lucidworks enterprise

DOCUMENTATION

lucid IMAGINATION

LucidWorks Enterprise Documentation

The LucidWorks Documentation is organized into several guides that cover all aspects of using and implementing a search application with LucidWorks Enterprise and LucidWorks Cloud.

Introduction

- How indexes are created
- How queries are handled

Installation & Upgrade Guide

- Installing a new LucidWorks Enterprise server
- System Directories and Logs
- Migrating configuration and index data to a new installation

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

LucidWorks Enterprise UI Guide

Use the built-in user interfaces to:

- Set up data sources and schedule crawls
- Configure fields and field types
- Enable user access
- Search for indexed documents

LucidWorks REST API Reference

Get programmatic access to:

- Configure data sources and administer crawls
- Set system settings
- Manage fields, field types, and collections
- Example clients

Lucid Query Parser

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

About Lucid Imagination

Lucid Imagination is the first commercial company exclusively dedicated to Apache Lucene/Solr open-source technology. As an active participant in the enormous community using Lucene and Solr, Lucid Imagination free software for developers, documentation, commercial-grade support, high-level consulting, and comprehensive training. Customers include AT&T, Sears, Ford, Verizon, Cisco, Zappos, Raytheon, The Guardian, The Smithsonian Institution, Salesforce.com, The MotleyFool, Macy's, Qualcomm, Taser, eHarmony, and many other household names around the world. The company's web site serves as a knowledge portal for the Lucene and Solr community, with information and resources to help developers build and deploy Lucene-based solutions in a more efficient and cost-effective manner. The Lucid Imagination founding team includes several key contributors and committers to Lucene and Solr, as well as experts in enterprise search application development.

For more information on product and support options for LucidWorks Enterprise, please write to: sales@lucidimagination.com or visit our website. Support inquiries can be submitted via our LucidWorks Enterprise forums.

Lucid Imagination
3800 Bridge Parkway, Suite 101
Redwood City, CA 94065

Tel: 650.353.4057
Fax: 650.525.1365

How to Use this Documentation

Audience and Scope

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

While LucidWorks Enterprise 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 Enterprise 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 Enterprise is identical to Solr or Lucene. One important note to remember is that LucidWorks Enterprise 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 Enterprise.

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 Enterprise.
Tip	Notes with a green background are Helpful Tips.
Warning	Notes with a red background are warning messages.

API Conventions

Parameters

Parameters shown in paths that need to be modified for your installation and specific configuration are indicated in italics.

