcheck":{
      ...
     }
   }
}
```

Update the Details of an Alert

`PUT /api/alerts/id`

Input

Path Parameters

Key	Description
id	A unique ID of the alert

Query Parameters

None.

Input Content
Only fields that will be updated need to be included in the request.

Key	Description
id	The unique ID of the alert.
name	The user-defined name of the alert.
collection	The collection that is queried for the alert.
username	The username for the owner of the alert.
query	The search string.
period	How frequently this alert is run.
email	The email address that will be sent notifications.
filters	The filters that will be added to the query string. This will only be shown if there are any filters defined.

Output

Output Content

Key	Description
id	The unique ID of the alert.
name	The user-defined name of the alert.
collection	The collection that is queried for the alert.
username	The username for the owner of the alert.
query	The search string.
checked_at	The timestamp of the last time the alert was checked.
period	How frequently this alert is run.
email	The email address that will be sent notifications.
filters	The filters that will be added to the query string. This will only be shown if there are any filters defined.

Return Codes

200: OK

Examples

Change the query for the "City Alert" to "San Francisco".

Input

```
curl -iX PUT -H 'Content-type: application/json' -d '{"query":"San Francisco"}'
'http://localhost:8989/api/alerts/3'
```

Output

```
{
   "id":3,
   "name":"City alert",
   "collection":"collection1",
   "username":"smiller",
   "query":"San Francisco",
   "period":86400,
   "email":"test@test.com",
   "checked_at":"2011-09-01T19:49:48Z"
}
```

Delete an Alert

`DELETE /api/alerts/id`

Input

Path Parameters

Key	Description
id	The unique ID of the alert

Query Parameters

None.

Output

Output Content

None.

Return Codes

204: No Content

Examples

Delete the City Search alert, number 3.

Input

```
curl -iX DELETE 'http://localhost:8989/api/alerts/3'
```

Output

None.

Users

The Users API allows you to manage users in LucidWorks Search who are manually created in the LucidWorks internal user database. You do not need to use this API if your installation is configured to check user authentication against an LDAP server.

Unlike most of the other APIs in LucidWorks Search, this API uses the same port as the LWE-UI component. If installed with the default port, that would be at `http://localhost:8989/`.

API Entry Points

`/api/users`: create a user or get all users

`/api/users/username`: update, get, or delete a user

Get All Users

`GET /api/users`

Input

Path Parameters

None

Query Parameters

None

Output

Output Content

Key	Type	Description
username	string	The username for the user. Each username is unique.
email	string	The user's email address.
authorization	string	Either **admin** or **search**. Users with 'admin' authorization can access all parts of the LucidWorks Search UI (Admin UI and Search UI); users with 'search' authorization can only access the Search UI. The authorization is case-sensitive and is always all lower-case.
encrypted_password	string	A secure hash of the user's password.

Return Codes

200: OK
422: Unprocessable Entity (with cause of error)

Examples

Get a listing of the existing users in the system.

Input

```
curl 'http://localhost:8989/api/users'
```

Output

```
[
  {
    "username": "tommickle",
    "email": "mickle@here.com",
    "authorization": "admin",
    "encrypted_password":
"$2a$10$LahkxlPD809eG3tThMoZbe.ceQteNcpyEdhmcUELTyBBSgDqmNSQ6"
  },
  {
    "username": "admin",
    "email": "admin@localhost.com",
    "authorization": "admin",
    "encrypted_password":
"$2a$10$LahkxlPD809eG3tThMoZbe.ceQteNcpyEdhmcUELTyBBSgDqmNSQ6"
  },
  {
    "username": "suser",
    "email": "john@there.com",
    "authorization": "search",
    "password": "$2a$10$l1TBrGT/1xXW1cay0HeHe.RmcaH3KFZyGKVQUTVV6eRpn1857ncKm"
  }
]
```

Create a New User

`POST /api/users`

Input

Path Parameters

None.

Query Parameters

None.

Input Content

JSON block with all fields.

Key	Type	Description
username	string	The username for the user. Each username must be unique.
email	string	The user's email address.

authorization	string	Either **admin** or **search**. Users with 'admin' authorization can access all parts of the LucidWorks Search UI (Admin UI and Search UI); users with 'search' authorization can only access the Search UI. The value for `authorization` must always be entered in all lower-case letters.
password	string	The user's password.
encrypted_password	string	An alternate to password: a secure hash of the user's password.

Output

Output Content

JSON representing the new user.

Key	Type	Description
id	integer	The unique ID for the user.
username	string	The user's username.
email	string	The user's email address.
authorization	string	The authorization for the user.
encrypted_password	string	The user's password.

Return Codes

201: Created

Examples

Create a new user.

Input

```
curl -H 'Content-type: application/json' -d '
{
  "username": "smiller",
  "email": "me@here.com",
  "authorization": "search",
  "password": "123456"
}' 'http://localhost:8989/api/users'
```

Output

```
{
   "username":"smiller",
   "email":"me@here.com",
   "authorization":"search",
   "encrypted_password":"$2a$10$l1TBrGT/1xXW1cay0HeHe.RmcaH3KFZyGKVQUTVV6eRpn1857ncKm"
}
```

Get Information About a User

`GET /api/users/username`

Input

Path Parameters

Key	Description
username	The unique username of the user.

Query Parameters

None.

Output

Return Codes

200: OK

Examples

Get information on the `smiller` user.

Input

```
curl 'http://localhost:8989/api/users/smiller'
```

Output

```
{
  "username": "smiller",
  "email": "me@here.com",
  "authorization": "search",
  "password": "$2a$10$l1TBrGT/1xXW1cay0HeHe.RmcaH3KFZyGKVQUTVV6eRpn1857ncKm"
}
```

Update a User

`PUT /api/users/username`

Input

Path Parameters

Key	Description
username	The unique username of the user to be updated.

Query Parameters

None.

Input Content

Not all attributes need to be passed, but they could if they all need to be updated.

Key	Type	Description
username	string	The username for the user.
email	string	The user's email address.
authorization	string	Either **admin** or **search**. Users with 'admin' authorization can access all parts of the LucidWorks Search UI (Search and Admin); users with 'search' authorization can only access the Search UI. The value for authorization must always be entered in all lower-case letters.
password	string	The user's password.
encrypted_password	string	An alternate to password: a secure hash of the user's password.

Output

Output Content

None.

Return Codes

204: No Content

Examples

Change the authorization for the `smiller` username to "admin", and change the password:

Input

```
curl -X PUT -H 'Content-type: application/json' -d
'{
     "authorization":"admin",
     "password": "batman"
}'
'http://localhost:8989/api/users/smiller'
```

Output

None. Check properties to confirm changes.

Remove a User

`DELETE /api/users/username`

Input

Path Parameters

Key	Description
username	The unique username of the user to remove

Query Parameters

None.

Input content

None.

Output

Output Content

None.

Return Codes

204: No Content

404: Not Found

Examples

Delete the user `smiller`.

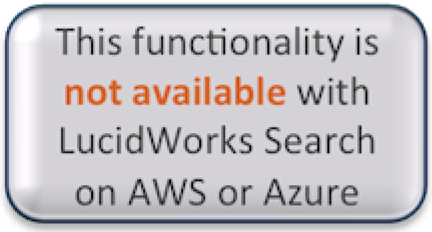
Input

```
curl -X DELETE 'http://localhost:8989/api/users/smiller'
```

Output

None.

SSL Configuration


This functionality is not available with LucidWorks Search on AWS or Azure

The SSL Configuration API allows you to work with some of the LucidWorks Search Secure Socket Layer-related settings. This API does not support configuring the Container-related settings. For more information about configuring the container-related SSL settings, see Enabling SSL.

It is possible to configure LucidWorks Search to allow only mutually authenticated SSL traffic. This feature is controlled with the parameters `auth_require_authorization` and `auth_authorized_clients`. When you set `auth_require_authorization` to true you can control which clients are allowed to access LucidWorks Search by listing the DNs from the certificates in `auth_authorized_clients`.

When using SSL in distributed environment the Solr internal client must be configured to allow it to access other nodes. The client keystore can be configured with parameters `client_keystore_url` and `client_keystore_password`. The client truststore can be configured with `client_truststore_url` and `client_truststore_password`. If there's a need to further limit down the client access to just some clients `auth_require_authorization` and `auth_authorized_clients` can be used.

- API Entry Points
- List The Existing SSL Configuration
- Update SSL Configuration

API Entry Points

`api/config/ssl`: List or update the existing SSL configuration.

List The Existing SSL Configuration

```
GET api/config/ssl
```

Input

Path Parameters

None.

Query Parameters

None.

Output

Output Content

JSON block with these parameters:

Key	Type	Required	Default	Description
client_truststore_url	url	no	null	Client truststore location, only required when using Mutually Authenticated SSL (Server)
client_keystore_url	url	no	null	Client keystore location, only required when using Mutually Authenticated SSL (Client)
auth_require_authorization	boolean	no		Enforces client authorization (with certificates). When enabled, only clients that are listed in `auth_authorized_clients` are allowed to access `/api` and `/solr` paths.
auth_authorized_clients	array of strings	no	[]	Lists authorized clients (certificate DN), only relevant when auth_require_authorization is set to true.

Response Codes

Examples

List SSL configuration:

Input

```
curl 'http://localhost:8888/api/config/ssl'
```

Output

```
{
  "client_truststore_url":"file:/truststore_url",
  "client_keystore_url":"file:/keystore_url",
  "auth_require_authorization":false,
  "auth_authorized_clients":["cn=foo"],
}
```

Update SSL Configuration

`PUT /api/config/ssl`

Input

Path Parameters

None.

Query Parameters

None.

Input Content

JSON block with these parameters:

Key	Type	Required	Default	Description
client_truststore_url	url	no	null	Client truststore location, only required when using Mutually Authenticated SSL (Server)
client_truststore_password	sting	no	null	Client truststore password, only required when using Mutually Authenticated SSL (Server)
client_keystore_url	url	no	null	Client keystore location, only required when using Mutually Authenticated SSL (Client)
client_keystore_password	sting	no	null	Client keystore password, only required when using Mutually Authenticated SSL (Server)
auth_require_authorization	boolean	no		Enforces client authorization (with certificates). When enabled, only clients that are listed in `auth_authorized_clients` are allowed to access `/api` and `/solr` paths.

auth_authorized_clients	array of strings	no	[]	Lists authorized clients (certificate DN), only relevant when auth_require_authorization is set to true.

Output

Output Content

None.

Response Codes

201 Success
422 Problem with Input

Examples

Configure LucidWorks Search Solr client so that distributed search uses SSL:

Input

```
curl -X PUT -H 'Content-type: application/json' -d '{
  "client_truststore_url":"file:/path_to_keystore/keystore.client",
  "client_truststore_password":"password",
  "client_keystore_url":"file:/path_to_truststore/truststore.client",
  "client_keystore_password":"password",
}' http://localhost:8888/api/config/ssl
```

Output

None.

Crawler Status

The crawler status API allows checking the availability of the Connectors component and a short history of communication between the Core component and the Connectors component. Because the API runs in the Core component, and the Connectors may be on a completely separate server or cluster node, this API may be helpful to check communication between the components.

API Entry Points

`/api/crawlers/status`: Get the status of the Connectors component

Get Connectors Status

```
GET /api/crawlers/status
```

Input

Path Parameters

None.

Query Parameters

None.

Output

Output Content

There are three main areas output from the status request, `clients`, `status` and `messages`.

The `status` section will return "OK" if everything is normal or "WARNING" if the Connectors component is down.

The `clients` section will list the connected clients. If one of these presents an error, then the communication between the components is not operational. The causes for this will be listed in the `messages` section. If `status` in this section is "OK", then communication is normal. For example, if the below is shown as one of the entries under `clients`, then the connection is OK:

```
"http://localhost:8765/connectors/v1/mgr": {
   "status": "OK",
   "messages": [
      {"message": "online",
       "time": "2012-10-29T14:54:17.553Z"
      }],
   "clients": {},
   "type": "local"
}
```

If the Connectors component is down, however, this client would have a status of WARNING and some information in the `messages` section:

```
"http://127.0.0.1:8765/connectors/v1/mgr": {
   "clients": {},
   "messages": [
     {"message": "Communication Error (1001) - The connector failed to complete the
communication with the server",
      "time": "2012-11-18T20:56:59.774Z"
      }],
   "status": "ERROR",
   "type": "local"
}
```

The `messages` section will show the time, date, and a history of messages. The `messages` section in the `clients` section show the current messages, if any, but the main `messages` section shows a history of messages. For example, if the Connectors component was stopped and restarted, the message history would look something like:

```
"messages": [
   {
      "message": "Initiating data source loading and back-compat verification...",
      "time": "2012-11-18T20:41:16.859Z"
   },
   {
      "message": "2 data sources loaded and verified.",
      "time": "2012-11-18T20:41:26.161Z"
   },
   {
      "message": "No connection to remote ConnectorManager! Switching to read-only
cache (noop).:
         Communication Error (1001) - The connector failed to complete the
communication with the server",
      "time": "2012-11-18T20:56:59.774Z"
   },
      "message": "Synchronizing local and remote data sources after re-connect...",
      "time": "2012-11-18T21:02:38.091Z"
   },
   {
      "message": "2 data sources loaded and verified.",
      "time": "2012-11-18T21:02:38.394Z"
   }]
```

Note the timestamps on this message. First communication was normal, then there was an error in communication, then after the Connectors component was restarted, the communication returned to normal.

Examples

Input

```
curl http://localhost:8888/api/crawlers/status
```

Output

```
{
    "clients": {
        "http://127.0.0.1:8765/connectors/v1/mgr": {
            "clients": {},
            "messages": [
                {
                    "message": "online",
                    "time": "2012-11-18T20:44:07.026Z"
                }
            ],
            "status": "OK",
            "type": "local"
        },
        "noop": {
            "clients": {},
            "messages": [
                {
                    "message": "read-only datasource cache (local)",
                    "time": "2012-11-18T20:41:26.162Z"
                },
                {
                    "message": "cached 2 datasources",
                    "time": "2012-11-18T20:41:26.162Z"
                }
            ],
            "status": "OK",
            "type": "No-op ConnectorManager"
        }
    },
    "messages": [
        {
            "message": "Initiating data source loading and back-compat
verification...",
            "time": "2012-11-18T20:41:16.859Z"
        },
        {
            "message": "2 data sources loaded and verified.",
            "time": "2012-11-18T20:41:26.161Z"
        }
    ],
    "status": "OK",
    "type": "REST"
}
```

Related Topics

- Working With LucidWorks Search Components

Example Clients

Example client code demonstrating how to communicate with LucidWorks Search with a variety of programming languages can be found in the `$LWE_HOME/app/examples/` directory of your LucidWorks Search installation.

- Example Perl Clients
- Example Python Clients
- Example .Net Clients

Information for LucidWorks Search in the Cloud Users
Customers using LucidWorks Search hosted on AWS or Azure who are interested in using these clients can download them from our website at LucidWorks Sample Clients.

Example .Net Clients

The `$LWE_HOME/app/examples/csharp` directory contains utilities demonstrating some of the LucidWorks REST API features from C# code. These utilities can be used to assist people in managing their LucidWorks Search installation, or as an example of how to write C# code as part of customer applications that will interact with LucidWorks and Solr.

- Dependencies
- Basic Usage

Dependencies

All of these tools require that the "Json.NET" DLL be installed.

All of these tools assume that the main URL for LucidWorks is "http://localhost:8888". If LucidWorks is running elsewhere, please set the LWE_URL Environment variable appropriately in the shell where you will be using these tools.

All of these tools deal with "collection1" by default. To use a different collection, please set the LWE_COLLECTION Environment variable appropriately in the shell where you will be using these tools.

Basic Usage

All of these tools can be run without any arguments to see "help" info about their usage.

Get Some basic Info about the collection...

```
info.exe show
info.exe show index_num_docs index_size free_disk_space
```

View, Modify Settings...

```
settings.exe show
settings.exe show boost_recent stopword_list
settings.exe update boost_recent=false stopword_list=a stopword_list=an
stopword_list=the
```

(note that creating a list is done by specifying the same setting key multiple times)

Execute Searches (with optional filters)

```
search.exe "gtk gnome"
search.exe "gtk -gnome"
search.exe "+gtk +gnome" "mimeType:text/html"
```

Example Perl Clients

The `$LWE_HOME/app/examples/perl` directory contains utilities demonstrating many of the LucidWorks REST API features from Perl code. These utilities can be used to assist people in managing their LucidWorks Search installation, or as an example of how to write Perl code as part of customer applications that will interact with LucidWorks and Solr.

Dependencies

All of these tools require that the "LWP" and "JSON" Perl modules be installed.

All of these tools assume that the main URL for LucidWorks is "http://localhost:8888" and that the URL for the UI is "http://localhost:8989"

If LucidWorks is running elsewhere, please set the LWE_URL and LWE_UI_URL Environment variables appropriately in the shell where you will be using these tools.

With the exception of "collections.pl" (which deals with multiple collections) all of these tools work with "collection1" by default. To use a different collection, please set the LWE_COLLECTION Environment variable appropriately in the shell where you will be using these tools.

Basic Usage

All of these tools can be run without any arguments to see "help" information about their usage.

View, Create, Modify, or Delete Collections

```
collections.pl show
collections.pl show name=collection1
collections.pl create name=products instance_dir=prod_dir
collections.pl delete name=products
```

Get Basic Information About the Collection

```
info.pl show
info.pl show index_num_docs index_size free_disk_space
```

View or Modify Settings

```
settings.pl show
settings.pl show boost_recent stopword_list
settings.pl update boost_recent=false stopword_list=a stopword_list=an
stopword_list=the
```

(Note that you can create a list by specifying the same setting key multiple times.)

View, Create, Modify, or Delete Data Sources

```
ds.pl show
ds.pl show id=74
ds.pl show name=simple
ds.pl create name=simple type=file crawler=lucid.aperture path=/usr/share/gtk-doc/html
ds.pl create name=docs type=file crawler=lucid.aperture path=/usr/share/gtk-doc/html
crawl_depth=100
ds.pl update id=74 crawl_depth=999
ds.pl update name=simple crawl_depth=999
ds.pl update id=74 name=new_name crawl_depth=999
ds.pl delete id=74
ds.pl delete name=simple
ds.pl delete-all YES YES YES
```

Manually Start or Stop a Data Source Job

```
ds.pl start id=74
ds.pl start name=simple
ds.pl stop id=74
ds.pl stop name=simple
```

Modify the Schedule of an Existing Data Source

```
ds.pl schedule id=74 active=true period=60 start_time=2076-03-06T12:34:56-0800
ds.pl schedule id=74 active=true period=60 start_time=now
ds.pl schedule name=simple active=true period=60 start_time=now
```

View the Status and Indexing History of Existing Data Sources

```
ds.pl status
ds.pl status id=74
ds.pl status name=simple
ds.pl history id=74
ds.pl history name=simple
```

View, Create, Modify, Check, or Delete Alerts

```
alerts.pl show
alerts.pl show username=bob
alerts.pl show id=68
alerts.pl create username=bob query=gnome name=gnome_alert
alerts.pl update id=68 period=5
alerts.pl check id=68
alerts.pl delete id=68
```

View, Create, Modify, or Delete Activities

```
activities.pl show
activities.pl show id=68
activities.pl create type=click active=true period=60
start_time=2076-03-06T12:34:56-0800
activities.pl create type=click active=true period=60 start_time=now
activities.pl update id=68 active=true period=300
activities.pl delete id=68
```

View the Status and History of Existing Activities

```
activities.pl status
activities.pl status id=68
activities.pl history id=68
```

View, Create, Modify, or Delete Fields

```
fields.pl show
fields.pl show name=mimeType
fields.pl create name=category field_type=string facet=true
fields.pl update name=category search_by_default=true
fields.pl delete name=category
```

View, Create, Modify, or Delete Users

```
users.pl show
users.pl show username=admin
users.pl create username=jim authorization=admin password=jimpass
users.pl update username=jim authorization=search
users.pl delete username=jim
```

Modify Roles

```
roles.pl show
roles.pl show name=DEFAULT
roles.pl create name=SECRET users=hank users=sam filters=status:secret
roles.pl update name=DEFAULT filters=status:public
roles.pl append name=SECRET users=jim users=bob groups=executives
roles.pl delete name=SECRET users=hank
roles.pl delete name=OLD
```

Pause or Resume All Background Jobs

```
maintenance.pl pause
maintenance.pl pause force
maintenance.pl resume ds=5 ds_sched=5 ds_sched=7 act_sched=9
```

Execute Searches with Optional Filters