For example, getting the status 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.
```

Getting Support

With the download of LucidWorks Enterprise, access is provided to online forums found at http://forums.lucidimagination.com/. Premium support is available with a subscription.

Introduction

This section will introduce you to basic concepts of LucidWorks Enterprise and search engine deployment. It includes the following topics:

Features of LucidWorks Enterprise gives an overview of the features provided by LucidWorks Enterprise over and above the core features of Apache Solr.

How Search Engines Work describes the processes used by search engines to allow people to submit queries and see links to documents in return.

How LucidWorks Enterprise Works gives an overview of LucidWorks Enterprise and how it was built on Solr and Lucene.

Indexing Documents gives a conceptual overview of the indexing process.

Managing Indexes and Fields gives a conceptual overview of document parsing and field management.

Managing Queries gives a conceptual overview of query processing and management.

Features of LucidWorks Enterprise

Because LucidWorks Enterprise is Apache Solr plus enhancements, it provides all of the benefits of Solr, along with the backing of a company staffed by many of the developers and maintainers of both Solr and Lucene.

Powerful Search
• Powered by Apache Lucene/Solr: Apache Lucene and Apache Solr are fast, efficient, and backed by a thriving community. LucidWorks Enterprise includes a complete Apache Solr release with all features, including parsers, stemmers, tokenizers, and so on. You do not need to be familiar with Solr in order to build search applications with LucidWorks Enterprise. • Near Real-Time Search: LucidWorks Enterprise allows you to update and delete index data in segments as it becomes available, allowing your users to search in near real time. • Scalable: Distributed search and distributed indexing allow for easily scaling out. • Replication: For high-traffic sites, replication can be employed to efficiently copy indexes to worker nodes configured behind a load balancer. • Cloud Connectors: Built-in support for Hadoop and Amazon EC2 S3 ensures robust, efficient search for data in the cloud. • REST API: LucidWorks Enterprise features can be easily integrated with existing enterprise infrastructure with a set of APIs that do everything the user interface can do.

Simple Administration
• Easy Installation: LucidWorks Enterprise comes with an easy-to-use installer, which can also be used to generate an automated installation script for playback on slave nodes in multi-server configurations. • Administration UI: A robust interface provides access to the most common administrative tasks, including configuration of crawlers and index fields, user management, and user experience options. System metrics also show the status of active processes and user search experience. • Data Connectors: LucidWorks Enterprise can index and search databases, file systems, Web sites, and Microsoft SharePoint repositories without additional programming. • Support for Popular File Formats: Support for many different filetypes is available, including Microsoft Office, Adobe PDF, and other common file formats. Additional options for file readers and document filters are also available. • Security: LucidWorks Enterprise is LDAP-aware, so search application developers can create document level security and admin function security in accordance with local security policies. Lucid Imagination has also added enhancements for optional end-to-end SSL encryption.

Advanced User Experience
• Query Parsing Enhancements: LucidWorks Enterprise enables a more resilient, richer user query experience. New query operators and better stop word and synonym handling together provide a simpler, more forgiving and intuitive end user search experience, improving the odds of turning user inputs into quality results. • Click Scoring Relevance Framework: Click Scoring is a suite of tools for adjusting scoring and ranking of documents based on analysis of historical user click data. LucidWorks Enterprise tracks which documents were selected by users, allowing manual and automated boosting based on the popularity of a given document. • User Alerts: To keep users up to date as new search results are available, LucidWorks Enterprise alerts enables automated notification. Any valid search query can form the basis for an alert; end users can select, define, and manage their own queries. • Integrated Spell Checking: LucidWorks Enterprise supports Lucene's spelling checker natively. • Integrated Auto-complete: As users type in partial queries, LucidWorks Enterprise will suggest completed search terms, so users can quickly complete queries or choose from frequently occurring query terms.

LucidWorks Enteprise also includes technical support from Lucid Imagination. Our engineers have extensive hands-on

knowledge of both Apache Solr and LucidWorks Enterprise and can answer questions and help you resolve problems. In addition, Lucid Imagination offers consulting services designed to help organizations better understand and implement search efficiently and cost-effectively. Contact your Lucid Imagination sales representative for details.

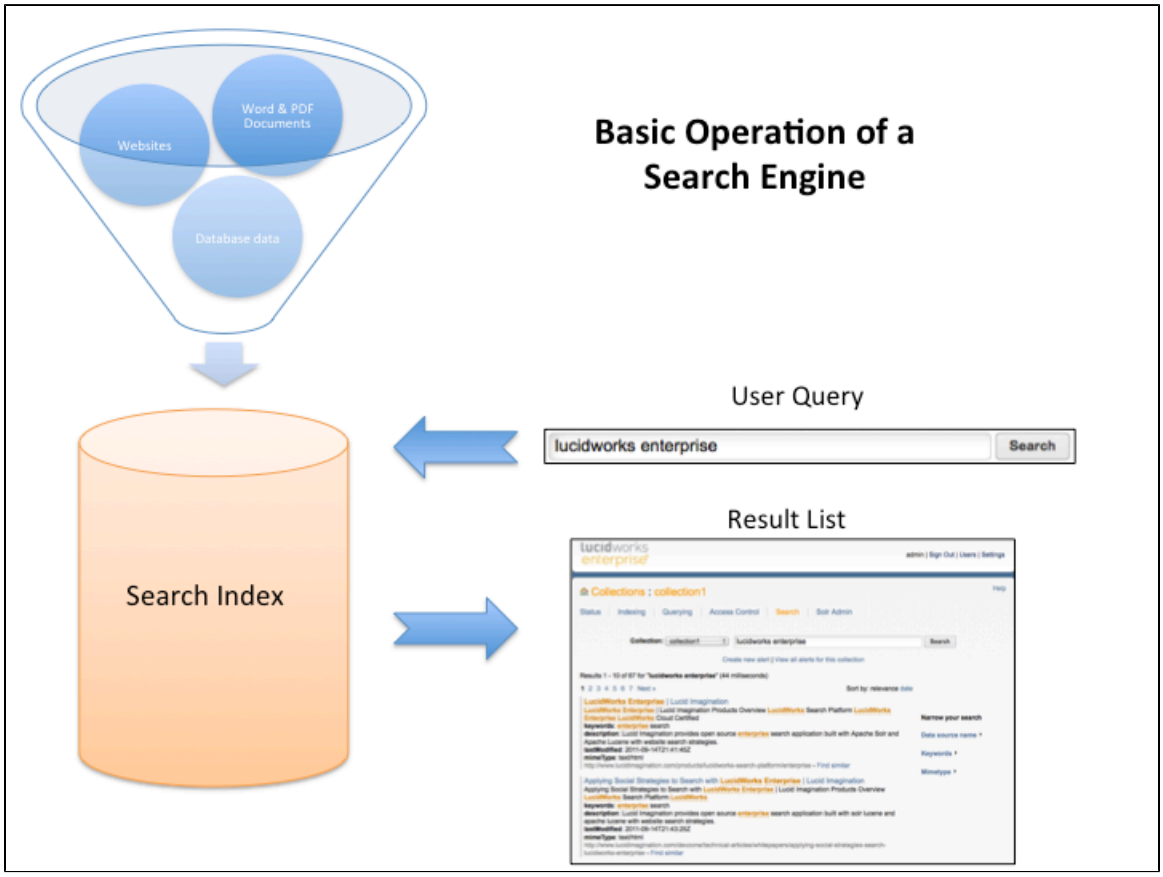
How Search Engines Work

Basics

In its simplest form, a search engine is an application that enables a user to query a data set for information. Most people are familiar with search engines that search the internet, but search engines also have many and varied applications inside the enterprise. 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. That means you need a separate search engine for internal use.

In LucidWorks Enterprise, each unit of text to be searchable 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.

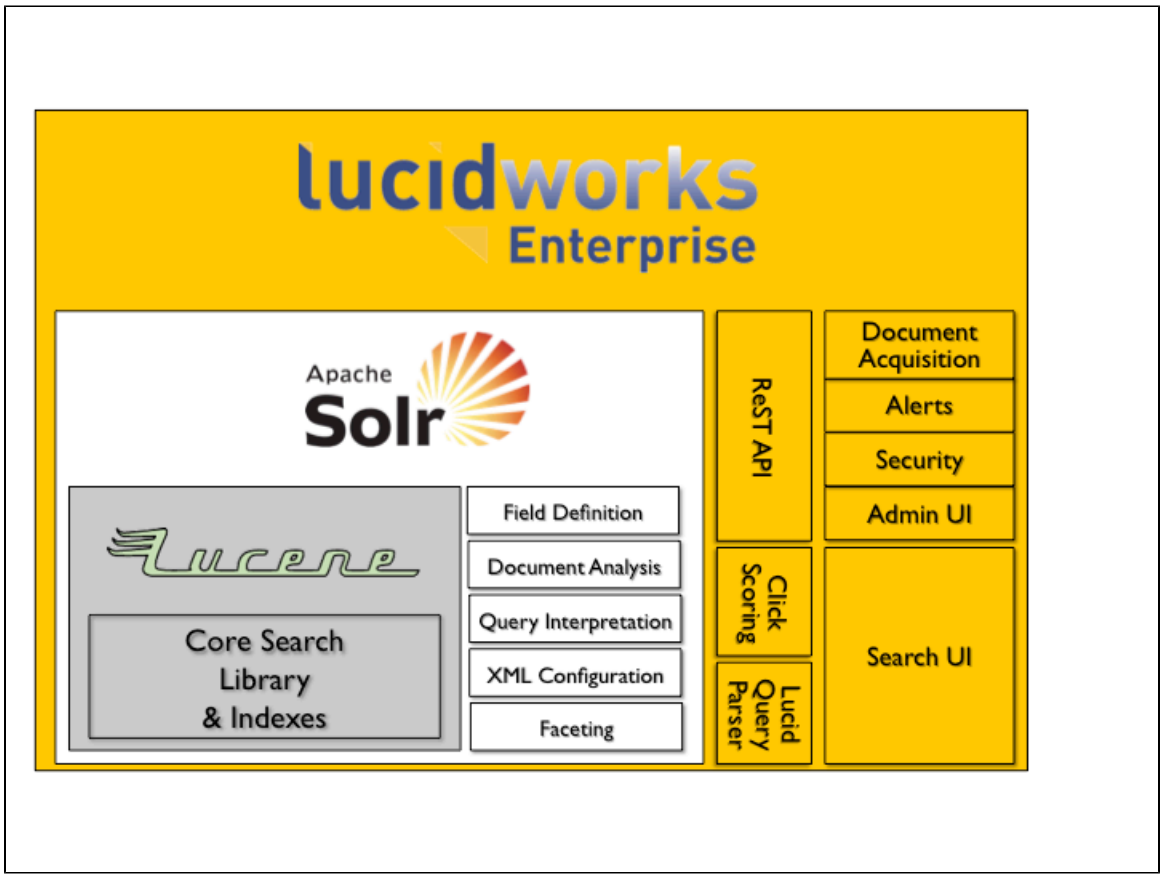
A system that understands natural language (that is, the way people speak or write) perfectly is usually considered the ultimate goal in search engine technology, in that it would do as good a job as a person in finding answers. But even that is not perfect, as variations in human communication and comprehension 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 LucidWorks Enterprise, are built with features that try to solve, or at least mitigate, these challenges.

How LucidWorks Enterprise Works

Like any other search engine, LucidWorks Enterprise 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 Enterprise 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 Enterprise.



The Apache Solr/Lucene core provides the indexing and searching functionality on which LucidWorks Enterprise is built. As an application developer, 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. The Apache Solr/Lucene core can be used as a standalone installation, if you want to work with it directly.

On top of the Apache Solr/Lucene core is LucidWorks Enterprise, which has been designed to take the pain out of running an Apache Solr-based search engine by providing programmatic or admin-interface-level access to features that are normally difficult to work with directly, such as field definition or data source creation and scheduling. It does this in several ways:

- The LucidWorks Enterprise Administration User Interface provides configuration and management tools for almost every feature of LWE, including document acquisition, security, and field definitions.
- LucidWorks Enterprise provides a Search User Interface that includes advanced features such as query completion, "find similar" searches, and integration with click scoring.
- Click Scoring enables LucidWorks Enterprise to adjust search results based on user actions: it automatically adjusts search results according to which ones users click most, and more so if the user's query is similar to the query for which the documents were selected before.
- The REST API provides programmatic access to almost all configuration and management functions within LucidWorks Enterprise.

- LucidWorks Enterprise provides end-to-end SSL security, as well as the ability to limit access to specific results based on a user's identity or group affiliation.
- Enterprise Alerts enable the search application to notify a user when new results have been found for a query.

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

Indexing Documents

Introduction to Indexing

The first step to being able to search is to create an index. Modern search applications use a technique called an "inverted index" to make search more efficient. An inverted index is similar to the index found in the back of a book, where words extracted during the indexing process are listed and stored with pointers for each document and the total frequency of each word (which is later used in relevance ranking). LucidWorks Enterprise also adds location information to each word in order to support proximity searching (where you can specify queries such as dog NEAR puppy).