```
search.pl "gtk gnome"
search.pl "gtk -gnome"
search.pl "+gtk +gnome" "mimeType:text/html"
```

Recipes

Indexing Data Sources

1. Start up LucidWorks
2. Create a data source using files on the same server as LucidWorks:
   ```
   ds.pl create name=localdocs type=file crawler=lucid.aperture
   path=/usr/share/gtk-doc/html crawl_depth=100
   ```
3. Schedule the "localdocs" data source to be indexed every 30 minutes starting now:
   ```
   ds.pl schedule name=localdocs active=true period=1800 start_time=now
   ```
4. Create a data source using a remote HTTP server:
   ```
   ds.pl create name=solrwiki type=web crawler=lucid.aperture
   url=http://wiki.apache.org/solr/ crawl_depth=1
   ```
5. Run the "solrwiki" data source once right now:
   ```
   ds.pl start name=solrwiki
   ```
6. Periodically check the status of your data sources to see when the initial indexing is done (look for "crawl_state"):
   ```
   ds.pl status
   ```
7. Execute some searches in your browser:
   http://localhost:8989/collections/collection1/search?search%5Bq%5D=configuration
8. Searches can also be executed via the REST API using search.pl:
   ```
   search.pl configuration
   ```

Indexing and Activating Filters for Certain Users

1. Start LucidWorks
2. Create a new role HTML_ONLY to restrict some users and groups to only searching for HTML documents

```
roles.pl create name=HTML_ONLY filters=mimeType:text/html
```

3. Create a new search user named jim:

```
users.pl create username=jim password=jimpass authorization=search
```

4. Add "jim" to the list of users with the HTML_ONLY role:

```
roles.pl append name=HTML_ONLY users=jim
```

5. Create a data source of a directory containing HTML files as well as other plain text files:

```
ds.pl create name=simple type=file crawler=lucid.aperture
path=/usr/share/gtk-doc/html crawl_depth=100
```

6. Run the data source once right now:

```
ds.pl start name=simple
```

7. periodically check the 'status' of your data source to see when the initial indexing is done (look for "crawl_state"):

```
ds.pl status name=simple
```

8. Use your browser to login as the "jim" (with password "jimpass") and execute a search: http://localhost:8989/collections/collection1/search?search%5Bq%5D=
9. As you execute various searches you should only see HTML documents (note the "Type" Facet in the right hand navigation column)
10. Click the "Sign Out" link in the upper-right corner of search pages and Log in again as the "admin" user: http://localhost:8989/users/sign_out
11. Execute the same searches as before: http://localhost:8989/collections/collection1/search?search%5Bq%5D=
12. As you execute various searches you should now see all documents (note the "Type" Facet in the right hand navigation column)

Indexing and Activating Click Boosting

1. Start LucidWorks
2. Update your settings to enable click tracking:

```
settings.pl update click_enabled=true
```

3. Create a data source:

```
ds.pl create name=local_click_ds type=file crawler=lucid.aperture
path=/usr/share/gtk-doc/html crawl_depth=100
```

4. Schedule the data source to be indexed every 30 minutes starting now:

```
ds.pl schedule name=local_click_ds active=true period=1800 start_time=now
```

5. Schedule the click processing activity to run every 10 minutes:

```
activities.pl create type=click active=true period=600 start_time=now
```

6. periodically check the 'status' of your data source to see when the initial indexing is done (look for "crawl_state"):

```
ds.pl status name=local_click_ds
```

7. Execute a search in your browser: http://localhost:8989/collections/collection1/search?search%5Bq%5D=gnome
8. As you execute searches and click on results, you should see the documents you click on filter up to the top of those searches as the click processing activity runs every 10 minutes.

Pause and Resume All Background Jobs for Maintenance

1. Start LucidWorks
2. Update your settings to enable click tracking:

```
settings.pl update click_enabled=true
```

3. Create a data source:

```
ds.pl create name=local_click_ds type=file crawler=lucid.aperture
path=/usr/share/gtk-doc/html crawl_depth=100
```

4. Schedule the data source to be indexed every 30 minutes starting now:

```
ds.pl schedule name=local_click_ds active=true period=1800 start_time=now
```

5. Schedule the click processing activity to run every 10 minutes:

```
activities.pl create type=click active=true period=600 start_time=now
```

6. Pause all active data source schedules and activities, blocking until any currently running data sources and activities are finished:

   ```
   maintenance.pl pause
   ```

   This command should output something like the following:

   ```
   $ maintenance.pl pause
   De-Activating activity #9:
   http://localhost:8888/api/collections/collection1/activities/9
   De-Activating schedule of ds#5:
   http://localhost:8888/api/collections/collection1/datasources/5/schedule
   Waiting for any currently running Activities to finish...
   ...Done!
   Waiting for any currently running DataSources to finish...
   ...Done!
   Run this command to resume everything that was de-activated...
   maintenance.pl resume activity=9 ds=5
   ```

7. Perform whatever maintenance is needed.
8. When you are ready, run the command mentioned in the output of the "Pause" step to resume scheduled data source and activity processing:

   ```
   maintenance.pl resume activity=9 ds=5
   ```

9. Your data source and click activity will now continue to run on the previously specified schedules.

Example Python Clients

The `$LWE_HOME/app/examples/python` directory contains utilities demonstrating many of the LucidWorks REST API features from Python code. These utilities can be used to assist people in managing their LucidWorks Search installation, or as an example of how to write Python code as part of customer applications that will interact with LucidWorks and Solr.

- Recipes
    - Indexing Data Sources
    - Indexing and Activating Filters for Certain Users
    - Indexing and Activating Click Boosting

Dependencies

All of these tools require that the "httplib2" library be available.

All of these tools assume that the main URL for LucidWorks is "http://localhost:8888" and that the URL for the UI is "http://localhost:8989"

If LucidWorks is running elsewhere, please set the LWE_URL and LWE_UI_URL Environment variables appropriately in the shell where you will be using these tools.

All of these tools work with with "collection1" by default. To use a different collection, please set the LWE_COLLECTION Environment variable
appropriately in the shell where you will be using these tools.

Basic Usage

All of these tools can be run without any arguments to see "help" information about their usage.

Get Basic Information About the Collection

```
info.py show
info.py show index_num_docs index_size free_disk_space
```

View or Modify Settings

```
settings.py show
settings.py show boost_recent stopword_list
settings.py update boost_recent=false stopword_list=a stopword_list=an
stopword_list=the
```

(Note that you can create a list by specifying the same setting key multiple times.)

View, Create, Modify, or Delete Data Sources

```
ds.py show
ds.py show id=74
ds.py show name=simple
ds.py create name=simple type=file crawler=lucid.aperture path=/usr/share/gtk-doc/html
ds.py create name=docs type=file crawler=lucid.aperture path=/usr/share/gtk-doc/html
crawl_depth=100
ds.py update id=74 crawl_depth=999
ds.py update name=simple crawl_depth=999
ds.py update id=74 name=new_name crawl_depth=999
ds.py delete id=74
ds.py delete name=simple
```

Modify the Schedule of an Existing Data Source

```
ds.py schedule id=74 active=true period=60 start_time=2076-03-06T12:34:56-0800
ds.py schedule id=74 active=true period=60 start_time=now
ds.py schedule name=simple active=true period=60 start_time=now
```

View the Status and Indexing History of an Existing Data Source

```
ds.py status
ds.py status id=74
ds.py status name=simple
ds.py history id=74
ds.py history name=simple
```

View, Create, Modify, or Delete Activities

```
activities.py show
activities.py show id=68
activities.py create type=click active=true period=60
start_time=2076-03-06T12:34:56-0800
activities.py create type=click active=true period=60 start_time=now
activities.py update id=68 period=300
activities.py delete id=68
```

View the Status and History of Existing Activities

```
activities.py status
activities.py status id=68
activities.py history id=68
```

View, Create, Modify, or Delete Fields

```
fields.py show
fields.py show name=mimeType
fields.py create name=category field_type=string facet=true
fields.py update name=category search_by_default=true
fields.py delete name=category
```

View, Create, Modify, or Delete Users

```
users.py show
users.py show username=admin
users.py create username=jim authorization=admin password=jimpass
users.py update username=jim authorization=search
users.py delete username=jim
```

Modify Roles

```
roles.py show
roles.py show name=DEFAULT
roles.py create name=SECRET users=hank users=sam filters=status:secret
roles.py update name=DEFAULT filters=status:public
roles.py append name=SECRET users=jim users=bob groups=executives
roles.py delete name=SECRET users=hank
roles.py delete name=OLD
```

Execute Searches with Optional Filters

```
search.py "gtk gnome"
search.py "gtk -gnome"
search.py "+gtk +gnome" "mimeType:text/html"
```

Recipes

Indexing Data Sources

1. Start LucidWorks
2. Create a data source using files on the same server as LucidWorks:
   ```
   ds.py create name=localdocs type=file crawler=lucid.aperture
   path=/usr/share/gtk-doc/html crawl_depth=100
   ```
3. Schedule the "localdocs" data source to be indexed every 30 minutes starting now:
   ```
   ds.py schedule name=localdocs active=true period=1800 start_time=now
   ```

4. Create a data source using a remote HTTP server:

```
ds.py create name=solrwiki type=web crawler=lucid.aperture
url=http://wiki.apache.org/solr/ crawl_depth=1
```

5. Schedule the 'solrwiki' data source to be indexed once right now:

```
ds.py schedule name=solrwiki active=true period=0 start_time=now
```

6. Periodically check the 'status' of your data sources to see when the initial indexing is done (look for "crawl_state"):

```
ds.py status
```

7. Execute some searches in your browser: http://localhost:8989/collections/collection1/search?search%5Bq%5D=configuration
8. Searches can also be executed via the REST API using search.py:

```
search.py configuration
```

Indexing and Activating Filters for Certain Users

1. Start LucidWorks
2. Create a new role HTML_ONLY to restrict some users and groups to only searching for HTML documents

```
roles.py create name=HTML_ONLY filters=mimeType:text/html
```

3. Create a new search user named jim:

```
users.py create username=jim password=jimpass authorization=search
```

4. Add "jim" to the list of users with the HTML_ONLY role:

```
roles.py append name=HTML_ONLY users=jim
```

5. Create a data source of a directory containing HTML files as well as other plain text files:

```
ds.py create name=simple type=file crawler=lucid.aperture
path=/usr/share/gtk-doc/html crawl_depth=100
```

6. Run the data source once right now:

```
ds.py start name=simple
```

7. periodically check the 'status' of your data source to see when the initial indexing is done (look for "crawl_state"):

```
ds.py status name=simple
```

8. Use your browser to login as the "jim" (with password "jimpass") and execute a search: http://localhost:8989/collections/collection1/search?search%5Bq%5D=
9. As you execute various searches you should only see HTML documents (note the "Type" Facet in the right hand navigation column)
10. Click the "Sign Out" link in the upper-right corner of search pages and Log in again as the "admin" user: http://localhost:8989/users/sign_out
11. Execute the same searches as before: http://localhost:8989/collections/collection1/search?search%5Bq%5D=
12. As you execute various searches you should now see all documents (note the "Type" Facet in the right hand navigation column)

Indexing and Activating Click Boosting

1. Start LucidWorks
2. Update your settings to enabled click tracking:

```
settings.py update click_enabled=true
```

3. Create a data source:

```
ds.py create name=local_click_ds type=file crawler=lucid.aperture
path=/usr/share/gtk-doc/html crawl_depth=100
```

4. Schedule the data source to be indexed every 30 minutes starting now:

```
ds.py schedule name=local_click_ds active=true period=1800 start_time=now
```

5. Schedule the click processing activity to run every 10 minutes:

```
activities.py create type=click active=true period=600 start_time=now
```

6. Periodically check the status of your data source to see when the initial indexing is done (look for "crawl_state"):

```
ds.py status name=local_click_ds
```

7. Execute a search in your browser: http://localhost:8989/collections/collection1/search?search%5Bq%5D=gnome
8. As you execute searches and click on results, you should see the documents you click on filter up to the top of those searches as the click processing activity runs every 10 minutes.

Advanced Operations Using the REST API

LucidWorks provides the ability to programmatically control administrative functions using the REST API.

To use the REST API, you send a JSON object via an HTTP request. For example, you can use the REST API to dynamically create a field:

```
curl -d '{
   "name": "new_field",
   "term_vectors": true,
   "default_value": "lucid rocks",
   "use_for_deduplication": true,
   "multi_valued": true,
   "stored": true,
   "indexed": true,
   "search_by_default": false,
   "facet": true,
   "index_for_spellcheck": true,
   "synonym_expansion": true,
   "short_field_boost": "moderate",
   "field_type": "text_en"
 }' -H 'Content-type: application/json'
'http://localhost:8888/api/collections/collection1/fields'
```

The request returns a JSON representation of the created object:

```
{
   "default_boost":1.0,
   "field_type":"text_en",
   "facet":true,
   "indexed":true,
   "short_field_boost":"moderate",
   "term_vectors":true,
   "include_in_results":false,
   "stored":true,
   "omit_tf":false,
   "highlight":false,
   "editable":true,
   "search_by_default":false,
   "user_field":true,
   "multi_valued":true,
   "default_value":"lucid rocks",
   "use_for_deduplication":true,
   "name":"new_field",
   "synonym_expansion":true,
   "index_for_spellcheck":true,
   "index_for_autocomplete":false,
   "query_time_stopword_handling":false,
   "copy_fields":["text_medium","text_all","spell"],
   "use_in_find_similar":false}
```

As with all operations against LucidWorks, you have the option to use any HTTP client to perform these operations, as long as they use the proper methods, as documented under REST API.

Perhaps the most common usage of the REST API is to control and monitor data sources and indexing. Consider this example, from Getting Started Indexing. First, you create the data source:

```
curl -H 'Content-type: application/json' -d '{
 "crawler":"lucid.aperture",
 "type":"web",
 "url":"http://www.lucidimagination.com/",
 "crawl_depth":1,
 "name":"Lucid Website"
   }'
http://localhost:8888/api/collections/collection1/datasources
```

The response includes the id value (it is quite long, so it's been truncated here):

```
{
    "add_failed_docs": true,
    "auth": [],
    "bounds": "none",
    "caching": false,
    "category": "Web",
    "collection": "collection1",
    "commit_on_finish": true,
    "commit_within": 900000,
    "crawl_depth": -1,
    "crawler": "lucid.aperture",
    "exclude_paths": [ ],
    "fail_unsupported_file_types": true,
    "id": 3,
    "ignore_robots": false,
    "include_paths": [ ],
    "indexing": true,
    "log_extra_detail": false,
    "mapping": {
  ...
    },
    "max_bytes": 10485760,
    "max_docs": -1,
    "name": "Lucid Website",
    "parsing": true,
    "proxy_host": "",
    "proxy_password": "",
    "proxy_port": -1,
    "proxy_username": "",
    "type": "web",
    "url": "http://www.lucidimagination.com/",
    "verify_access": true,
    "warn_unknown_mime_types": true
}
```

The next step is to tell LucidWorks to start indexing the documents by submitting a job request:

```
curl -X PUT http://localhost:8888/api/collections/collection1/datasources/3/job
```

This request does not return anything, but it does start the index running. To check the status, we send a `GET` request:

```
curl http://localhost:8888/api/collections/collection1/datasources/3/status
```

This request tells us that the data source is still being indexed:

```
{
    "batch_job": false,
    "crawl_started": "2012-02-06T18:40:12+0000",
    "crawl_state": "RUNNING",
    "crawl_stopped": null,
    "id": 3,
    "job_id": "6",
    "num_access_denied": 0,
    "num_deleted": 0,
    "num_failed": 2,
    "num_filter_denied": 0,
    "num_new": 227,
    "num_not_found": 0,
    "num_robots_denied": 0,
    "num_total": 229,
    "num_unchanged": 0,
    "num_updated": 0
}
```

Another common use for the REST API is to manage users and alerts. In most cases, you will use an LDAP server to manage users, but you also have the option to manage them internally using the REST API. For example, to create a user, you would send a `POST` request:

```
curl -H 'Content-type: application/json' -d '
{
  "username": "smiller",
  "email": "me@here.com",
  "authorization": "search",
  "password": "123456"
}' 'http://localhost:8989/api/users'
```

You can also create an Enterprise Alert, which notifies the user when there are new results for a search. For example, you can create an alert that sends the new user, `smiller`, notification when there is new data for the search "robot":

```
curl -H 'Content-type: application/json' -d '{
   "name":"Robot search",
   "collection":"collection1",
   "username":"smiller",
   "query":"robots",
   "period":86400,
   "email":"test@test.com"
}' http://localhost:8989/api/alerts
```

⚠ The Users API and Enterprise Alerts do not run on the Solr port; they're handled by the LWE UI component, installed by default on port 8989.

The request returns a JSON representation of the object, which includes the ID:

```
{
   "id":1,
   "name":"Robot search",
   "collection":"collection1",
   "username":"smiller",
   "query":"robots",
   "checked_at":"2012-06-07T14:22:54Z",
   "period":86400,
   "email":"me@here.com"
}
```

You can then use that ID to check the status of the alert:

```
curl -X PUT http://localhost:8989/api/alerts/1/check
```

In this case, we have not yet seen any new results:

```
{
   "checked_at": "2012-06-07T14:22:54Z",
   "collection": "collection1",
   "email": "me@here.com",
   "id": 1,
   "name": "Robot search",
   "period": 86400,
   "query": "robots",
   "username": "smiller"
}
```

Lucid Query Parser

The Lucid query parser is designed as a replacement for the existing Apache Lucene and Solr query parsers to produce good results with queries in any format, extracting the maximum possible information from the user entry to produce the best matches. The user (or system administrator) does not need to indicate what type of query it is: query interpretation is "modeless". The system will never produce an error message in response to a query and will always make the best interpretation possible. The system will match documents with alternative word forms (for example, singulars/plurals; this is on by default), expand synonyms, spell-check queries and handle queries in various languages (English by default) as configured by the administrator.

Features

The basic features of the Lucid query parser include all those of the traditional Apache Lucene and Apache Solr DisMax query parsers and is designed to work well for users accustomed to the search syntax used by popular Web search engines.

The Lucid query parser also includes a number of features not found in some or all of these other query parsers, including:

- Relational operators (comparative operators) - '<', '<=', '==', '!=', '>=', '>'
- More flexible range searches
- Implicit AND operator when no explicit operators are present
- The ALL pseudo-field to search across all fields
- Automatic bigram and trigram relevancy boosting
- Support for natural language queries
- Support for a wide variety of date formats
- Support for advanced proximity operators - NEAR, BEFORE, AFTER
- Support for multi-word synonyms
- Enhanced support for hyphenated terms
- Case-insensitive field names
- Sticky field names
- Good results even in the presence of query syntax errors with automatic error recovery
- Intelligent out-of-the-box settings for best results
- Wide range of configuration options for special needs

Simply put, the Lucid query parser is capable of handling existing Lucene and Solr queries and many Web search engine queries.

Building Search Queries

This section reviews some of the most commonly used types of search queries and gives examples of how LucidWorks processes them. More information can be found in the section on Building Advanced Queries.

Basic Usage

The Lucid query parser uses a number of built-in but configurable heuristics to automatically detect and process queries in a variety of formats.

Basic keyword queries are made up of words and quoted phrases, such as those used in Web search engines. As in those search engines, the default interpretation rule for such queries is that all significant words in the query must be present for a document to match. Examples:

- `quantum physics`
- `address of "White House"`

Simple Boolean queries are made up of words or phrases preceded by a '+' to indicate that a word or phrase must be present for a document to match, or by a '-' to mean they cannot be present. If a '+' or '-' or relational operator is present, other words or phrases in the query are interpreted as 'nice to have' but are not required or excluded for documents to match. For example,

- `+pet cat dog fish rabbit -snakes`

Full Boolean queries use any legal combination of the Boolean operators AND, OR, NOT and an unlimited number of matching parentheses. The AND and OR operators may be any case, but the NOT operator must always be upper case. Configuration settings can be used to alter these defaults. For example,

- `(dog AND puppy) OR (cat AND kitten)`
- `lincoln and washington NOT (bridge or tunnel)`

Extended Boolean queries combine simple Boolean queries (a term list or list of terms and quoted phrases, optionally using the '+' and '-' operators) with the Boolean AND, OR, and NOT operators. For example,

- `Abraham +Lincoln -tunnel or George +Washington -bridge and (early history or influences)`

Natural language queries, are made up of either sentence fragments or natural language questions that begin with one of the question words 'Who', 'What', 'When', 'Where', 'Why', or 'How' and end with a question mark. For example,

- `the return of the pink panther`
- `what is aspirin?`
- `How does a rocket work?`

Queries using Lucene syntax (see below) include field delimiters, Proximity Operators, wildcards, and so on. This syntax may be used with arbitrarily complex Boolean expressions. For example,

- `title:psycholog*`
- `[1968 TO 1972]`

- `"software development"~10`

More like this queries are those in which the users types or pastes in a long section of text as a model to match. The Lucid query parser uses a statistical approach for matching such queries.

For queries in any other format, the Lucid query parser will do a best effort interpretation.

Although these features work together automatically out of the box, a wide range of configuration options are available and described in the following sections.

Users should use quotation marks around any series of words they intend to be interpreted as a literal phrase in order to find documents that must have that exact phrase. However, in most cases users are better served by omitting the quotation marks, since the query parser ranks higher those documents that have consecutive words near each other while also bringing back potentially relevant documents that have those words but not necessarily as a literal phrase.

By default, the Lucid query parser also pays attention to every word in the query and tries to use each word appropriately. Common words such as `a`, `the`, `to`, etc. (often called "stop words"), are not ignored but are treated specially. For example, a query consisting of only stop words will, as the default, require all of them to be present. On the other hand, a query such as `what is aspirin`, will only require the word `aspirin`, but make use of all the words in the query to rank the best documents highest when documents are returned in order of relevancy.

Error Handling

The Lucid query parser is designed to be able to handle and give good results for even the most malformed queries. No exceptions will be thrown in any event. Common mistakes include mismatched parentheses, brackets, or braces, unterminated strings, missing operand for a Boolean operator, or any other operator, unknown field names, and extraneous punctuation.

In the worst case, the offending character, operator, or term will be discarded.

Understanding Terms

The basic unit of a query is a *term*. In its simplest form, a term is simply a *word* or a *number*. A term may also include embedded punctuation such as hyphens or slashes, and may in fact be more than one word separated by such embedded punctuation. A term may also include *wildcard* ('?' and '*').

This basic form of term is referred to as a *single term*. For example, the following are all single terms:

- `cat`
- `CD-ROM`
- `123`
- `1,000`
- `-456`
- `+7.89`

- `$1,000.00`
- `cat*`
- `?at`
- `at?c*he`

Numbers optionally may have a leading + or `-` sign that is considered part of the number. If you wish to place the 'must' or 'must not' operators in front of an unsigned number, add an extra '+' between the operator and the unsigned number.

There is a second form of term called a *phrase*, which is a sequence of terms enclosed in double quotation marks. The intention is that the terms (words) are expected to occur in that order and without intervening terms. An alternative purpose is simply to indicate that a Lucid keyword operator should be interpreted as a natural language word rather than as an operator. For example:

- `"In the beginning" "George Washington" "AND" "myocardial infarction"`

A third form of term is the *range query*, which is a pair or terms which will match all terms that lexically fall between those two terms. For example, `[cat TO dog]` matches all terms between cat and dog in lexical order (i.e., alphabetically, in this case).

A fourth form of term is a parenthesized sub-query which may be a complex Boolean query. For example:

- `cat (bat OR fish AND giraffe) zebra`

A sequence of terms (of any form – single, phrase, range, or sub-query) is referred to as a *term list*. A term list might be used to represent one or more compound terms, or simply a list of keywords and phrases, or a combination of the two. A term list is the most common form of query. In this basic form, a term list has no operators.

Note that for non-text fields, a term may not actually be a word or number, but may have a special syntax specific to that field such as a part number or telephone number.

In order to offer advanced search functions, each term can optionally be preceded or followed by various *term modifiers*. An example of a modifier before a term is a *field name*. An example of a modifier after a term is a *boost factor*.

A subset of operators (*term operators*, + and `-`) may also be included within a term list.

Case Insensitivity

As a general rule, query text is *case insensitive*, meaning that you may use any combination of upper and lower case letters regardless of the case used in documents in the search collection. This also includes field names and keyword options. Mixed case may be used for technical correctness (e.g., proper names) or for readability or emphasis, but will in no way affect the query results.

One exception is in *string* or *keyword* fields, as distinct from *full-text* fields, where the administrator may have chosen to maintain precise case for precision or some other reason.

Another exception is that the administrator may decide to remove the term analysis filter which is responsible for making sure that terms are converted to lower case when they are indexed as well as query time.

But in general you should feel free to enter queries as feels most natural and feel free to enter the entire query in lower case if that is what feels most natural to you.

The following queries will be interpreted identically:

- `president george washington "cherry tree" datemodified:1994`
- `President George Washington "Cherry Tree" dateModified:1994`
- `president GEORGE Washington "CHERRY TREE" DateModified:1994`
- `PRESIDENT george WAshington "CHERRY tree" DATEMODIFIED:1994`

Simple Boolean Queries

A simple Boolean query consists of a list of terms, single keywords or quoted phrases, some of which may be preceded with the '+' and '-' Boolean operators. Terms preceded by the '+' operator are *required*, meaning that only documents containing those terms will be selected by the query. Terms preceded by the '-' operator are *prohibited*, meaning that only documents that do not contain any of the prohibited terms will be selected by the query. All other terms are considered *optional*, meaning that none of those terms are required for a document to be selected by the query, but documents containing those terms will be ranked higher based on how many of the optional terms they contain. For example,

- `president george +washington -bridge`

Which selects documents which contain "Washington" but not "bridge". Selected documents do not have to contain "president" or "george", but they will be ranked higher if they do.

Natural Language Queries

A *natural language* query is one in which the user does not use any special syntax, but instead enters their search in the form of a statement or question, or as they would normally speak to another person. In general, the best first stab at any query is to write the query as a natural language question, such as:

- `what is aspirin?`

Or, include all of the connective words to fully express the topic so that they can participate in relevancy boosting. For example, rather than asking a user to conform their query to the query parser with:

- `title: return "pink panther"`

the user can simply enter the title as it would normally appear:

- `the return of the pink panther`

The Lucid query parser will automatically reformulate the query so that documents containing the exact wording will rank high, but documents containing subsets of the query will rank reasonably high as well.

A special feature supporting natural language queries is that a trailing question mark ('?', which normally would be treated as a wildcard character) will automatically be stripped from the query if there are at least three terms in the query and the initial term is a question word, "who", "what", "when", "where", "why", or "how". If for some reason you do wish to enter a query that ends with a wildcard question mark but starts with a question word, simply follow it with a space and a period or other punctuation character. For example,

- `what is aspirin? .` (Treated as a wildcard)

The exact minimum word count is configurable with the `minStripQMark` configuration setting.

Phrase Query

In natural language, a phrase is simply a sequence of words or terms. Query languages usually require that the phrase be enclosed within double quotation marks. Raw natural language text is frequently as good as explicitly quoted phrases. However, in some cases an explicit phrase may be best for a particular query.

A quoted phrase is simply any query text (or an n-gram) enclosed within double quotation marks. For example:

- `"John Doe"`
- `"John Q. Doe"`
- `"Java software development"`
- `"The quick brown fox jumped over the lazy dog's back."`

The text may consist of words as well as punctuation. Although operators and other special characters may also be present, they will not have their usual meaning and will be considered as words or punctuation. In particular, wildcards, parentheses, "+' and "=", and boost factors have no special meaning inside of a quoted phrase. For example, the following queries are equivalent:

- `"--The +cat (in the hat?), ran^2.0 * -away."`
- `"The cat in the hat ran 2.0 away"`

An empty phrase query (that is, "") will be ignored.

A phrase query containing a single term will be treated as a single term query, but any wildcards or operators within the term will be ignored and stop words will be processed as non-stop words.

In addition to simple Phrase Queries, the Lucid Query Parser also supports Advanced Proximity Queries.

More Like This

A *more like this* query is a passage of natural language text that has more than a threshold number of terms (the default is 12, but can be configured by the administrator with the `likeMin` configuration setting in `solrconfig.xml`). All terms will be implicitly ORed. This may result in a large number of results, but automatic bigram and trigram relevancy boosting will tend to rank results that more closely match the query text. Two applications of this feature are to detect plagiarism and derivative works or subtle variations. Exact matches will rank highest, but close matches will also rank high. For example:

```
When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.
```

All of that text would typically be pasted into the query box as one line of text.

Be sure not to enclose the passage in double quotation marks, otherwise the query will be treated as a precise phrase query. There is no harm in using the quoted phrase, other than that an exact match must be made in that case.

This feature may be suppressed by simply enclosing the entire query within parentheses:

```
(When in the Course of human events, it becomes necessary for one people to dissolve ... )
```

Boolean Operators

Although the majority of queries can be constructed without resorting to explicit Boolean operators, enterprise users sometimes do need the extra power to form complex queries to narrow searches. The basic Boolean operators are AND, OR, and unary and binary NOT. Evaluation is left to right, but parenthesis can be used to control the evaluation order. For example,

- `Cat OR dog AND pet NOT zebra`
- `cat OR NOT dog`
- `cat AND NOT dog`
- `cat NOT dog`
- `(cat OR dog) AND (table OR chair)`
- `hit AND run`
- `hot OR not`
- `NOT cat AND dog`

- `cat AND (NOT dog)`

Binary 'NOT' is equivalent to 'AND NOT' unless preceded by 'OR'.

A Boolean operator is used to combine the results of two term lists or the results of another boolean operator. For example,

- `Abraham Lincoln OR George Washington`
- `second-hand furniture AND (table OR chair)`

Parentheses are not required around term lists (any sequence of terms without Boolean operators, also known as an *extended Boolean* query). For example, the following are equivalent to the preceding examples:

- `(Abraham Lincoln) OR (George Washington)`
- `(second-hand furniture) AND (table OR chair)`

Although Boolean operators are normally written in all upper case, lower case '`and`' and '`or`' are also permitted by default. Sometimes that may cause a query to be misinterpreted, but usually with little or no harm. Lower case 'not' is not permitted since it would cause very wrong results if it happened to occur as a query term. For example:

- `Abraham Lincoln and George Washington`
- `second-hand furniture and (table or chair)`
- `hit and run`
- `hot or not`

The `upOp` configuration setting can be enabled to require all Boolean operators to be upper case-only. The `notOp` configuration setting can be disabled to allow lower case '`not`' as a Boolean operator. See Query Parser Customization for more details on how to change these settings.

In addition to the Boolean keywords, non-keyword operator equivalents are available as substitutions for the keyword Boolean operators. A double ampersand ('&&') means 'AND', a double vertical bar ('||') means 'OR' and a single exclamation point ('!') means 'NOT'. So, the above examples can also be written as:

- `Cat || dog && pet ! zebra`
- `cat || ! dog`
- `cat && ! dog`
- `cat ! dog`
- `(cat || dog) && (table || chair)`
- `Abraham Lincoln || George Washington`
- `second-hand furniture && (table || chair)`
- `(Abraham Lincoln) || (George Washington)`
- `(second-hand furniture) && (table OR chair)`
- `hit && run`
- `hot || not`

- `! cat && dog`
- `cat && (! dog)`

Sticky field names or keyword options remain in effect only until either a new sticky field name (or keyword option) or a right parenthesis at the current parenthesis nesting level. For example,

User Entry	Equivalent
`title:cat nap OR dog bark`	`title:(cat nap) OR title:(dog bark)`
`title:cat nap OR body: dog bark`	`title:(cat nap) OR body:(dog bark)`
`(body:cat AND ((title:dog) OR fish)) AND bat`	`(body:cat AND ((title:dog) OR body:fish)) AND default:bat`

Left to Right Boolean Evaluation Order

Unlike Lucene and Solr, the Lucid query parser assures that Boolean queries without parentheses will be evaluated from left to right. For example, the following are equivalent:

- `cat OR dog OR fox AND pet`
- `(cat OR dog OR fox) AND pet`
- `((cat OR dog) OR fox) AND pet`

Implicit AND versus Implicit OR

If none of the terms of a term list have explicit operators (+ or -, the individual terms will be implicitly ANDed into the query. Lucene and Solr default to implicit ORing of terms, but implicit ANDing tends to produce better results in most applications for most users. For example:

User Input	Query Interpreted as
`heart attack`	`heart AND attack`
`George Washington`	`George AND Washington`
`Lincoln's Gettysburg Address`	`Lincoln's AND Gettysburg AND Address`

But if one or more of the terms in a term list has an explicit term operator (+ or - or relational operator) the rest of the terms will be treated as "nice to have." For example,

- `cat +dog -fox`

Selects documents which must contain "dog" and must not contain "fox". Documents will rank higher if "cat" is present, but it is not required.

- `cat dog turtle -zebra`

Selects all documents that do not contain "zebra". Documents which contain any subset of "cat", "dog", and "turtle" will be ranked higher.

The `defOp` configuration setting in `solrconfig.xml` can be used to disable the implicit AND feature, but it is enabled by default.

Strict vs. Loose Extended Boolean Queries

A *strict query* or Boolean query is one in which explicit Boolean operators are used between all terms. A loose query, also known as an extended Boolean query, uses a combination of explicit Boolean operators and term lists in which the operators are implicit. Put simply, extended Boolean queries allow free-form term lists as operands for the Boolean operators, while strict Boolean queries permit only a single term or quoted phrase (or parenthesized sub-query.) Loose, extended Boolean queries provide every bit of the power of a strict Boolean query, but are more convenient to write and can be easier to read. In fact, queries written in more of a natural language format with fewer explicit Boolean operators facilitate relevancy boosting of adjacent terms.

Examples of strict Boolean queries	**The equivalent loose, extended Boolean queries**
cat AND dog	cat dog
(cat OR dog) AND (food OR health)	(cat OR dog) AND (food OR health)
cat OR dog NOT pets	cat dog -pets
(George AND Washington) OR (Abraham AND Lincoln)	George Washington OR Abraham Lincoln
"George Washington" OR ("Abraham" AND "Lincoln")	"George Washington" OR "Abraham Lincoln"

Hyphenated Terms

Hyphenated terms, such as `plug-in` or `CD-ROM`, are indexed without their hyphens, both as a sequence of sub-words and as a single, combined term which is the concatenation of the sub-words. That combined term is stored at the position of the final sub-word. Users authoring documents are not always consistent on whether they use the hyphens or not, but the goal of the Lucid query parser is to be able to match either given a query of either. To do this as well as possible, the Lucid query parser will expand any hyphenated term into a Boolean OR of the sub-words as a phrase and the combined term.

Simple Hyphenated Terms

A query of `plug-in` will automatically be interpreted as `("plug in" OR plugin)`. If we have these mini-documents:

- Doc #1: This is a plugin.
- Doc #2: This is the plug-in.
- Doc #3: Where is my plug in?

The query will match all three documents.

A query of `plugin` will only match the first two documents, but that is a limitation of this heuristic feature. The query results are better than without this feature even if they are still not ideal.

Hyphenated Terms within Quoted Phrases

Quoted phrases may contain any number of hyphenated terms, in which case the Lucene "span query" feature is used for the entire phrase as well as the individual hyphenated terms which are expanded as above.

A query of:

- `"buy a cd-rom with plug-in software"`

would match any of the following mini-documents:

- Doc #1: I want to buy a cdrom with plugin software
- Doc #2: I want to buy a cdrom with plug-in software
- Doc #3: I want to buy a cd-rom with plugin software
- Doc #4: I want to buy a cd-rom with plug-in software

In terms of the new proximity operators, this query is equivalent to:

- `buy a before:0 cd-rom before:0 with before:0 plug-in software`

which is equivalent to:

- `buy a before:0 ("cd rom" or cdrom) before:0 with before:0 ("plug in" or plugin) before:0 software`

Multiple Hyphens in Terms

Some hyphenated terms have more than two sub-words. For example:

- `on-the-run and never-to-be-forgotten`

will be interpreted as:

- `("on the run" OR ontherun) and ("never to be forgotten" OR nevertobeforgotten)`

Multiple hyphens occur in various special formats, such as phone numbers. For example:

- `646-414-1593 1-800-555-1212`

which will be interpreted as:

- `("646 414 1593" OR 6464141593) AND ("1 800 555 1212" OR 18005551212)`

Social Security numbers and ISBNs also have multiple hyphens. For example,

- `101-23-1234 and 978-3-16-148410-0`

will be interpreted as:

- `("101 23 1234" OR 101231234) and ("978 3 16 148410 0" OR 9783161484100)`

Part numbers and various ID formats also tend to contain more than one hyphen. These would be treated similarly to the examples above.

Punctuation and Special Characters

In general, any punctuation or special character that is not a query operator is treated as if it were white space. This includes commas, periods, semi-colons, slashes, etc. Punctuation or special characters before and after either a single term or the terms within a phrase will be ignored. Punctuation is sometimes referred to as a *delimiter*. For example,

User Input	Query Interpreted as
`/this/`	`this` (note: still treated as a stop word)
`cat, dog; fox.`	`cat dog fox`
`Yahoo!`	`Yahoo`
`C++`	`C`
`B-`	`B`

However, punctuation embedded within a single term or the terms of a phrase will be treated as if it were a hyphen and the term is treated as if it were a phrase with white space in place of the punctuation. For example,

User Input	Query Interpreted as
`x,y,z`	`"x y z"`
`Jan/Feb/Mar`	`"Jan Feb Mar"`
`;Jan/Feb/Mar.`	`"Jan Feb Mar"`
`Jan&Feb`	`"Jan Feb"`
`"Reports for Jan&Feb"`	`"Reports for Jan Feb"`
`C++/C#/Java`	`"C C Java"`
`AT&T`	`"AT T"`
`U.S.`	`"U S"`

Dollar signs, commas, and decimal points are treated similarly. For example,

User Input	Query Interpreted as

`1,000`	`"1 000"`
`$1,275.34`	`"1 275 34"`

Web URLs are not treated specially, other than to allow the colon rather than treating it as a field name, so the URL special characters are removed using the same punctuation removal rules as any other term. For example,

User Input	Query Interpreted as
http://www.cnn.com/	`"http www cnn com"`
http://people.apache.org/list_A.html	`"http people apache org list A html"`

Similarly, email addresses have no special treatment, other than to treat all special characters, including the "@" and dots as delimiters within a phrase. For example,

User Input	Query Interpreted as
`joe@cnn.com`	`"joe cnn com"`
`joseph.smith@whitehouse.gov`	`"joseph smith whitehouse gov"`

Alphanumeric Terms

Alphanumeric single terms and terms within phrases are split into separate terms as a phrase. For example:

User Input	Query Interpreted as
`A20`	`"A 20"`
`B4X3`	`"B 4 X 3"`
`Alpha7`	`"Alpha 7"`
`"Nikon Coolpix P90"`	`"Nikon Coolpix P 90"`
`24x Zoom Z980`	`"24 x" Zoom "Z 980"`

Wildcard Queries

Any term in a query, except for quoted phrases, may contain one or more wildcard characters. Wildcard characters indicate that the term is actually a pattern that may match any number of terms. There are two forms of wildcard character: asterisk ("*") which will match zero or more arbitrary characters and question mark ("?") which will match exactly one arbitrary character.

Wildcards Within or At End of Terms

A term consisting only of one or more asterisks will match all terms of the field in which it is used. For example, `title:*`.

The most common use of asterisk is as the last character of a query term to match all terms that begin with the rest of the query term as a prefix. For example, `paint*`.

One traditional use of asterisk is to force plurals to match. This use is usually unnecessary because LucidWorks uses a stemming filter to automatically match both singular and plural forms. However, this technique may still be useful if the administrator chooses to disable the stemming filter or for fields that may not have a stemming filter. For example, `Sneaker*` will match both "sneaker" and "sneakers".

A question mark can be used where there might be variations for a single character. For example:

User input	Matches
`?at`	"cat", Bat", "fat", "kat", and so on
`c?t`	"cat", "cot", "cut"
`ca?`	"cab", "can", "cat", and so on

Any combination of asterisks and question mark wildcards can be used in a single term, but care is needed to avoid unexpected results.

Note that wildcards are not supported within quoted phrases. They will be treated as if they were white space. Wildcards can be used for non-text fields.

If you need to use a non-wildcard asterisk or question mark in a non-text field, be sure to escape each of them with a backslash. For example,

```
myField:ABC\*DEF\?GHI
```

will match the literal term `"ABC*DEF?GHI"`.

If you need to use a trailing question mark wildcard at the end of a query that starts with a question word (who, what, when, where, why or how), be sure to add a space and some extraneous syntax such as a +, otherwise the natural language query heuristic will discard that trailing question mark. For example:

User Entry	Behavior
`What is aspirin?`	The question mark is ignored
`myField: XX/YY/Z?`	The question mark is treated as a wildcard
`Where is part AB004x?`	The question mark is ignored

`Where is part AB004x? +`	The question mark is treated as a wildcard and the extraneous "+" will be ignored
`myField: XX/YY/Z? +`	The question mark is treated as a wildcard and the extraneous "+" will be ignored

Wildcards at Start of Terms

Wildcards can be placed at the start of terms, such as `*ation`, which is known as a *leading wildcard* or sometimes as a *suffix query*. The syntaxes are the same as described above, but there may be local performance considerations that need to be evaluated.

Lucene and Solr technically support leading wildcards, but this feature is usually disabled by default in the traditional query parsers due to concerns about query performance since it tends to select a large percentage of indexed terms. The Lucid query parser does support leading wildcards by default, but this feature may be disabled by setting the `leadWild` configuration setting in `solrconfig.xml` to 'false'. To address performance concerns, Lucene 2.9+ and Solr 1.4+ now support a 'reversed wildcards' (or 'reversed tokens') strategy to work around this performance bottleneck.

This optimization is disabled by default. To enable this optimization you must manually add the `ReversedWildcardFilterFactory` filter to the end of the index analyzer tokenizer chain for the field types in the `schema.xml` file for the fields that require this optimization.

This affects all fields for the selected field types, so if you have multiple fields of a selected type and do not want this feature for all of them, you must create a new field type to use for the selected field.

The Lucid query parser will detect when leading wildcards are used and invoke the reversal filter, if present in the index analyzer, to reverse the wildcard term so that it will generate the proper query term that will match the reversed terms that are stored in the index for this field.

The rules for what constitutes a leading wildcard are not contained within the Lucid query parser itself. Rather, the query parser invokes the filter factory (if present) to inquire whether a given wildcard term satisfies the rules. There are a variety of optional parameters for the filter factory, described below, to control the rules. The default rules are that a query term will be considered to have a leading wildcard and to be a candidate for reversal only if there is either an asterisk in the first or second position or a question mark in the first position and neither of the last two positions are a wildcard. If a wildcard query term does not meet these conditions, the wildcard query will be performed with the usual, un-reversed wildcard term.

Use of the wildcard reversal filter will double the number of terms stored in the index for all fields of the selected field type since the filter stores the original term and the reversed form of the term at the same position.

There is no change to the query analyzer for the optimized field or field type. The reversal filter factory must only be specified for the index analyzer.

As an example, the index analyzer for field type `text_en` should appear as follows after you have manually edited `schema.xml` to add the wildcard reversal filter at the end of the index analyzer for this field type:

```
<fieldType name="text_en" class="solr.TextField"
                          positionIncrementGap="100">
 <analyzer type="index">
  <tokenizer class="solr.WhitespaceTokenizerFactory"/>
  <filter class="solr.WordDelimiterFilterFactory"
          generateWordParts="1" generateNumberParts="1"
          catenateWords="1" catenateNumbers="1" catenateAll="0"
          splitOnCaseChange="0"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ISOLatin1AccentFilterFactory"/>
  <filter class="com.lucid.analysis.LucidPluralStemFilterFactory"
          rules="LucidStemRules_en.txt"/>
  <filter class="solr.ReversedWildcardFilterFactory"/>
 </analyzer>
 ...
```

You must place the wildcard reversal filter at the end of the index analyzer for the field type since it is reversing the final form of the terms as they would normally be stored in the index.

Although this feature improves the performance of leading wildcards, it will not improve the performance of search terms that have both leading and trailing wildcards, since such a term will still have a leading wildcard even after being reversed. In such a case, which depends on the rule settings, the filter factory will inform the Lucid query parser that such a wildcard term is not a candidate for reversal. In that case, the Lucid query parser would generate a wildcard query using the un-reversed wildcard term.

The filter factory has several optional parameters to precisely control what forms of wildcard are considered leading and candidates for reversal at query time:

- `maxPosAsterisk="n"` -- maximum position (1-based) of the asterisk wildcard ('*') that triggers the reversal of a query term. Asterisks that occur at higher positions will not cause the reversal of the query term. The default is 2, meaning that asterisks in positions 1 and 2 will cause a reversal (assuming that the other conditions are met.)
- `maxPosQuestion="n"` -- maximum position (1-based) of the question mark wildcard ('?') that triggers the reversal of a query term. The default is 1. Set this to 0 and set `maxPosAsterisk` to 1 to reverse only pure suffix queries (i.e., those with a single leading asterisk.)
- `maxFractionAsterisk="n.m"` -- additional parameter that triggers the reversal if the position of at least one asterisk ('*') is at less than this fraction (0.0 to 1.0) of the query term length. The default is 0.0 (disabled.)
- `minTrailing="n"` -- minimum number of trailing characters in query term after the last wildcard character. For best performance this should be set to a value larger than one. The default is two.

These optional parameters only affect query processing, but must be associated with the index analyzer even though they do not affect indexing itself.

Range Queries

A range query is a pair of terms which matches all terms which are lexically between those two terms. The two terms are enclosed within square brackets ("[]") or curly braces ("{}") and separated by the keyword "TO". For example,

- `[cat TO dog]`

Square brackets indicate that the specific terms will also match terms in documents. This is referred to as an inclusive range. Curly braces indicate that the specific terms will not match terms in documents and that only terms between the two will match. This is referred to as an exclusive range.

LucidWorks supports open-ended ranges, in which one or both terms are written as an asterisk ("*") to indicate there is no minimum or maximum value for that term.

In contrast to Lucene and Solr, the brackets and braces can be mixed in LucidWorks, so that one term (either) can be inclusive while the other is exclusive.

Another LucidWorks extension allows the "TO" keyword to be entered in the lower case ("to"). Some examples:

User input	Matches
`[cat TO dog]`	All terms lexically between "cat" and "dog", including "cat" and "dog
`{cat TO dog}`	All terms lexically between "cat" and "dog", excluding "cat" and "dog
`{cat TO dog]`	All terms lexically between "cat" and "dog", excluding "cat", but including "dog"
`[cat TO dog}`	All terms lexically between "cat" and "dog", including "cat", but excluding "dog"
`[* to dog]`	"dog" and all terms lexically less
`[cat to *]`	"cat" and all terms lexically greater.
`[cat to *}`	Same as above because "*" forces inclusive match.
`[* to *]`	All terms
`{* to *}`	Same, because "*" forces inclusive match.

Range queries only work properly for fields that are lexically sorted or trie numeric fields, as specified by the administrator.

Trie Numeric Range Query

Lucene now supports *trie numeric* fields, which enable much faster range queries in addition to being sorted in numeric order rather than lexical order. Each trie numeric field has a datatype, integer, real, or date.

User input	**Matches**
`pageCount:[10 to 20]`	Documents between 10 and 20 pages in size
`weight:[0.5 to 70]`	Documents with weights between 0.5 and 70.0
`dateModified:[1984 to 2005]`	Documents modified between 1984 and 2005
`dateModified:[20-Jan-04 to 05-Feb-07]`	Documents modified between January 20, 2004 and February 5, 2007

Fuzzy Queries

A fuzzy query is a request to match terms that are reasonably similar to the query term. A wildcard is a strict form for specifying similarity, requiring the individual characters of the term to match precisely as written, while a fuzzy query allows characters to be shuffled, inserted, or deleted. Fuzzy query measures the editing distance, which is the number of characters which would have to be moved, inserted, or deleted to match the query term and then comparing that to half the length of the query term as a ratio. In other words, half as many characters as appear in the query term can be shuffled or changed in order to match the query term. Fuzzy query is good for matching similar terms as well as catching misspellings.

A fuzzy query is simply a single term (not a phrase) followed by a tilde ("~"). For example,

`soap~` matches "soap", "soup", "soan", "loap", "sap", "asoap", and so on.

For cases in which you want to require greater or less similarity, an optional similarity ratio can be specified.
The similarity ratio is a float ratio written after the tilde, ranging from 0.0 to 0.999. The default is 0.5. Smaller ratios indicate that less similarity is required. Larger ratios indicate that greater similarity is required. Lucene does not support a ratio of 1.0 which would require no differences from the query term. Lucid will treat any value of 1.0 or greater as 0.999, which effectively requires an exact match unless the term is very long.

Lucene's implementation of fuzzy search is not very effective for very short terms (three characters or less) because it uses half of the term length as the maximum editing distance.

Fields and Field Types

An enterprise search engine organizes the data and properties for documents of the collection into distinct categories called *fields*, each of which has its own type of data as defined by a *field type*. Fields and field types are defined in a *schema*. The formatting and handling of data within fields is performed by a software component known as an *analyzer*. Analyzers are defined for each field type. There is a *crawler* which collects documents and data and analyzes their content using the analyzers and stores the analyzed data in the *index*, where it can be searched by user queries. This is an open-ended architecture that provides tremendous flexibility, but can also add to the level of detail that users, or at least advanced users, need to know to query a LucidWorks search collection.

The system administrator is responsible for setting up the schema and defining the fields and field types. Those details are beyond the scope of this section and are generally not needed by a typical user. The only information the user needs is the list of field names that are supported by the schema and possibly details about how fields may be formatted. Many fields will be simple text fields, so no additional detail is needed, and many other fields will be simple numbers or strings or dates. However, for sophisticated enterprise applications, there may be fields with more complex structure.

The LucidWorks query parser recognizes four distinct field types:

- Text
- Date
- Numeric (trie)
- Other

Text fields behave virtually the same as in a typical Web search engine, with queries consisting of words and quoted phrases.

Date fields have a standard format, but because dates formats vary so widely, a variety of alternative formats are recognized by the Lucid query parser to allow users to more easily specify dates in queries. For example, the year alone can be used in a query rather than writing a range from January 1 to December 31, or a range of years can be used without the non-year portion of the standard format. See Date Queries for more information.

For trie numeric fields, the Lucid query parser will make sure that numbers are properly formed before being handed off to the analyzer. For integer fields, real numbers used as query terms will be truncated to integers. Negative numbers are also permitted for trie numeric fields.

For all other field types, the Lucid query parser simply hands the source query term text off to the analyzer and relies on it to "do the right thing." If the analyzer has a problem, the term will be ignored.

Related Topics

- LucidWorks FieldTypes API
- LucidWorks Fields API

Field Queries

When thinking about what is to be searched, the user has two choices: either to rely on the default field(s) specified by the system administrator or to explicitly specify field names within the query. In general, the former is sufficient, but on occasion the latter is needed.

Lucene had the concept of a single default field, but in Solr that has been replaced with a list of fields known as the *query fields* or *default fields*. Of course, there is no reason that the query fields list could not consist of a single field. Searching for a term will actually search for it in each of the fields listed in the query fields, and a document will match for that term if the term is found in any of the query fields.

Because every enterprise application has its own data needs, you will have to consult your system administrator for the list of field names.

There are three query formats for explicitly specifying field names:

- `title:aspirin`
  Single term - write the field name, a colon, and then the single term (or quoted phrase). Only that one term will be searched in the specified field and subsequent terms will be search in the default fields.
- `title:(cat OR (dog AND fish))`
  Parenthesized sub-query - write the field name, colon, left parenthesis, a sub-query (which may simply be a full query, ranging from a single term to a complex Boolean query with nested parentheses), and a right parenthesis.
- `title:cat OR (dog AND fish)`
  Sticky field name - write the field name, a colon, a space, and then the rest of the query. All subsequent terms will be searched in that *sticky field name*, until a new sticky field name is specified or a right parenthesis is reached at the current parenthesis level, for example.

There are two special pseudo-field names:

- ALL: - searches all fields defined in the schema or even defined dynamically, for example,
  `ALL:cat OR dog`
- DEFAULT - revert to searching the default query fields if in a sticky or parenthesized field name.
  `title:cat OR dog AND DEFAULT:fish`

FYI

Field names are case insensitive, including `ALL` (`All`, `all`) and `DEFAULT` (`Default` and `default`).

Note

If the query parser encounters a field name that is not defined, it will be treated as a simple term. For example, `noField:foo` will be treated as the term list `noField foo`. This is done because it is not uncommon to paste natural language text into the search box, and colon is a not uncommon punctuation character.

Date Queries

When the LucidWorks query parser detects that a field is defined in the collection schema as the field type "date" (or an instance of a "DateField", or a trie), a variety of common date formats are also supported in addition to the traditional Solr date format as defined in the "Solr Date Format" section. These alternative formats are designed to make it easy to specify individual dates, months, and years. They can be used as standalone query terms, or with the range and relational operators. The non-Solr, alternative, date formats will automatically be expanded internally by the LucidWorks query parser into Lucene range queries. For example, `2006` would be treated the same as `[2006-01-01T00:00:00Z TO 2006-12-31T23:59:59.999Z]`.

The common date formats are:

- YYYY
  - 2001
- YYYY-MM
  - 2001-07
  - 2001-7
- YYYY-MM-DD
  - 2001-07-04
  - 2001-7-4
- YYYYMM
  - 200107
- YYYYMMDD
  - 20010704
- YYYY-MM-DDTHH
  - 2001-07-04T08
  - 2001-07-04t08
  - 2001-7-4t08
- YYYY-MM-DDTHH:MM
  - 2001-07-04T08:30
  - 2001-07-04t08:30
  - 2001-7-4t08:30

- MM-YY
  - 07-01
  - 7-98
  - 7-1
- MM-YYYY
  - 07-2001
  - 7-2001
- MM-DD-YY
  - 07-04-01
  - 7-4-01
  - 7-4-1
- MM-DD-YYYY
  - 07-04-2001
  - 7-4-2001
- MM/YY
  - 07/01
  - 7/01
  - 7/1
- MM/YYYY
  - 07/2001
  - 7/2001
- MM/DD/YY
  - 07/04/01
  - 7/4/1
- MM/DD/YYYY
  - 07/04/2001
  - 7/4/2001
- DD-MMM-YY
  - 04-Jul-01
  - 4-jul-01
  - 4-JUL-01
  - 04-Jul-1
  - 4-jul-1
  - 4-JUL-1
- DD-MMM-YYYY
  - 04-Jul-2001
  - 4-jul-2001
  - 4-JUL-2001
- MMM-YY
  - Jan-01
  - jan-01
  - JAN-1

- MMM-YYYY
    - Jan-2001
    - jan-2001
    - JAN-2001

Date Ranges

Even a simple date term can expand into a range, but the alternative date formats can be used in range terms as well.

User Input	Query Interpreted As
`[2001 TO 2005]`	`[2001-01-01T00:00:00Z TO 2005-12-31T23:59:59.999]`
`{2001 TO 2005}`	`{2001-12-31T23:59:59.999Z TO 2005-01-01T00:00:00Z}`
`[Jun-2001 to 7/2005]`	`[2001-06-01T00:00:00Z TO 2005-07-31T23:59:59.999Z]`
`[7/4/2001 to 9-7-2002]`	`[2001-07-04T00:00:00Z TO 2002-09-07T23:59:59.999Z]`

The alternative data formats may also be used with relational operators to imply ranges.

User Input	Query Interpreted As
`dateModified: < 2001`	`dateModified: [* TO 2001-01-01T00:00:00Z]`
`dateModified: <= 2001`	`dateModified: [* TO 2001-12-31T23:59:59.999Z]`

Date format expansion only occurs for fields that the LucidWorks query parser recognizes as being "date" fields. A date stored in a text field or string field must be manually and correctly formatted by the user.

Solr Date Format

The user-friendly date formats supported by the LucidWorks query parser are in addition to the date format that is supported by Solr. Solr provides a format for a specific date and time, as well as *date math* to reference relative dates and times based on a starting date and time.

The simplest form of Solr date is the keyword '`NOW`' which refers to the current date and time. It is case sensitive in Solr, but the Lucid query parser will permit it to be in any case. '`NOW`' can be used either if no explicit date or date math is specified, or it can be used if date math is specified without an explicit date.

An explicit date is written in Solr using a format based on ISO 8601, which consists of a string of the form `yyyy-mm-ddThh:mm:ss.mmmZ`, where 'yyyy' is the four-digit year, the first 'mm' is the two-digit month, 'dd' is the two-digit day, 'T' is the mandatory literal letter 'T' to indicate that time follows, 'hh' is the two-digit hours ('00' to '23'), the second 'mm' is the two-digit minutes ('00' to '59'), 'ss' is the two-digit seconds ('00' to '59'), optionally '.mmm' is the three-digit milliseconds preceded by a period, and 'Z' is the mandatory literal letter 'Z' to indicate that the time is UTC ('Zulu'). The millisecond portion, including its leading period, is optional. Trailing zeros are not required for milliseconds.

Note

The Lucid query parser does not require the 'Z' and will translate a lower case 't' or 'z' to upper case.

Some examples:

- `NOW`
- `Now`
- `now`
- `2008-07-04T13:45:04Z`
- `2008-07-04T13:45:04.123Z`
- `2008-07-04T13:45:04.5Z`

Solr requires hyphens between the year, month, and day, but the LucidWorks query parser will add them if they are missing.

Solr date math consists of a sequence of one or more addition, subtraction, and rounding clauses. '+' introduces an addition clause, '-' introduces a subtraction clause, and '/' introduces a rounding clause. Addition and subtraction require an integer followed by a calendar unit. Rounding simply requires a calendar unit.

The calendar units are:

- `YEAR` or `YEARS`
- `MONTH` or `MONTHS`
- `DAY` or `DAYS` or `DATE`
- `HOUR` or `HOURS`
- `MINUTE` or `MINUTES`
- `SECOND` or `SECONDS`
- `MILLISECOND` or `MILLISECONDS` or `MILLI` or `MILLIS`

Example rounding clauses:

- `/YEAR`
- `/MONTH`
- `/DAY`
- `/HOUR`

Example addition and subtraction clauses:

- `+6MONTHS`
- `+3DAYS`
- `-2YEARS`
- `-1DAY`

Examples using dates and date math:

- `NOW`
- `NOW/DAY`
- `NOW/HOUR`
- `NOW-1YEAR`
- `NOW-2YEARS`
- `NOW-3HOURS-30MINUTES`
- `/DAY`
- `/HOUR`
- `-1YEAR`
- `-2YEARS`
- `/DAY+6MONTHS+3DAYS`
- `+6MONTHS+3DAYS/DAY`
- `2008-07-04T13:45:04Z/DAY`
- `2008-07-04T13:45:04Z-5YEARS`
- `[2008-07-04T13:45:04Z TO 2008-07-04T13:45:04Z+2MONTHS+7DAYS]`

Whitespace is not permitted.

As a general rule, any tail portion of a proper date/time term can be omitted and the Lucid query parser will fill in the missing portions. But, the result will be an implicit range query. For example:

- `2008-01-01T00:00:00Z`
- `2008-01-01T00:00:00`
- `2008-01-01T00:00` same as `[2008-01-01T00:00:00Z TO 2008-01-01T00:00:59Z]`
- `2008-01-01T00:` same as `[2008-01-01T00:00:00Z TO 2008-01-01T00:59:59Z]`
- `2008-01-01T` same as `[2008-01-01T00:00:00Z TO 2008-01-01T23:59:59Z]`
- `2008-01-01` same as `[2008-01-01T00:00:00Z TO 2008-01-01T23:59:59Z]`
- `2008-01` same as `[2008-01-01T00:00:00Z TO 2008-01-31T23:59:59Z]`
- `2008` same as `[2008-01-01T00:00:00Z TO 2008-12-31T23:59:59Z]`

The same technique can be used in explicit date range queries and with relational operators.

See the `org.apache.solr.schema.DateField` and `org.apache.solr.util.DateMathParser` Java classes for more information about the Solr date/time format.

If there is any parsing problem in a date term, the LucidWorks query parser will catch the exception and simply ignore the date term.

Non-Text, Date, Numeric Field Queries

The LucidWorks query parser has a number of features to support text, date, and trie numeric fields, but other field types are simply passed through the schema query analyzer that has been specified by the administrator in the schema. Non-trie numbers (integers, floats) and strings are two common non-text/date/numeric field types.

String fields may seem similar to text fields, but the full string is one term, even if it has embedded whitespace and punctuation.

If you make a mistake in formatting query text for a non-text, non-date, non-trie numeric field, the offending query term will simply be ignored. String fields would not typically have any mistakes, but any mistakes could cause a failure to match documents properly. A non-digit or a decimal point in an integer field or an improperly formatted float are mistakes that could occur in non-trie numeric fields.

There are two forms for terms in non-text/date/numeric fields:

- **Simple term**: a term as in a text field, typically delimited by whitespace or one of the special syntax characters.
- **Quoted string**: looks like a quoted text phrase, but every character between the quotes, including whitespace and any special syntax characters are considered part of the text of the term.

In both forms any special syntax characters or whitespace can be escaped (with backslash). For many cases, either form can be used. An advantage of quoted string terms is that less escaping of special syntax characters is needed. A key difference is that wildcard character cannot be used as wildcards within quoted strings, but they can be used as wildcards in simple terms.

Some examples:

- `myIntField:123`
- `myFloatField:123.45`
- `myStringField:Hello`
- `myStringField:Hello\ World!` (With escaped embedded space and exclamation point.)
- `myStringField:"Hello World!"` (Same, but no escaping needed.)
- `myIntField:"123"`
- `myStringField:*cat*` (Wildcard matches string values containing the sub-string "cat".)
- `myStringField:"cat"` (Non-wildcard matches Matches string values that are literally "**cat**".)
- `myStringField:"\"cat\""` (Matches string values that are the text "cat" with enclosing double quotes.)
- `myStringField: ({[A~B\:C^D]})` (Matches the string ""({[A~B:C^D]})".)
- `myStringField:"({[A~B:C^D]})"` (Same, but no escaping needed.)

Whitespace

Whitespace can be used as liberally as a user desires between terms and operators, with only a few exceptions. There must not be any space:

- Between a field name and the colon (':') which follows it.
- Between a term operator ('{+}' or '-') and the term that follows.
- Between a term and a suffix modifier (tilde or circumflex), or within a suffix modifier, or between suffix modifiers.
- Between a backslash ("\") and the special character that it is escaping.

Also, whitespace is *not* required, but permitted at the following points:

- Before or after parentheses.
- Before or after a non-keyword Boolean operator ("&&", "||" or "!".)
- Before or after a double quotation mark (") enclosing phrases.
- Before a term operator ('{+}' or '-'.)

There must be either whitespace or some operator, such as a parenthesis, after any single term and any '+' or '-' operator that follows it, otherwise the '+' or '-', and any non-whitespace that follows, is considered as part of the preceding term.

Line breaks are treated as whitespace. So, very long queries can be broken into shorter lines for readability with no impact of the query interpretation.

Quoted phrases may be split over two or more lines as well.

Term Operators

In addition to the Boolean operators ('AND', 'OR', 'NOT') used to construct complex queries with sub-queries, there are operators that are used at the term level, within term lists. They are written immediately before a term, without whitespace.

- '+' - Require a term - it must be present in documents
- '-' - Exclude a term - it must not be present in documents
- Relational operators - '==', '!=', '<', '<=', '>', and '>='. Whitespace is permitted after the operator.

Term operators, field names, keyword options, and prefix modifiers can be written in any order. For example:

```
title:>cat
```

```
>title:cat
```

```
+title:dog
```

```
title:+dog
```

```
-title:frog
```

```
title:==frog
```

```
==title:frog
```

```
nostem:+title:bats
```

Selecting All Documents

The LucidWorks query parser has a special feature that selects all documents as efficiently as possible using a special Lucene API. The syntax is simply `*:*`. The LucidWorks default Search UI allows querying for all documents simply by hitting the **Search** button with an empty query box.

Alternately, a query consisting of a single asterisk ('*') will select all documents that contain a term in the list of default search fields. This will not necessarily select all documents since there may be some documents in the collection which do not have values in the default query fields.

All documents containing a term in a specified field can be queried by writing the field name, a colon, and the asterisk. For example,

- `title:*`

would return all documents that have titles, although not all documents may have titles.

Combining that with exclusion permits a query to find all documents that do not have a value in the specified field. For example,

- `-title:*`

is equivalent to:

- `*:* -title:*`

But that is not necessarily equivalent to:

- `* -title:*`

because the latter depends on precisely which fields are listed in the query fields, so it may return no documents or a subset of the total documents in the collection.

Relational Operators

In addition to the Lucene range syntax, the LucidWorks query parser supports the standard collection of *relational operators* (also known as *comparative operators*, '==', '!=', '<', '<=', '>=', and '>'), for example:

- `Nevada politics >= dateCreated: 2003`
- `Nevada politics dateCreated: >= 2003`

Whitespace may be freely used both before and after a relational operator.

Other term prefix modifiers, such as a field name and term keyword options, can be combined with a relational operator, in any order. For example, the following queries are equivalent:

- `cat nosyn:body: < dog`
- `cat < nosyn:body: dog`
- `cat nosyn:body: <dog`
- `cat nosyn: <body: dog`
- `cat nosyn: < body: dog`
- `cat nosyn: body: < dog`
- `cat nosyn: body: <dog`

The '==' and '!=' relational operators are equivalent to the '+' and '-' term operators. So, these queries are equivalent:

- `cat +dog -fox`
- `cat ==dog !=fox`
- `cat == dog != fox`

Accented Characters

The LucidWorks query parser supports text terms written using the so-called accented characters that appear in Unicode and ISO Latin-1 as hex codes 00A1 though 00AF, but it is common that the administrator will set up the schema so that a *filter* such as the `ISOLatin1AccentFilter` will be included in field type analyzers for the purpose of *stripping* the accents by mapping the accented characters to unaccented ASCII equivalents.

For example, `Café Française` would be treated identically to `Cafe Francaise`

Typically, accents are removed when documents are indexed. That means that they must also be removed at query time so that query terms can match indexed terms. But, the administrator could decide to preserve accented characters at index time, in which case accents will then also need to be preserved at query time.

The LucidWorks query parser normally bypasses the analyzer for text fields, but it will invoke the accent removal filter if it is present and has the word `Accent` in its name.

It is technically possible for the administrator to construct an index filter that indexes both the accented and unaccented forms of terms and remove the query accent filter, and then the user could query the unaccented term to get both accented and unaccented terms or query the accented term and get only the accented terms.

Accents are not normally stripped for non-text fields, but that depends purely on whether each non-text field type does or does not have an accent removal filter specified.

Building Advanced Queries

This section describes more advanced search queries some of the most commonly used types of search queries and gives examples of how LucidWorks processes them.

Minimum Match for Simple Queries

None of the optional terms in a simple Boolean query is required to be present for a document to be selected by the query, but in some cases you would like to require that at least *some* of the optional terms be present. This can be accomplished by supplying a *minimum match* modifier, which specifies either a count or percentage of the optional terms that are required for the immediate following simple Boolean query. The minimum match can be specified either with the `minMatch` (or `atLeast`) keyword option or by enclosing the simple query within parentheses and appending a tilde ('~') followed by the term count or percentage (which may also be a fraction). The keywords `minMatch` and `atLeast` are synonymous. For example:

- `minMatch:1 +pet cat dog fish rabbit -snakes`
- `minMatch:2(+pet cat dog fish rabbit -snakes)`
- `minMatch:25%(+pet cat dog fish rabbit -snakes)`
- `minmatch:0.25(+pet cat dog fish rabbit -snakes)`
- `minMatch:50% +pet cat dog fish rabbit -snakes`
- `minmatch:25(+pet cat dog fish rabbit -snakes)`
- `atLeast:25(+pet cat dog fish rabbit -snakes)`
- `atleast:25(+pet cat dog fish rabbit -snakes)`
- `(+pet cat dog fish rabbit -snakes)~1`
- `(+pet cat dog fish rabbit -snakes)~25%`
- `(+pet cat dog fish rabbit -snakes)~25`
- `(+pet cat dog fish rabbit -snakes)~0.25`

If a space follows the `minMatch` keyword option, then the setting is *sticky* and applies to all subsequent Boolean queries until the next closing parenthesis, otherwise the setting applies only to the parenthesized Boolean query that immediately follows.

Since each of the above examples has four optional terms, 25% means that one out of four of the optional terms must be present in a document for it to be selected by the query. A value of 50% (or two) requires that at least half of the optional terms be present in a document.

A value of 0 or 0% means that no optional terms are required. This is the default. A value that matches (or exceeds) the count of optional terms or 100% means that all optional terms are required.

If a whole number is specified and no percentage is present, the Lucid query parser will do an excellent job of *guessing* whether the number is a count of terms or a percentage. In other words, the percent symbol ('`%`') is almost always optional.

As a special case, a small percentage, such as `1%`, is treated as requiring a minimum of one optional term to match.

Keyword option names are *not* case sensitive, although they tend to be written in their proper *camel case* form in this documentation. So, `minMatch`, `minmatch`, `MinMatch`, and `MINMATCH` are all equivalent.

The administrator can change the default (for example, to one to assure that at least one optional term is present) using the `minMatch` configuration setting.

Negative Queries

A term list with only the '-' term operator and no '+' term operators is known as a *negative query* and will query all documents that do not have the specified '-' terms. This is equivalent to a term list requesting all documents and then excluding the specified terms.

User Input	Equivalent to

`-cat`	`*:* -cat`
`-cat -dog`	`*:* -cat -dog`

Escaping Wildcard Characters

The wildcard characters, "*" and "?" are a special case. They are always part of the term in which they are embedded, but they have their special wildcard meaning rather than being simply characters in a term. But, if you do have a non-text field in which the wildcard characters are actually text in that field, you can escape them using a backslash. For example,

- `myField: E\*TRADE`
  The term will literally be "`E*TRADE`" rather than a wildcard.
- `myField: x\?y\?z`
  The term will literally be "`x?y?z`" rather than a wildcard.

Note that due to a limitation of Lucene, if there are any non-escaped wildcard characters in a term, escaping will be ignored for all other wildcard characters in that term. For example,

- `myField: E\*TRA?E`

Will be treated as a wildcard query for the term "`E*TRA?E`", with both the `*` and `?` being treated as wildcard characters. On the other hand,

- `myField: E\*TRA\?E`

Will be treated as a non-wildcard term query for the term "`E&*TRA?E`".

Proximity Operations

A *proximity* query searches for terms that are either near each other or occur in a specified order in a document rather than simply whether they occur in a document or not.

Phrase Proximity Queries

Exact phrase matching is a powerful query tool, but frequently the phrasing used in relevant documents is not exactly the same. It is commonly the case that there are extra terms, or the terms may be in another order. In other cases, the phrase terms may be relatively near, with quite a few extra words between them. For example, the following two queries may return different results even though they are semantically equivalent:

```
"team development"
```

```
"development of teams"
```

The difference between the two is an extra word in the middle and a reversal of the two key terms.

We can write a single *phrase proximity query* that will match both phrases:

```
"team development"~3
```

The tilde ("~") is used after a quoted phrase to indicate a phrase proximity search. It is followed by an integer (whole number) which is the maximum editing distance for phrases that will match the query phrase. The editing distance treats each term as a single unit and measures how many unit terms need to be moved to translate from one phrase to another. In this case, it takes one unit to move "team" to "of", a second unit to move it to "development", and a third unit to move it before "development".

To query for two terms that are within 50 words of each other:

```
"cat dog"~50
```

To query a person's name and allow for an optional middle initial:

`"John Doe"~1` matches `"John Doe"` and `"John Q. Doe"`

To query a person's name and allow for both first name first or last name first:

`"John Doe"~2` matches `"John Doe"` and `"Doe, John"`, as well as `"John Q. Doe"`

Advanced Proximity Operators

The Lucid query parser also supports advanced proximity query operators to specify more elaborate sequences of terms and to control the order of terms and how many intervening terms are permitted. The advanced proximity operator keywords are:

Advanced Operator	**Sample Query**	**Matches**
`NEAR`	`x near y`	Documents containing "x" within 15 terms of "y", either before or after
`BEFORE`	`x before y`	Documents containing the term "x" no more than 15 terms before the term "y"
`AFTER`	`x after y`	Documents containing the term "x" no more than 15 terms before the term "y"

These operators are case insensitive and may be upper, lower, or mixed case, unless the `opUp` configuration setting is set to "true", which would then treat them (and all other operator keywords) as normal terms unless they are entirely upper case.

Excluding Terms from Advanced Proximity Queries

Normally, any combination of terms may appear between the terms that mark the start and end of an advanced proximity query (the BEFORE, AFTER, and NEAR operators), but in some situations it is desirable to prevent specific terms from occurring between those start and end terms. Just as with a simple keyword query, this exclusion can be done by listing terms preceded by the minus sign '-' or NOT operator.

For example, these pairs of queries are equivalent:

```
George NEAR Washington -person
George NEAR Washington NOT person
```

```
George NEAR Lincoln -person
George NEAR Lincoln (NOT person)
```

Also, the exclusions may be specified on either side of the proximity operator, so the following queries are equivalent:

```
George NEAR Lincoln -person
George -person NEAR Lincoln
```

```
George NEAR Lincoln (NOT person)
George (NOT person) NEAR Lincoln
```

Controlling Distance Between Terms

By default, the distance between the two terms of a proximity operator can be up to 15 additional terms. That default distance is controlled by the `nearSlop` configuration setting. But if you need more or fewer intervening terms for a specific proximity operator, you can specify the desired limit of intervening terms by writing a colon (":") and the number immediately after the operator name. For example,

```
x before:3 y
```

matches documents containing "x" no more than three terms before "y".

A distance of 0 (zero) means no intervening terms. For example,

```
x before:0 y
```

is the same as:

```
"x y"
```

which matches documents where the terms are adjacent and in that order.

Composing Longer Sequences of Terms

The advanced proximity operators can be composed (or "daisy-chained") to match more complex term sequences. For example:

```
x before y before z
```

matches documents containing "x" before "y" with no more than 15 intervening terms and followed by "z" with no more than 15 intervening terms after "y".

The distance limit can be controlled for each proximity operator, such as:

```
x before:10 y before:100 z
```

which requires that there be no more than 10 terms between "x" and "y", but "z" can be up to 100 terms after "y".

Any combination of any number of NEAR, BEFORE, and AFTER proximity operators can be composed into a sequence, such as

```
cat near dog before:50 fox after fish near:3 bat before zebra
```

Left to Right Evaluation Order

When multiple advanced proximity operators are composed, they are evaluated left to right, except as parentheses are used to explicitly specify the evaluation order. So, the previous example is evaluated as:

```
(x before y) before z
```

In fact, the evaluation order does not matter in that example, which could also be written as:

```
x before (y before z)
```

But evaluation order does matter with:

```
x near:3 (y before:50 z)
```

where the intent is that "x" could be shortly before or after either end of the "y"/"z" sequence. But,

```
x near:3 y before:50 z
```

would evaluate as:

```
(x near:3 y) before:50 z
```

which would match "x" close to "y" but not close to "z".

Within parentheses used for operands of proximity operators, only the OR and proximity operators can be used. Other operators will be treated as if they were the OR operator.

Quoted Phrases

Quoted phrases with any number of terms can be used as the operands of the proximity operators. For example,

```
"First step" before:200 "last step"
```

The terms in the quoted phrase must occur in order, with no intervening terms between the quoted terms.

Quoted Proximity Phrases

Quoted phrases may specify a maximum number of terms that may appear between the terms of the phrase, using the usual quoted phrase proximity query notation of a tilde ("~") and the number of terms permitted. For example,

```
"proposal development"~3 near:50 project
```

Would match the terms "proposal" and "development" (in that order) with no more than three intervening terms and occurring no more than 50 terms before or after "project".

Note

Unlike normal quoted proximity phrases, the phrase terms are expected to occur in order. So, this example will not match `"development proposal...project"`.

Alternative Terms

When several different terms are permitted at a position in a proximity sequence, the alternative terms can be specified using the OR operator and parentheses for either or both terms of the operator. For example,

```
(cd-rom or dvd) before:1 drive
```

would match documents with the term "drive" preceded by either "cd-rom" or "dvd" with at most one intervening term. Alternatives can also be used with composed proximity operators. For example,

```
(cd-rom or dvd) before:1 ((built-in or external) before:0 drive)
```

which requires "built-in" or "external" to immediately precede "drive", but an intervening term is permitted after "cd-rom" or "dvd".

Alternatives can also be quoted phrases. For example, `("In the beginning" or "At the start" or "Starting out") before:1000 "the end"` will match documents containing the phrase "the end" preceded by either the phrase "In the beginning", "At the start", or "Starting out" with up to 1,000 intervening terms.

Term Lists

A phrase that is not enclosed within quotes is known as a *term list* and may be used as either of the operands of a proximity operator, where it will be treated as if it were a quoted phrase. For example,

```
pets before animal judgments before book
```

will match the same documents as:

```
pets before "animal judgments" before book
```

Parenthesized Proximity Expressions in Term Lists

Although term lists with proximity operators may seem like a mere convenience to avoid typing the quotes around a phrase, the construct is much more powerful. Each of the terms in a proximity term list can be one of:

- Single term (but no wildcard or fuzzy term)
- Quoted phrase
- Parentheses enclosing:
  - One or more proximity operators (evaluated left to right)
  - Another term list
  - List of term alternatives separated by OR operators. Each term alternative can be a full proximity expression, including nested parentheses.

For example,

```
red (light or sign) picture near street
```

would be equivalent to:

```
("red light picture" or "red sign picture") near street
```

which could also be written using nested term lists:

```
(red light picture or red sign picture) near street
```

which is also equivalent to:

```
((red light picture) or (red sign picture)) near street
```

Single Field

Although field names can be used for terms within a proximity expression, only the first field name is used and the others are ignored since an entire proximity expression is evaluated within only a single field.

```
title:x after (body:y near author:z)
```

is evaluated as:

```
title: x after (y near z)
```

A proximity query with no field or the DEFAULT field will query against all of the fields listed in the `qf` (query fields) request parameter. The proximity query will be evaluated against each field in turn and the results combined with the disjunction max query operation. But, that will still evaluate the full proximity query expression on only one field at a time.

Boolean Operations on Proximity Expressions

Multiple proximity expressions, each with its own field, can be used within a single query simply by combining them with the AND, OR, or NOT boolean operators. The precedence of the boolean operators is such that entire proximity expressions will be evaluated before the surrounding boolean operators. So,

```
title: red before light or body: empty before tank
```

would evaluate as:

```
(title: red before light) or (body: empty before tank)
```

The AND operator can be used to require a set of proximity queries to be satisfied, such as:

```
(title: red before blue) and (body: night after day) and (town near city)
```

where "red" must occur before "blue" in the title field, "night" must occur after "day" in the body field, and "town" must occur near "city" in any field.

Term Boosting

Although the Lucid query parser will automatically add relevancy boosting for bigrams and trigrams of query terms, the sophisticated user may also explicitly add a boost factor for any term.

A boost factor is a suffix modifier placed after a term which consists of a circumflex ('^') followed by a decimal number indicating a multiplication factor to use in the relevancy calculation for a term. For example:

- `cat^4 dog^1.5 fox "the end"^0.3`

The default boost factor is 1.0, but it is actually derived from the default boost factors specified for the various fields given in the default query field configuration which is controlled by the administrator.

A boost factor of 1.0 indicates that there should be no change from the default boost factor. A factor greater than 1.0 will increase the relevancy of the term. A factor less than 1.0 will decrease the relevancy of the term.

In the example above, "fox" will get the default boosting, "dog" will get modestly higher boosting, "cat" will get significantly higher boosting, and "the end" will have its relevancy reduced well below the default.

You can also give a relevancy boost factor to a term list or sub-query by enclosing it within parentheses. For example:

- `(cat +dog -fox)^2.5`
- `(cat OR dog)^3.5 AND (fox NOT bat)^0.5`

Boolean Relevancy Boosting

Boolean 'AND' and 'OR' operators will also participate in relevancy boosting, by treating the operators as the text words 'and' and 'or' and then combining them into phrases with the last term of the term list to their left and the first term of the term list to their right. For example, each of the following queries will give a higher relevancy ranking for the phrase 'hit and run' than a document that simply contains the terms 'hit' as well as 'run':

- `hit AND run`
- `hit and run`
- `hit && run`
- `hit "and" run`
- `"hit and run"`

Query Analysis for Relevancy Boosting

Bigrams, trigrams, unigrams, and n-grams are generated in the analysis of a user query to identify sequences of terms; in this context, an n-gram is a series of terms or words, though in other contexts an n-gram may refer to a series of characters. A bigram is a sequence of two terms, a trigram is a sequence of three terms, a unigram is a single term, and an n-gram is any sequence of terms, but generally four or more terms. A term list is an n-gram. In the context of n-grams, a quoted phrase counts as a single term. Although the user of the Lucid query parser does not need to worry about this level of detail since it is handled automatically by LucidWorks, it is helpful to understand how Lucid is going to analyze the query, perform the search, and rank the results for better relevance. Basically, the Lucid query parser uses the n-grams (particularly the bigrams and trigrams) from the original query to boost results that contain not only the discrete query terms, but n-grams of the query terms as well.

For example, both of the following queries match documents containing the phrase "meet the press":

- `meet the press`
- `"meet the press"`

The first query consists of three terms, or two bigrams ("meet the" and "the press"), and one trigram ("meet the press"). The second is actually a unigram because a quoted phrase counts as a single term.

The first query also returns any documents that contain "meet" and "press" ("the" is a stop word), even if the document does not contain the full sequence "meet the press". Traditionally that might be annoying, but the Lucid query parser automatically adds extra clauses to the user query to `OR` in the bigrams and trigrams from the query with a boost factor so that occurrences of "meet the", "the press", and "meet the press" will rank higher than documents that merely contain "meet" and "the".

Superficially, the second query might seem better because it is more precise, but sometimes extra words may be present so that multi-word fragments of the phrase might match, but will not if the full phrase is used.

A simple natural language phrase can be used directly as a query and can be expected to return quite good results without the need to add extra operators or specific formats. This point may not be completely obvious when using a simple three-word phrase, but should be more clear with a longer sentence fragment, such as:

- `The company meeting was attended by employees`
- `"The company meeting was attended by employees"`

The precise quoted phrase will match the exact phrasing so will not catch the statement if it is reworded as "Employees attended the company meeting." But the first query will match that rewording, and rank it reasonably high due to the match on the trigram "The company meeting".

Term Modifiers

One method for selecting advanced search features is the use of *term modifiers*, which precede or follow a term.

There are three forms of term modifier that may appear before a term, referred to as a *prefix modifier*, all of which consist of a name followed by a colon:

- **Field name**, for example:
  `title:cat`
- **Pseudo-field name**: ALL, DEFAULT, *, for example:
  `ALL:cat DEFAULT:dog`
- **Keyword options**: nostem, nosyn, and so on. Any number of keyword options can be specified for a single term, each with its own colon, for example:
  `nostem:nosyn:title:paintings`

Note

There is no whitespace between prefix modifiers or the term to which they apply, if they are intended to apply to the single term (or quoted phrase or parenthesized sub-query) immediately following the colon. But, if there is a space, then the modifier will be a *sticky modifier* that applies to all subsequent terms at the same parentheses nesting level.

A term may also be preceded by a relational operator or a '+' or '-' operator, but those are considered *term operators* rather than term modifiers.

There are three forms of term modifier that may appear after a term, also called a *suffix modifier*:

- **Boost factor**: a circumflex ('^') followed by a decimal number indicating a multiplication factor to use in the relevancy calculation for a term, which may be greater than 1.0 to increase boosting or less than 1.0 to lower boosting, for example:
  `cat^4 dog^1.5 "the end"^0.3`

- **Proximity distance**: a phrase followed by a tilde ('~') followed by an integer to specify that additional words may occur between the terms of the phrase, as well as reordering of the terms, up to the specified distance.
  `"product security"~5`
- **Fuzzy search**: a single term followed by a tilde ('~'), optionally followed by a decimal number to specify that the term should match similar terms, where the number specifies how similar.
  `soap~` (Defaults to 0.5.)
  `soap~0.5`
  `soap~0.1` (Not very similar.)
  `soap~0.99` (Virtually identical.)

Note

Lucene considers wildcards and range searches to be term modifiers, but in this guide they are discussed separately.

Default Query Fields

Although explicit field names can be specified for all terms, a default set of field names will be used for terms which are not preceded by a field name (or sticky field name). The administrator must decide which fields make the most sense for default fields for the application. In some cases that may be a single field, which may be a merger of a number of separate fields, but it may also be a list of field names. The `qf` configuration setting lists the default fields and their boosts.

The default field list may also specify default term boost factors for the fields. A field name can be followed by a circumflex and the boost factor for that field.

For example, the administrator might set the default query fields configuration setting to:

```
body title^5 abstract
```

That will cause the query:

```
cat title:dog fox
```

to be equivalent to:

```
(body:cat title:cat^5 abstract:cat) title:dog^5 (body:fox title:fox^5 abstract:fox)
```

Technically, LucidWorks generates a *disjunction maximum query* if multiple fields are specified for the default query fields.

The LucidWorks query parser also has the ability to support an asterisk ("*") for the field list to indicate that all fields should be searched when no field is specified for a query term.

Empty Queries

The query user interface may prevent the user from entering an empty query, but the LucidWorks query parser supports an alternate query string (see the `q.alt` configuration setting) to be used in such cases. This string can be configured by the administrator, but defaults to `*:*`, which will return all documents.

Queries with Unicode Characters

Although the LucidWorks query parser itself is capable of accepting Unicode characters directly, it is usually not very convenient to enter them on a typical computer keyboard. As with Lucene and Solr, LucidWorks supports a variation of the Java escaping format to enter explicit Unicode characters as hexadecimal character codes. Each explicit Unicode character is introduced with a backslash "`\`", the letter "`u`", followed by one to four hexadecimal digits.

Unlike Lucene and Solr, which require a lower-case "`u`" and require exactly four digits, LucidWorks allows upper-case "`U`" and any leading zero digits need not be entered, unless the first character after the hexadecimal code is itself a character which is used for a hexadecimal digit (digits "`0`" to "`9`", letters "`a`" to "`f`", and letters "`A`" to "`F`".) But as a general practice it is safest and most consistent to write the full four digits with any leading zeros

Examples for the word "Cat":

- `Ca\u0074`
- `Ca\U074`
- `Ca\u74`
- `C\U61t`
- `\u0043at` (zeroes needed since 'a' is a hex digit)
- `\u0043\u0061\u0074`

Examples for the word "Cattle9":

- `Cattle\u39`
- `Ca\u74tle9`
- `C\u0061\U74\u074\u006ce9`
- `C\u0061\u74\u074\u6c\u65\u0039`
- `\u0043\U0061\u0074\u0074\u006c\u0065\u0039`

Explicit Unicode characters are assumed to be part of query terms and will not be interpreted as query operators. For example, `\u0021` will not be treated as if it were the "`!`" NOT operator.

Escaping Special Syntax Characters

Many non-alphanumeric characters will be accepted within a term, such as hyphen (-), period (.), comma (,), asterisk (*), at sign (@), number sign (#), dollar sign ($), and semicolon (;), but a handful have special meaning to the Lucid query parser, such as the non-keyword Boolean operators parentheses ( ), colon (:), double quotation mark ("), circumflex (^), and tilde (~). Terms are commonly delimited with white space, but the special syntax characters will delimit terms as well.

The special syntax characters are:

```
&& (But a single "&" is in fact permitted in a term)
|| (But a single "|" is in fact permitted in a term)
\ (Backslash)
!
^
~
(
)
{
}
[
]
:
"
==
<
>
&nbsp; (space)
```

A plus sign, "+", or minus sign, "-", at the beginning of a term is also treated as a special syntax character.

Usually, the user need not be concerned in any way about the special syntax characters unless they explicitly wish to use them as operators, but some non-text fields may have terms that use some of the special syntax characters. In those cases, individual special characters can be *escaped* by preceding them with a backslash, "\". Any character can be escaped without any harm or translation.

For example, all of the above listed special syntax characters can be escaped to be used in terms for non-text fields as shown in this odd-looking but valid query term:

```
myField: A\&&B\|\|C\\D\!F^F~G\ (H)I\{J\}K\[L\]M\:N\"O\==P\<Q\>R
```

which passes the following term to the field analyzer for myField:

```
A&&B\|\|C\D\!F^F~G (H)I{J}K\[L\]M:N"O==P<Q>R
```

Alternatively, a term for a non-text field can simply be enclosed within quotes, although any quotes or backslashes within the term must still be escaped. The previous example can be written as follows:

```
myField: "A&&B||C\\D!F^F~G (H)I{J}K[L]M:N\"O==P<Q>R"
```

Here are some examples of special syntax characters which do not need to be escaped, as per the rules given above:

- `http://www.foo.com/index.html`
  A URL
- `info@foo.com`
  @ has no special syntax meaning and periods are allowed
- `20010630T12:30:00Z`
  colon appears to be in a time value
- `AT&T`
  Single ampersand is considered a valid character in terms
- `ab|cd`
  Single vertical bar is considered a valid character in terms

The apostrophe or single quote mark, "`'`", is not treated the same as a double quotation mark. It is commonly used for contractions and possessives. It will be preserved for non-text fields, but the typical analyzer for text fields will discard it.

Term Keyword Options

Term keyword options provide a flexible mechanism for controlling the interpretation of a term. A term keyword option is a keyword followed by a colon that appears before a term. Any number of keyword options can be specified for a single term, each with its own colon. For example:

```
nostem:nosyn:title:paintings
```

The supported term keyword options are:

- `like`: to indicate that specified terms are not required, but will boost relevancy if present, so that selected documents will be "like" the specified terms. Can also be used to select documents containing the most relevant terms from a document specified by its document id.
- `minMatch`, `atLeast`: to set the minimum count of percentage of optional terms in a term list that must be present in selected documents. See the "Minimum Match for Optional Terms of Simple Boolean Queries" section.
- `syn`, `nosyn`: to enable or disable synonym expansion of the following term.
- `stem`, `nostem`: to enable or disable stemming of the following term. Although supported by the parser, there is not currently search support for both stemmed and unstemmed terms.
- `debugLog`: to enable debug output for a query to permit the administrator to examine the detailed query interpretation.

Term keyword options and any field name can be written in any order, so that the following queries are equivalent:

```
nosyn:title:tv
```

```
title:nosyn: tv
```

Like Term Keyword Option

You can use the `like` term keyword option to specify terms that are not required in documents but will enhance relevancy if they are present: this option selects documents that are *like* a set of terms rather than absolutely requiring the terms. This option can specify a single term, a parenthesized list of terms, or be written as a sticky option that applies to all subsequent terms at this current parenthesis level. For example,

- President like: Lincoln Washington
- President like:(Lincoln Washington)
- President like:Lincoln like:Washington

All three forms are equivalent and will return all documents that have the term "President", with documents that also contain "Lincoln" or "Washington" ranked higher, and documents containing all three ranked highest.

> The single-term form cannot be used if the term has any punctuation or digits, because that triggers the *like document* feature which extracts high-relevancy terms from the specified document and then uses that term list as if it were specified using the like term keyword option.

A query may not even have *any* required terms. For example,

- like: Lincoln Washington Roosevelt
- like:(Lincoln Washington Roosevelt)
- like:Lincoln like:Washington like:Roosevelt

All three forms are equivalent and will return all documents that have at least one of the terms "Lincoln", "Washington", or "Roosevelt", with documents containing more of the terms ranked higher.

The `like` option can be used to reference documents that have text *similar* to a passage. For example,

- like: Four score "and" seven years ago our fathers brought forth
- like:(Four score "and" seven years ago our fathers brought forth)

Both forms are equivalent and will return all documents that have at least one of the words listed, with documents containing more of the words ranked higher and with documents containing more of the words adjacent as listed ranked even higher. Note: The quotes around "`and`" are needed to prevent it from being interpreted as a boolean operator.

For some simple cases it may be more convenient to use the "+" operator. For example:

- +cat white stray
- cat like:(white stray)

Both forms are equivalent and will return documents that must have "cat", but with any documents also containing "white" or "stray" ranked higher. Note: If there are not explicit " + " or " - " operators, the query terms will all be treated as if "" were written.

The `like` option can also be used in conjunction with the `minMatch` option to require at least a specified fraction of the optional terms to be present in documents. For example,

- minMatch:75% like: Four score "and" seven years ago our fathers brought forth
- minMatch:75 like: Four score "and" seven years ago our fathers brought forth
- minMatch:0.75 like: Four score "and" seven years ago our fathers brought forth
- minMatch:8 like: Four score "and" seven years ago our fathers brought forth
- minMatch:75%:(like: Four score "and" seven years ago our fathers brought forth)
- like:(Four score "and" seven years ago our fathers brought forth)~8
- like:(Four score "and" seven years ago our fathers brought forth)~75
- like:(Four score "and" seven years ago our fathers brought forth)~75%
- like:(Four score "and" seven years ago our fathers brought forth)~0.75

All nine forms are equivalent and will return documents that have at least 8 (75% of 10 is 7.5 which is rounded up to 8) of the listed terms.

Like Document Term Keyword Option

If the *like* term keyword option has a single term specified and that term has digits, or any punctuation, such as period, slash, colon, and so on, the term is assumed to be a *document id* and the most relevant terms will be extracted from that document and used as if they had been listed for the *like* term keyword option to boost relevancy for other documents containing those terms. This feature is sometimes referred to as "more like this" or "find similar" and is available in various commercial search engines.

A document ID is typically a web page URL, a file system path, a number, or some other special format, other than a term consisting of only letters, that uniquely identifies a given document. Most document IDs, including URLs, can be written directly, but the ID can be enclosed within quotes if it has any embedded spaces or to enhance readability. For example:

- Washington like:http://cnn.com -"New York"
- Washington like:"http://cnn.com" -"New York"

Both forms would select documents that contain "Washington" and do not contain "New York", with relevancy boosted by the most relevant terms contained in the web page at "http://cnn.com". This would find documents similar to the CNN web page, but requiring "Washington" and excluding "New York".

As a simple but reasonably detailed example, consider the following mini-documents in a Unix file system:

- /usr/home/jsmith/george.txt - George Washington
- /usr/home/jsmith/abe.txt - Abraham Lincoln
- /usr/home/jsmith/both.txt - George Washington and Abraham Lincoln

The following query would return george.txt and both.txt:

- like:/usr/home/jsmith/george.txt

That query is effectively the same as:

- like:(George Washington)

The following query would return all three documents:

- like:/usr/home/jsmith/both.txt

That query is effectively the same as:

- like:(George Washington Abraham Lincoln)

Note: Short words are ignored. The actual minimum word length is a configurable parameter.

The actual process of selecting the most relevant terms from the specified document is a bit more complex, but includes term frequency.

In addition, the terms are each given a calculated boost factor that corresponds to their calculated relevancy. The examples given here are simplified, but the actual queries include term weights based on frequency in the specified document.

The *like* option can be combined with the *minMatch* option to assure that only documents with some required percentage of terms are matched. For example, give these mini-documents:

- /usr/home/jsmith/alpha.txt - Alpha
- /usr/home/jsmith/beta.txt - Beta
- /usr/home/jsmith/gamma.txt - Gamma
- /usr/home/jsmith/alpha-beta.txt - Alpha Beta
- /usr/home/jsmith/beta-gamma.txt - Beta Gamma
- /usr/home/jsmith/alpha-gamma.txt - Alpha Gamma
- /usr/home/jsmith/all.txt - Alpha Beta Gamma

The following query would match all seven documents:

- like:"/usr/home/jsmith/all.txt"

The following query uses *minMatch* to select only those documents containing 66% or two-thirds of the relevant words extracted by the *like* option:

- like:"/usr/home/jsmith/all.txt"~2

- like:"/usr/home/jsmith/all.txt"~66%
- like:"/usr/home/jsmith/all.txt"~0.66

All three of those query forms are equivalent and will exclude the first three documents since they have too few of the optional terms.

The previous query is equivalent to this query:

- like:(Alpha Beta Gamma)~66%

Query Parser Customization

This functionality is not available with LucidWorks Search on AWS or Azure

The Lucid query parser offers a wide range of configuration settings, called *request parameters* that can be set in the Solr configuration XML file, `solrconfig.xml` (`solrconfig.xml` is specific to each collection. If using `collection1`, `solrconfig.xml` will be found in `$LWE_HOME/conf/solr/cores/collection1_0/conf`). After editing `solrconfig.xml`, LucidWorks Search should be restarted. On some Windows systems, it may be necessary to stop LucidWorks Search before editing any configuration file.

First, locate the "/lucid" *request handler*, which appears as follows:

```
<requestHandler class="solr.StandardRequestHandler" name="/lucid">
```

Next, locate the "defaults" entry, which appears as:

```
<lst name="defaults">
```

Not all of the configuration settings will be present in `solrconfig.xml`. If not present, the settings will default to internal default settings. In general, `solrconfig.xml` is used to *override* internal default settings.

Before adding an override setting you should scan the existing settings to see if there is already a setting that can be modified. If none is present, add a new entry for the setting as detailed below. The order of the settings does not matter, so new settings can simply be inserted after the "defaults" entry.

Each configuration setting entry has the following format:

```
<str name="sname">svalue</str>
```

where *sname* is the name of the setting, as detailed below, and *svalue* is the value of the setting. The setting value does not use quotes, even for string values.

A number of settings are Boolean on/off settings, where a value of *true* indicates that the setting is "on" or enabled, and *false* indicates that the setting is "off" or disabled.

q.alt : Alternate Query

The `q.alt` setting specifies a default query to be used if the input query passed to the Lucid query parser is empty. By default, this setting is `*:*`, which selects all documents, which is equivalent to placing this entry in the request "defaults" in `solrconfig.xml`:

```
<str name="q.alt">*:*</str>
```

To disable this behavior and simply select no documents to return no query results, use an entry as follows:

```
<str name="q.alt"></str>
```

leadWild : Enable Leading Wildcards

The `leadWild` setting controls whether leading wildcards are permitted in queries. By default, this setting is "on" or enabled, which is equivalent to placing this entry in the request "defaults" in `solrconfig.xml`:

```
<str name="leadWild">true</str>
```

To disable leading wildcards, turn this setting off, with an entry as follows:

```
<str name="leadWild">false</str>
```

maxQuery : Query Limits

The Lucid query parser defaults to handling queries of up to 64K (65,536) characters in length. This should be sufficient to support even the most demanding of applications. But, should even this not be sufficient, configuration settings in `solrconfig.xml` may be added or modified to override these limits. In other cases, it may be desirable to dramatically decrease these limits to prevent rogue users from overloading the query/search servers in high-volume applications.

Here are the four configuration parameters that control maximum query length and their default values:

- `maxQuery` = 65,536 – Maximum length of the source query string (64K).
- `maxTerms` = 20,000 – Maximum number of terms in the source query string.

- maxGenTerms = 100,000 – Maximum number of Lucene Query terms that will be generated.
- maxBooleanClauses = 100,000 – Maximum number of Lucene Boolean Clauses that can be generated for a single BooleanQuery object. This includes original source terms, plus relevance-boosting phrases that are automatically generated.

Those default settings do not normally appear in solrconfig.xml, but if they did they would appear as follows:

```
<str name="maxQuery">65536</str>
<str name="maxTerms">20000</str>
<str name="maxGenTerms">100000</str>
<str name="maxBooleanClauses">100000</str>
```

If you wish to reduce the query length limit for maximum throughput of "casual" queries, a query length of 1,000 and 200 terms might be appropriate, which would require entries as follows:

```
<str name="maxQuery">1000</str>
<str name="maxTerms">200</str>
```

nearSlop : Default Distance Limit for Proximity Operators

The nearSlop setting controls the default distance limit for the NEAR, BEFORE, and AFTER proximity operators. The internal default is 15, which is equivalent to placing this entry in the request "defaults" in solrconfig.xml:

```
<str name="nearSlop">15</str>
```

The default distance can be changed to 10, for example, with an entry as follows:

```
<str name="nearSlop">10</str>
```

notUp : Whether Upper Case is Required for the NOT Operator Keyword

The NOT keyword operator is a special case among the keyword operators since "not" is such a common word in natural language text and would be too easily confused with "not" as a keyword operator. For this reason, the NOT operator *must* be all upper case, unless the notUp request parameter is disabled to allow "not" as a lower case operator.

The `notUp` setting controls whether the `NOT` operator keyword must be all upper case. A setting of `true` means that the `NOT` keyword must be all upper case to be considered as an operator rather than as a simple text term. A setting of `false` means that the `NOT` operator keyword may be lower case or upper case, or even mixed case. The internal default is `true`, meaning that all upper case *is required* for the `NOT` operator keyword, which is equivalent to placing this entry in the request "defaults" in `solrconfig.xml`:

```
<str name="notUp">true</str>
```

Lower case or mixed case for the `NOT` operator keyword can be enabled with an entry as follows:

```
<str name="notUp">false</str>
```

opUp : Whether Upper Case is Required for Operator Keywords

The `opUp` setting controls whether operator keywords, such as `AND`, `OR`, and `NEAR`, must be upper case. A setting of `true` means the keywords must be all upper case to be considered as operators rather than simple text terms. A setting of `false` means that operator keywords may be lower case or upper case, or even mixed case. The internal default is `false`, meaning that upper case is not required, which is equivalent to placing this entry in the request "defaults" in `solrconfig.xml`:

```
<str name="opUp">false</str>
```

Operator keywords can be required to be all upper case with an entry as follows:

```
<str name="opUp">true</str>
```

The `NOT` keyword operator is a special case since "not" is such a common word in natural language text and would be too easily confused with "not" as a keyword operator. For this reason, the `NOT` operator *must* be all upper case, unless the separate parameter, `notUp`, is disabled to allow "not" as a lower case operator.

minMatch : Minimum Match of Optional Terms for Boolean Query

The `minMatch` configuration setting controls the minimum percentage of optional terms of a simple Boolean query that must match for a document to be selected. This configuration setting is used as the default unless the user explicitly uses the `minMatch` (or `atLeast`) keyword option or the tilde ('~') modifier on a simple Boolean query in the query string. A setting of `0` (the default) means that none of the optional terms is required for a document match. A value of `100` means that all optional terms must match. As a special case, any small percentage, such as `1`, means that at least one optional term must match in any simple Boolean query. The default value of `0` is equivalent to placing this entry in the request "defaults" in `solrconfig.xml`:

```
<str name="minMatch">0</str>
```

To assure that at least one optional term matches in every simple Boolean query, use an entry as follows:

```
<str name="minMatch">1</str>
```

To require half (50%) of the optional terms to match in simple Boolean queries, use an entry as follows:

```
<str name="minMatch">50</str>
```

Custom Connector Guide

This guide discusses how to build a custom connector with the LucidWorks Connector Framework.

A simple example crawler is provided in the `$LWE_HOME/app/examples/java` directory of a LucidWorks installation. That directory also contains the source code for the example crawler, which can be used as a basis for a custom crawler.

The guide contains the following sections:

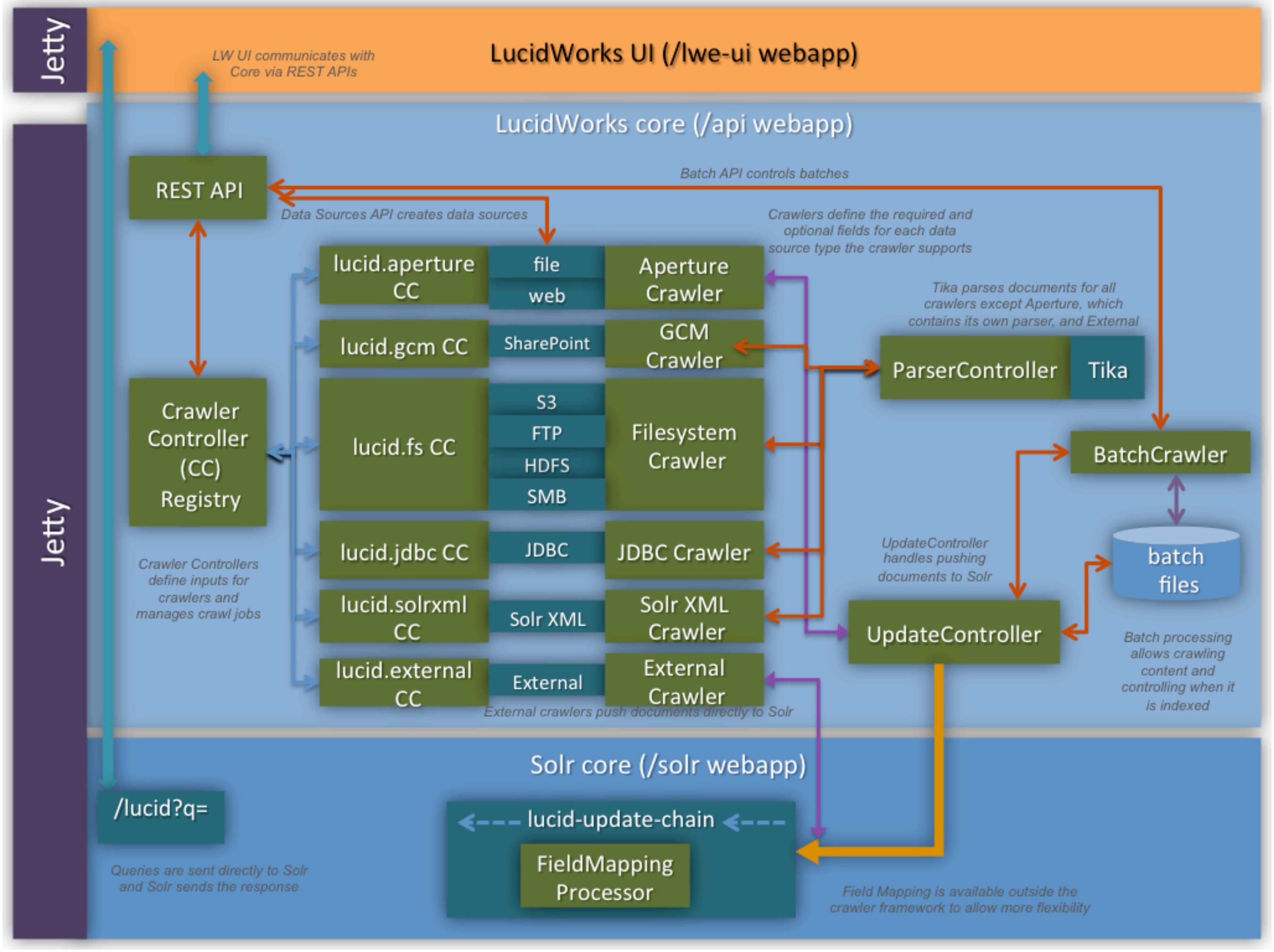
- Introduction to Lucid Connector Framework: Provides a technical overview of how connectors work in LucidWorks Search and an introduction to the various components of the Framework.
- How To Create A Connector: Provides detailed information about each component, including how to build a custom component and which parts of the example connector to reference while creating a connector.
- Integrating New Crawlers with LucidWorks: Once the custom connector is made, LucidWorks Search needs to be able to discover it; this section describes how to do that.

Introduction to Lucid Connector Framework

LucidWorks Search offers an open API for managing the process of content acquisition (crawling) from document repositories, and for adding new implementations for content acquisition (crawlers).

The main principle behind the design of this API was to isolate the core of LucidWorks Search both from the details of each crawler component implementation and the details of the indexing platform, and to allow for integration of externally-developed crawler components. Currently this API is in use and provides the integration for each crawler integrated with LucidWorks (whether developed by LucidWorks or as a 3rd party integration).

This graphic gives an overview of the Crawler architecture:


Jetty
LucidWorks UI (/lwe-ui webapp)
LW UI communicates with Core via REST APIs
LucidWorks core (/api webapp)
REST API
Batch API controls batches
Data Sources API creates data sources
Crawlers define the required and optional fields for each data source type the crawler supports
lucid.aperture CC
file
web
Aperture Crawler
Tika parses documents for all crawlers except Aperture, which contains its own parser, and External
lucid.gcm CC
SharePoint
GCM Crawler
ParserController
Tika
Crawler Controller (CC) Registry
lucid.fs CC
S3
FTP
HDFS
SMB
Filesystem Crawler
BatchCrawler
Jetty
Crawler Controllers define inputs for crawlers and manages crawl jobs
lucid.jdbc CC
JDBC
JDBC Crawler
UpdateController handles pushing documents to Solr
batch files
lucid.solrxml CC
Solr XML
Solr XML Crawler
UpdateController
Batch processing allows crawling content and controlling when it is indexed
lucid.external CC
External
External Crawler
External crawlers push documents directly to Solr
Solr core (/solr webapp)
/lucid?q=
lucid-update-chain
FieldMapping Processor
Queries are sent directly to Solr and Solr sends the response
Field Mapping is available outside the crawler framework to allow more flexibility

LucidWorks Crawler architecture

In general terms, each Crawler defines possible data source types and their parameters.

Each data source is defined by a DataSourceSpec, which lists all possible properties that a DataSource can take, their default values, and whether they are mandatory or optional. The DataSourceFactory for each CrawlerController defines the valid DataSourceSpec(s) for a particular crawler. The DataSourceSpec(s) for a crawler are also known as a type of data source, as a type must be unique for each CrawlerController. The DataSourceFactory may also contain validation rules for data source properties; for example, requiring that input to a property is a string instead of numeric.

The CrawlerController handles the scheduling and execution of crawl jobs. Each job has a unique identifier and the job definition is reusable. The job definition may include information to initiate a crawl at a specific time each day, for example. The status of each job is also managed by the CrawlerController with defined states such as RUNNING, FINISHED, STOPPED and others. Each crawl job definition also includes instructions for how to handle the output of the job.

All CrawlerControllers are created and managed by the CrawlerControllerRegistry.

Once data has been acquired from content sources, the CrawlProcessor defines how it is further processed. The ParserController is a document parsing and content extraction service and the UpdateProcessor represents the output for SolrInputDocument(s) to be indexed. The default LucidWorks CrawlProcessor currently uses Apache Tika v1.0 for content extraction and parsing from the raw data. The UpdateProcessor uses a SolrJ connection to the LucidWorks instance of Solr.

It's possible to handle the output of a crawl job as a batch, meaning that no parsing or indexing can take place (or a combination: no parsing but indexing or parsing but no indexing). The output of the job is stored for later processing - either to be parsed by a separate process or indexed at a more convenient time. Batches allow the crawl process to be split into three stages: fetching raw content; parsing content; and indexing content. A BatchManager handles these processes and stores the fetched content in a hierarchy of folders on the filesystem, which may consume a great deal of space depending on the content. Stored content is not automatically cleaned after indexing (if the content is ever indexed) to allow multiple indexing runs, so care must be taken to remove unwanted batches when they are no longer needed.

After raw documents have been fetched and their text and metadata extracted, an initial version of a SolrInputDocument is prepared. Each data source has a property called "mapping" which defines how to handle the incoming fields of each document. The mappings can specify a field to use as the unique key, map incoming fields to other previously defined fields, add fields to documents, or convert field types to other field types. The FieldMapping definitions are sent to Solr in JSON format to update the rules for the FieldMappingProcessor.

After FieldMapping has been completed, the documents are input to Solr and indexed.

How To Create A Connector

As described in the introduction, the Lucid Connector Framework is an open API for managing the process of content acquisition, known as crawling, from document repositories and for adding new implementations of content acquisition, referred to as crawlers.

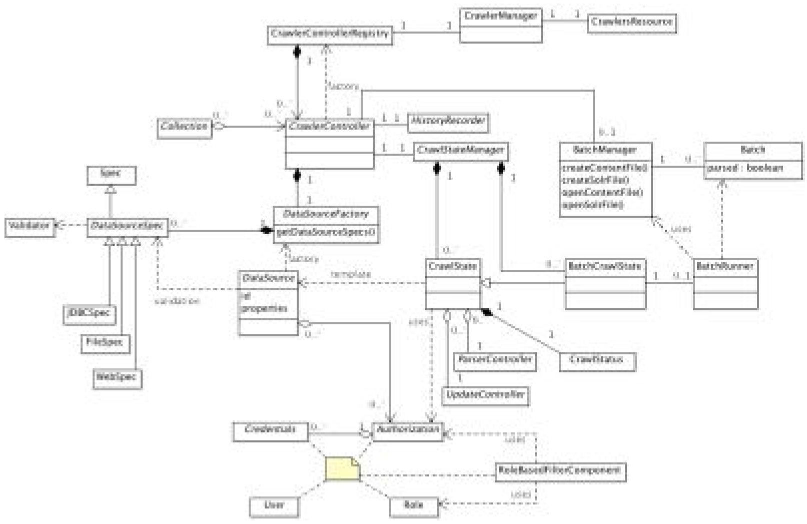
This section and the ones following it, describe how to create a custom connector. Each section will start with a technical overview of the component and then discuss the classes that are required to create a custom component for your own needs. An example connector is also provided and can be found in `$LWE_HOME/app/examples/java`.

The following sections will discuss each of the Connector Framework components:

- `CrawlerController` facade, responsible for defining and controlling the crawl jobs executed by a given crawler platform. `CrawlerController` instances are created by `CrawlerControllerRegistry`. This is the central class in the API, and users interact with crawler platforms via methods defined in this class.

- `DataSource` which describes in an abstract way the configuration parameters for accessing a content repository and the set of documents to retrieve. `DataSource`-s are created by a `DataSourceFactory` specific to a concrete crawler platform. Third-party crawlers should use a `CrawlDataSource` subclass that exposes more functionality and can be loaded from per-crawler classloaders.
- Closely related to `DataSource`-s are `DataSourceSpec`-s, which are descriptors that define what properties can be set, their default values and how the values set by a user can be validated.
- Crawl jobs run by `CrawlerController`-s are represented by `CrawlState`, and their status can be obtained from `CrawlStatus`.
- `CrawlProcessor` is an abstraction for processing the output of a crawl job. This in turn uses a `ParserController` and `UpdateController`, responsible for parsing and indexing the results of a crawl job.
- `BatchManager` handles batch jobs, which are crawl jobs that don't immediately send their output to a `CrawlProcessor` for parsing and indexing, but instead store the raw content for later processing (either parsing + indexing, or just indexing).
- `FieldMapping` and `FieldMappingUtil` constitute a metadata mapping facility to map metadata extracted from documents to the target index fields. This facility includes some rudimentary type conversion.

The following graphic shows the architecture of the components in detail:



Connector Framework Architecture

Example Crawler

A simple example crawler is provided in the `$LWE_HOME/app/examples/java` directory of a LucidWorks installation. That directory also contains the source code for the example crawler, which can be used as a basis for a custom crawler.

Custom Classes

To build a custom crawler you need to write the following classes. In each example, the crawler name should begin each class name; replace `crawler` with the crawler name when creating the files.

- `crawlerItemSpec`: Derived from `DataSourceSpec`. Contains crawler-specific properties
- `crawlerDataSource`: Derived from `CrawlDataSource`. Contains properties and methods for registration of the crawler.
- `crawlerDataSourceFactory`: Derived from `DataSourceFactory`. Registers the `ItemSpec`, `DataSource`, `CrawlerController`.
- `crawlerCrawlerController`: Derived from `CrawlerController`. Defines the startup, stop and reset of the crawler.
- `crawlerCrawlState`: Derived from `CrawlState`. Defines stop and start for a daemon-thread the crawler should run with.
- `crawlerCrawler`: Derived from `Runnable`. The traversal of the `DataSource` is done here.

DataSource and DataSourceSpec

Overview

The `DataSource` class (and `CrawlDataSource` subclass) is essentially a wrapper for a set of properties, with some specialized methods and constructors that enforce providing some details. `DataSourceFactory` (an abstract class, whose implementation is specific to a specific crawler platform) knows what kind of data sources are supported by the platform, and it also knows how to validate a set of provided properties to verify whether they can define a valid DataSource for this platform. For this purpose the `DataSourceFactory` also keeps a registry of `DataSourceSpec`-s. These "specs" list all possible properties that a DataSource can take, their default values and whether they are mandatory. `DataSourceSpec` also provides a limited conversion facility (casting) of input data to the formats and types expected by a corresponding DataSource (e.g., it's common for numeric parameters to be supplied by a UI as strings - the casting then uses Validator subclasses to convert such strings to a numeric format, so that other parts of the API will deal only with the expected types of the properties).

`DataSource` type is an arbitrary string identifier that must be unique in the scope of a given crawler platform. DataSource category is a purely informative string that may provide hints to the user about how to present documents retrieved from this source (e.g. files, database records, ...).

`DataSource` instances should be viewed as purely passive data containers. Any state related to crawl jobs should be kept in implementation-specific subclass of `CrawlState`.

The properties of a `DataSourceSpec` are those that are entered by a user when creating a new data source of this type (either through the UI or via the API). Careful consideration should be given to which properties should be required and each property should have as good a default as possible. Validators that match the expected type of the value input for a property should be called to catch configuration errors as early as possible (e.g., before they produce errors in the crawl). Several predefined validators are available which match many common types (such as int, float, URL, etc.) and they should be used where appropriate.

Lucid Imagination uses underscores for property names in each of our crawler implementations instead of CamelCase (so, a property is called "ignore_robots" instead of "ignoreRobots"). Following the same standard may help your custom crawlers appear to work the same as the included set of crawlers.

The properties are also used by the Admin UI to dynamically create an entry form for users to configure new data sources. The flexibility of this approach allows new crawlers to be added, or properties of a data source modified, without having to create or update forms for each data source type. There are some conventions the UI uses when reading the properties for a specific data source. First, property names are normalized and used as form labels. The normalization removes any underscores and the first word is capitalized. A property name such as "access_token" is transformed to display as "Access token". Property descriptions are used to display information to the user about what kind of information is expected for that attribute. Descriptions are optional, but if they are used, they should be kept short.

Custom Classes for This Component

In the example crawler provided with LucidWorks, you can find examples of customized classes for this component in `$LWE_HOME/app/examples/java/crawler/src/java/com/lucid/examples/crawl`.

To build a custom crawler you need to write the following classes for the DataSource component. In each example, the crawler name should begin each class name; replace `crawler` with the crawler name when creating the files.

- `crawlerItemSpec`: Derived from `DataSourceSpec`. Contains crawler-specific properties. The super-constructor from `DataSourceSpec` has to be called with the Category of the Crawler. If it is not defined, it can be defined in DataSourceSpec (enum Category and enum Type). This is not recommended because the LucidWorks Connector Framework has to be recompiled. Use `Category.Other` instead.
- `crawlerDataSource`: Derived from `CrawlDataSource`. Contains properties and methods for registration of the crawler. It is the super-constructor has to be called with the collection, the CrawlerType, and the Category, for example: `super("exchange", Category.Exchange.toString(), crawlerType, collection)`
- `crawlerDataSourceFactory`: Derived from `DataSourceFactory`. Registers the `ItemSpec`, `DataSource`, and `CrawlerController`. The `CrawlerController` has to be set as a parameter in the super-constructor. The `ItemSpec` and DataSource has to be registered, for example:

```
super(cc);
this.types.put("exchange", new ExchangeItemSpec());
this.impls.put("exchange", ExchangeDataSource.class);
```

CrawlerController, CrawlStatus and CrawlerControllerRegistry

Overview

The abstract class `CrawlerController` models interactions with a crawler implementation. `CrawlerController` instances are created and managed by a `CrawlerControllerRegistry`. Each crawler provided in a separate jar (including some built-in crawlers and all third-party crawlers) is loaded using its own classloader - this way crawler implementations are isolated and can use conflicting versions of dependencies.

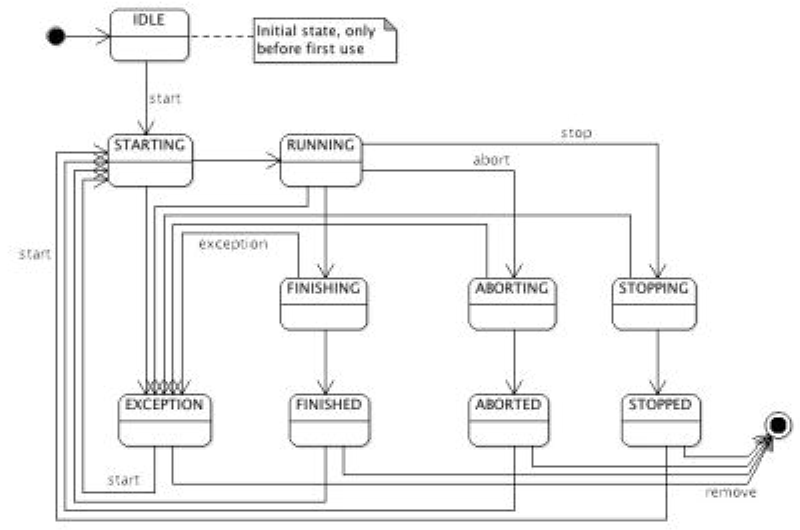
Although the singleton pattern for `CrawlerController`-s is not enforced, it is followed in practice through the use of `CrawlerControllerRegistry` (which is a singleton in LucidWorks Search).

A crawl job is the process of going to a document repository and retrieving documents.

The life-cycle of a crawl job consists of the following states:

- The crawl job is defined and registered with the crawler platform using `CrawlerController.defineJob(...)`. This is also a chance for the crawler platform to perform additional verification of the crawl parameters. A defined job gets a unique identifier, and the job definition with this identifier is reusable. An internal component `CrawlerController.jobStateMgr` manages this definition in memory. For crawler platforms that run in separate processes there may be a need to synchronize this internal job status with the external process; the crawler API makes no such provisions, since the details of this process are implementation-specific.
- The crawl job is started using `CrawlerController.startJob(...)`. The job itself is then being executed asynchronously (in a separate thread). This call should return quickly and must not block for the duration of the crawl job. In the current API there is provision only for one running crawl job per job definition.
- A transitory state is associated with the process of starting a job called "STARTING", which is expected to last relatively shortly, after which the job transitions to a "RUNNING" state or to one of the final failure states (see below). Most of the work for the crawl job is expected to take place in the RUNNING state. When the work is finished in an orderly manner, the job transitions to a FINISHING state, where the necessary commits and cleanups are performed, and then finally transitions to a FINISHED state.
- A running crawl job can be stopped or aborted. When a crawl job is stopped, the CrawlerController will make an attempt to preserve as much of the partial crawl results as possible, and stops the job in an orderly manner. When the crawl job is aborted there are no such guarantees; however, partial results may still become visible and committed to the index. If this is not desireable the `CrawlerController` implementation may track documents by adding a run identifier (batch_id) and then issue a delete request with this batch_id. There are two transitory states associated with these actions: STOPPING and ABORTING. During these states it's expected that the controller will perform necessary cleanups. Final states after these actions are STOPPED and ABORTED, respectively.
- In case of a non-recoverable error the job goes into a final EXCEPTION state.

The following shows the various crawl job states and how they interact with one another:



Crawl Job States

Crawl job status can be retrieved using `CrawlerController.getStatus(...)` or all running and recently finished crawl jobs can be listed using `CrawlerController.listJobs(...)`. This list is maintained in memory, so it's cleared on restart. There is also a persistent job history that is maintained with `CrawlerController.getHistoryRecorder().record(crawlStatus)`.

Crawler platforms that want to support batch operations should return a non-null implementation of `BatchManager`, such as the provided SimpleBatchManager that stores intermediate crawl results in local files or another method.

Custom Classes for This Component

In the example crawler provided with LucidWorks, you can find examples of customized classes for this component in `$LWE_HOME/app/examples/java/crawler/src/java/com/lucid/examples/crawl`.

To build a custom crawler you need to write the following classes for the `CrawlerController` component. In each example, the crawler name should begin each class name; replace `crawler` with the crawler name when creating the files.

- `crawlerCrawlerController`: Derived from `CrawlerController`. Defines the startup, stop and reset of the crawler.
- `crawlerCrawlState`: Derived from `CrawlState`. Defines stop and start for a daemon-thread the crawler should run with.

For every derived class there are some methods that can be overridden:

- `crawlerCrawlerController`:
  - `reset(String collection, String dataSourceId)`: tells the crawler to reset the used data source, to clean up timestamps used to reinitialize the crawl state etc.
  - `resetAll(String collection)`: tells the crawler to reset all data sources
  - `CrawlId defineJob(DataSource ds, CrawlProcessor processor)`: Initializes the `CrawlState`. The `DataSource` and the `CrawlProcessor` have to be registered like this:

```
ExchangeCrawlState state = new ExchangeCrawlState();
state.init(ds, processor, this.historyRecorder);
this.jobStateMgr.add(state);
```

  - `startJob(CrawlId descrId)`: starts the job with `CrawlState.start()`
  - `stopJob(CrawlId jobId)`: stops the job
  - `abortJob(CrawlId jobId)`: aborts the job (can be the same as stopJob)
- `crawlerCrawlState`:
  - `start()`: starts a Crawler over a daemon-thread

Crawler

Overview

The `Crawler` does the work of traversing the Data Source and collecting the data for input.

Custom Classes for this Component

In the example crawler provided with LucidWorks, you can find examples of customized classes for this component in `$LWE_HOME/app/examples/java/crawler/src/java/com/lucid/examples/crawl`.

To build a custom crawler you need to write the following classes for the Crawler component. The crawler name should begin each class name; replace `crawler` with the crawler name when creating the files.

- `crawlerCrawler`: Derived from `Runnable`. The traversal of the `DataSource` is done here.

All classes have to override some methods from the interfaces they are derived from or call a method of the super class to get a successful registration in the LucidWorks framework.

- `crawlerCrawler`:
  - `run()`: Overridden by the interface `Runnable`. Here the traversal of the data source is done.
  - `stop()`: Overridden by the interface `Runnable`.

CrawlProcessor, ParserController and UpdateController

Overview

Each crawl job definition should specify a sink for the output of the crawl job. A null value may be provided to mean the default `CrawlProcessor` implementation that uses the `TikaParserController` and `UpdateController` depending on the setting of the datasource (see below). This default implementation can be also configured to persist raw results of the crawl job as a batch when the "caching" property is true for a datasource. Document parsing and indexing can also be turned off (and the data stored in a batch) when a "parsing" property of a data source is set to false.

The `CrawlProcessor` exposes a minimal API to simply consume the raw output documents plus the protocol-level metadata. Alternatively, it can consume a `SolrInputDocument` if the parsing process was already performed. Usually only one of the methods is called; specific implementations of `CrawlProcessor`-s pass data between these methods as automatically as necessary.

The default `CrawlProcessor` that comes with LucidWorks (the one that is instantiated when no `CrawlProcessor` is specified) uses internally two other abstractions:

- `ParserController` represents a document parsing and content extraction service. The default implementation of this abstract class in LucidWorks is called `TikaParserController` and uses Apache Tika v1.0 for content and metadata extraction.
- `UpdateController` represents the output for `SolrInputDocument`-s to be indexed. The default implementation of this component obtained from `UpdateController.create(...)` uses a SolrJ connection to the Lucidworks instance of Solr as output when the "indexing" property is set to true, otherwise it stores parsed documents in batch data.

Custom Classes for This Component

None required, unless the default `CrawlProcessor` is not sufficient.

BatchManager

Overview

`BatchManager` is a component that is responsible for persisting and managing intermediate crawl results. This allows the crawling process to be split into up to three stages:

- fetching raw content from remote repositories
- parsing content, as well as text and metadata extraction
- indexing the extracted data as SolrInputDocuments

The advantage of this functionality is that the fetching of the data is usually the most costly step in the crawling pipeline, and it's sometimes better to execute the parsing and indexing at a different time than the fetching. It can also be used to re-do some of the steps e.g., after fixing configuration errors (either in field mapping or in the `schema.xml` file). The disadvantage is that it complicates the data flow and consumes additional disk space, but that may be an acceptable tradeoff.

LucidWorks comes with an implementation of this API called `SimpleBatchManager`. This implementation stores batch data in a hierarchy of folders that in turn contain record-oriented files. `SimpleBatchManager` creates separate folder hierarchies for each crawler, each crawl job, and each crawl job run that resulted in some batch data. The files are the following:

- `batch.status` - describes the status of the batch, e.g. how many raw documents are present, how many parsed documents, timestamp, etc.
- `content.raw` - contains the raw content of retrieved documents together with protocol-level metadata.
- `solr.json` - contains SolrInputDocument-s ready for indexing, in JSON format.

A `CrawlerController` implementation supports batch operations when it provides a non-null instance of a `BatchManager`. The following operations can be performed on batch data:

- Batch data can be created by setting appropriate options in `DataSource`-s, when the default implementations of `CrawlProcessor`, `ParserController` and `UpdateController` are used. There is also a lower-level API available for writing individual records to a specified batch, see the `ContentFileWriter` and `SolrFileWriter` javadoc for more details.
- `BatchManager.listBatchStatuses(...)` (or a REST API under `/api/collections/<collection>/batches`) can be used to retrieve a list of available batch data sets.
- A batch processing job can be started with `CrawlerController.startBatchJob(...)`, where the user can specify the output of the batch processing, whether the content should be parsed (or re-parsed) and/or indexed, using the supplied `CrawlProcessor` instance or a default one when `null` is provided.
- running batch jobs can be listed with `CrawlerController.listBatchJobs()`, or otherwise controlled in the same way as regular crawl jobs, i.e. stopped or aborted. They also undergo the same state changes as regular crawl jobs.

Obsolete or no longer needed batch data can be deleted using `BatchManager.deleteBatch(...)` or using the bulk delete `BatchManager.deleteBatches(...)`. Batch data is not deleted by default, so must be managed to ensure the batches do not consume too much disk space.

Custom Classes for This Component

None. Customizations to support batch operations are defined in the custom `CrawlerController`.

FieldMapping and FieldMappingUtil

Overview

After the raw documents are parsed and their text and metadata are extracted, an initial version of `SolrInputDocument` is prepared (this usually takes place somewhere in a `CrawlProcessor` or a `ParserController` that it uses). Since the metadata names in documents coming from various sources can be pretty arbitrary it is necessary to normalize their values and to map their names to Solr fields valid for the current index schema. This is the role of the `FieldMapping` and the `FieldMappingUtil` helper class.

Each `DataSource` has a property "mapping" that contains an instance of `FieldMapping` (if there was none specified on DataSource creation then a default one with the default mappings will be provided). You can examine the details of the default field mapping by looking through the data source records serialized to `collections.yml`. These mappings can be also edited by hand (remember to preserve the YAML format of the file).

Field mappings are specified per data source, and then passed to `UpdateController`. The process of field mapping is performed by the `UpdateController` so usually it should not be invoked explicitly. `UpdateController` implementations may further verify the mappings using the current Solr schema so that the mappings produce valid fields.

The mappings consist of the following main areas:

- `uniqueKey` - this property specifies the name of the uniqueKey in the Solr schema. This value is verfied with the current schema when a new crawl job is defined.
- `mappings` - this is a map of source metadata names to target field names. Source names are case-insensitive. A value of null means that this metadata should be discarded. See below for the details of the mapping algorithm.
- `literals` - this is a map of key/value pairs that define fields to be added to every document.
- `types` - defines any special field types if a conversion is necessary from the default STRING type. Currently recognized types are STRING, INT, LONG, FLOAT, DOUBLE and DATE. If a type is not specified then STRING is assumed.
- `multiVal` - this is a map of field names and boolean values. True means that the target index field with this name supports multiple values. False (or a missing key/value) means that the field is single-valued.
- `dynamicField` - this property specifies a prefix for dynamic fields if a more specific mapping is missing. See also below for the details of the mapping algorithm.
- `defaultField` - this property specifies a name of the default field if a more specific mapping is missing. See also below for the details of the mapping algorithm.
- `datasourceField` - this property specifies a prefix of the fields that preserve data source id, data source type and data source display name.
- `lucidworks_fields` - this boolean property (defaults to true) indicates that LucidWorks-specific fields should be automatically added to incoming documents (such as data_source id, data_source_name and data_source_type).

Field mapping is usually created and initialized in DataSourceFactory.create(...) method when the datasource is initially created. During job startup this mapping is passed to `UpdateController`, which can optionally verify field mappings with the current index schema for the collection specified in the data source. If a field is missing in the schema then it's mapped to null (i.e., discarded). The arity and the type of the field is checked and set appropriately. This process is repeated each time `CrawlerController.startJob(...)` is called.

The process of mapping source metadata names to the target field names works like this (note: this process is already encapsulated in one of the existing `UpdateController` implementations):

- first a case-insensitive match is tried with the source names present in the "mappings". If a value is found then it's returned (a value of null should be interpreted as "discard").
- then if `dynamicField` is non-null the source name is converted to a dynamic field name like this: to the value of `dynamicField` an underscore is appended, and then an escaped version of the source name is appended (the escaping replaces any non-word character with an underscore).
- then if `defaultField` is non-null then the value of `defaultField` is returned
- finally, the source name is returned.

Internally, this process uses a helper class `FieldMappingUtil`. This class contains methods to:

- initialize field mappings with values suitable for Aperture or Tika parsing
- verify the mapping with the current index schema
- normalize fields - this normalization step should always be performed to make sure that the SolrInputDocument instances contain only fields valid for the current schema, with correct multiplicity and correct type. The normalization works like this:
    - if a field is defined as type DATE then the value is checked - if it's an instance of `java.util.Date` then it's left unchanged, otherwise the string representation of the value is parsed using Solr's `DateUtil` to obtain a valid `Date` instance. In case of parsing errors the offending value is discarded.
    - if a field is defined as single-valued but multiple values are present then:
        - if a field is type DATE only the first value is retained, all other values are discarded
        - otherwise a set of unique string representations of values is concatenated using single space character, and the original multiple values are replaced with this single concatenated value.
    - there is also some other special treatment for the "mimeType" field to avoid common Tika and Aperture parsing errors.
- addLucidworksFields - this method ensures that some fields necessary for the LucidWorks UI are populated.

Custom Classes for This Component

None necessary or required.

Integrating New Crawlers with LucidWorks

Register the Crawler

Once a new crawler has been created, it needs to be registered with the `CrawlerControllerRegistry` in order to work with LucidWorks. In simple terms the way to do this is to follow these steps:

1. Create a MANIFEST.MF file for the plugin `.jar` with the required properties.
2. Place the .jar in the `$LWE_HOME/app/crawlers` directory.
3. Restart LucidWorks.

Create MANIFEST.MF File

LucidWorks requires the following entries in the crawler plugin's jar `META-INF/MANIFEST.MF` file. These properties may be placed either in the main section of the manifest, or in multiple sections (e.g., one section per crawler implementation):

- Crawler-Alias: (required) this is a symbolic name under which this crawler implementation will be known to LucidWorks. For example, the built-in Aperture crawler is registered under alias "lucid.aperture". Implementors should pick a meaningful name that is unique.

- Crawler-Class: (required) this is a fully-qualified class name of the CrawlerController implementation. For example, the built-in Aperture crawler's class is "com.lucid.crawl.aperture.ApertureCrawlerController".

- Crawler-Exclude: (optional) this is a whitespace-separated list of packages and fully-qualified class names that are excluded from loading using this class loader, instead their look-up and loading will be delegated to the parent classloader. Inn some cases nested jars may provide classes that conflict with other classes loaded from the parent classloader. Names of packages and classes on this list are treated as plain string prefixes and regular expressions are not supported.

An example MANIFEST.MF file for the included Aperture-based crawler looks like this:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.2
Created-By: 1.6.0_29-b11-402-11M3527 (Apple Inc.)
Crawler-Alias: lucid.aperture
Crawler-Class: com.lucid.crawl.aperture.ApertureCrawlerController
Crawler-Exclude: javax.xml.namespace
```

Loading the JAR

Each crawler class is loaded in a separate classloader, together with its dependencies. In case of some of the built-in crawlers (such as the JDBC crawler), which are distributed as a part of the core platform, this process is simplified to use the default class loader.

In all other cases crawler controller implementations are loaded by `CrawlerControllerRegistry` from JAR files, typically found in `$LWE_HOME/app/crawlers`. These jar files contain both the main `CrawlerController` implementation class, all other related classes (such as DataSourceFactory subclass), and may also contain nested jar-s with dependencies (libraries) used by the implementation. A special class loader is used to load these classes, which unlike the default classloader:

- can discover and load classes from nested jar files,
- prefers classes found in the crawler jar over classes found in the parent classloader. This means that you can implement crawlers that use different, possibly mutually conflicting versions of dependencies,
- processes the crawler jar's META-INF/MANIFEST.MF file looking for specific entries, and initializes the crawler plugin.

This process is executed during Lucidworks start-up as a part of `CrawlerControllerRegistry` initialization. This means that it's sufficient to just put a crawler plugin jar in `$LWE_HOME/app/crawlers` for LucidWorks to discover it and initialize it.

How the Admin UI Reads Crawlers

The LucidWorks Admin UI has been designed to dynamically read available crawler and data source types and display the list based on the currently enabled crawlers. When a specific data source type has been selected by the user, the UI also dynamically draws the screen with the latest available properties. So, once a new crawler is completed and properly registered (as above), then it's enough to restart LucidWorks to see the data source in the UI.

There are some conventions the UI uses when reading the properties for a specific data source. First, property names are normalized and used as form labels. The normalization removes any underscores and the first word is capitalized. A property name such as "access_token" is transformed to display as "Access token". Property descriptions are used to display information to the user about what kind of information is expected for that attribute. Descriptions are optional, but if they are used, they should be kept short.

Glossary of Terms

Where possible, terms are linked to relevant parts of the documentation for more information.

Jump to a letter:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

Alerts
An alert allows a user to save searches. There are two types: *active*, which will send notifications when new results are found, and *passive*, which do not send notifications.

Auto-Complete
A way to provide users suggestions for possible matching queries before they have finished typing. In LucidWorks Search, this relies on an index of terms to be created on a regular basis by scheduling it as an activity.

B

Boolean Operators
These control the inclusion or exclusion of keywords in a query by using operators such as AND, OR, and NOT.

C

Click Scoring Relevance Framework
A method of changing the relevance ranking of a document based on the number of times other users have clicked on the same document.

Collection
One or more documents grouped together for the purposes of searching. See also Document.

Component
A part of LucidWorks Search that has been designed to stand alone or can be run independently from other components. LucidWorks Search has three main components: *LWE-Core*, which runs Solr, indexing, and other critical application functions, *LWE-Connectors*, which handles all crawling activities, and *LWE-UI*, which runs the Administrative UI, the front-end search interface, and the alerting functionality.

Connector
A connector is a program or piece of code that allows a connection to be made to a data source and content to be extracted from it.

Crawler
Also known as a "spider", this is a program that is able to retrieve documents internal or external servers.

D

Data Source
Defines the metadata required to connect to a location containing content to be indexed. It could be a file system path, a Web URL, a JDBC connection, or some other set of values.

Distributed Index
A distributed index is one where the search index for a collection is spread across more than one shard.

Distributed Search
Distributed search is one where queries are processed across more than one shard.

Document
One or more Fields. See also Field.

F

Field
The content to be indexed/searched along with metadata defining how the content should be processed by LucidWorks Search.

I

Inverse Document Frequency (IDF)
A measure of the general importance of a term. It is calculated as the number of total Documents divided by the number of Documents that a particular word occurs in the collection. See http://en.wikipedia.org/wiki/Tf-idf and http://lucene.apache.org/core/old_versioned_docs/versions/3_5_0/scoring.html for more info on TF-IDF based scoring and Lucene scoring in particular. See also Term Frequency.

Inverted Index
A way of creating a searchable index that lists every word and the documents that contain those words, similar to an index in the back of a book which lists words and the pages on which they can be found. When performing keyword searches, this method is considered more efficient than the alternative, which would be to create a list of documents paired with every word used in each document. Since users search using terms they expect to be in documents, finding the term before the document saves processing resources and time.

M

Metadata
Literally, *data about data*. Metadata is information about a document, such as it's title, author, or location.

N

Natural Language Query
A search that is entered as a user would normally speak or write, as in, "What is aspirin?"

Q

Query Parser
A query parser processes the terms entered by a user.

R

Recall
The ability of a search engine to retrieve *all* of the possible matches to a user's query.

Relevance
The appropriateness of a document to the search conducted by the user.

Replication
A method of copying a master index from one server to one or more "slave" or "child" servers. In LucidWorks Search, the master continues to manage updates to the index, while queries are handled by the slaves. This approach enables LucidWorks Search to properly manage query load and ensure responsiveness.

REST API
An alternative way of controlling LucidWorks Search without accessing the user interface.

S

Shard
A method of partitioning a database or search engine to maximize performance and efficiency.

SolrCloud
Ongoing work within the Solr community to improve Solr's ability to operate in a cloud environment.

Solr Schema (schema.xml)
The Apache Solr index schema. The schema defines the fields to be indexed and the type for the field (text, integers, etc.) The schema is stored in schema.xml and is located in the Solr home conf directory.

Solr Config (solrconfig.xml)
The Apache Solr configuration file. Defines indexing options, RequestHandlers, highlighting, spellchecking and various other configurations. The file, solrconfig.xml is located in the Solr home conf directory.

Spell Check
The ability to suggest alternative spellings of search terms to a user, as a check against spelling errors causing few or zero results. In LucidWorks Search, effective spell checking requires an index to be built on a regular basis by scheduling it as an activity.

Stopwords
Generally, words that have little meaning to a user's search but which may have been entered as part of a natural language query. Stopwords are generally very small pronouns, conjunctions and prepositions (such as, "the", "with", or "and")

Synonyms
Synonyms generally are terms which are near to each other in meaning and may substitute for one another. In a search engine implementation, synonyms may be abbreviations as well as words, or terms that are not consistently hyphenated. Examples of synonyms in this context would be "Inc." and "Incorporated" or "iPod" and "i-pod".

T

Term Frequency
The number of times a word occurs in a given document. See http://en.wikipedia.org/wiki/Tf-idf and http://lucene.apache.org/java/2_3_2/scoring.html for more info on TF-IDF based scoring and Lucene scoring in particular.
See also Inverse Document Frequency (IDF).

W

Wildcard
A wildcard allows a substitution of one or more letters of a word to account for possible variations in spelling or tenses. In LucidWorks Search, there are two ways to use them. One is to use an asterisk (*) at the end of a term to find all documents that contain words that start with that pattern. For example, `paint*` would find `paint`, `painter` and `painting`. A second way is to use a question mark (?) in the middle of a term to substitute for one character in that term. Such as, `c?t` would find `cat`, `cot` and `cut`. It's also possible to use wildcards at the start of a term in the same way - either to replace a single letter (using the ? symbol) or to find documents that contain words that end with a pattern using a *. For example, `*sphere` would find `ecosphere` and `stratosphere`.

About LucidWorks

LucidWorks (formerly known as Lucid Imagination) is the trusted name in Search, Discovery and Analytics, delivering the only enterprise-grade embedded search development solution built on the power of the Apache Lucene/Solr open source search project. Founded in 2008, the company initially provided support, consulting services, documentation and training for the Apache Lucene/Solr open source search project.

Within a few years, the LucidWorks team realized the need to add value to the open source search platform by developing an extensive layer of services which made Lucene/Solr secure and easier to use and manage. The company shipped the first version of its flagship product, LucidWorks Search, in 2011, followed by LucidWorks Big Data in May 2012. LucidWorks continues to offer support, documentation, consulting services and training products for Lucene/Solr.

LucidWorks remains committed to giving back to the Apache Lucene/Solr community. Out of the 37 Core Committers to the Apache Lucene/Solr project, 9 individuals work for LucidWorks, making the company the largest supporter of open source search in the industry. Further, LucidWorks hosts the Lucene Revolution, a conference dedicated to sharing ideas and promoting the Apache Lucene/Solr open source search project.

For more information on product and support options for LucidWorks Search, please write to: sales@lucidworks.com or visit our website. Support inquiries can be submitted to our Support group.

LucidWorks™

LucidWorks
3800 Bridge Parkway, Suite 101
Redwood City, CA 94065

Tel: 650.353.4057
Fax: 650.525.1365