This example shows how an inverted index may be constructed:

Sample Documents

ID	Text
1	Baseball is played during summer months.
2	Summer is the time for picnics here.
3	Months later we found out why.
4	Why is summer so hot here?

Word List

Term	Freq	Document ids
baseball	1	[1]
during	1	[1]
found	1	[3]
here	2	[2], [4]
hot	1	[4]
is	3	[1], [2], [4]
months	2	[1], [3]
summer	3	[1], [2], [4]
the	1	[2]
why	2	[3], [4]

How Indexing Works

Indexing documents is the slowest part of a search application. Each document needs to be broken into individual words and a word list created. As each new document is indexed, the word list is updated with new words or existing words on the list are updated with pointers to the new documents. The index will be very big (although usually not as big as the documents themselves), and various techniques are used to compress the index to make it smaller. A smaller index saves disk space, lowering hardware costs for the search application, but also allowing faster retrieval during query processing. This compression makes adding new documents a slower process than with relational databases, for example. It is often most efficient to add documents in batches for this reason.

Advanced indexing processes, such as the one used with LucidWorks Enterprise, pay attention to the fact that 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. It also allows the search application to treat the data in each field properly: dates, for example, should be treated differently than an author name, which is treated differently than a price.

Our written language includes a lot of information that is extraneous to a search application when it comes to matching user queries to words extracted from documents. For example, we end sentences with a period, or put periods between individual letters of an acronym. To humans, there is no difference between UCLA and U.C.L.A., but a computer will treat those as two different words because they are literally different strings. However, a user searching for UCLA probably does not care much whether it is spelled with periods or not in the matching document (there are

cases where it matters, but generally it does not). To overcome these differences in our written language, words are normalized in several ways during the indexing process. All terms are made lower case so differences in capitalization do not impact results. Plural words are made singular so users who enter dogs will also find dog. Punctuation, apostrophes, accent marks, and other special characters are stripped.

Stop Words

It used to be quite common for search applications to remove very common words, often called stop words, such as a, the, of, from, and so on from the index to save disk space. Because these terms were most common, they had the largest document lists associated with them, and they are usually the least useful in actually finding the right document. However, disk space is many times cheaper now than it used to be, and compression of the index is also vastly improved so conservation of disk space is much less a concern than it used to be. And, while most of the time these terms don't add much to a search, excluding them meant they could never be used, even when they were the most essential part of the search (the failure of "to be or not to be" as a query was a common example from the 1990s of the cost of removing these stop words from the index). There may be valid reasons to remove stop words from a user's search, but there are few reasons to exclude them from the index. We will discuss the impact of stop words on a user's search later.

Indexing Data Sources

In order for users to be able to search, LucidWorks Enterprise provides a way for administrators to configure data sources to collect documents and index them. The available data source types are preconfigured to be able to parse documents and understand the fields commonly found in documents: for example, the Web data source understands the fields commonly found on web pages, so the content found there is indexed appropriately. Set up data sources on the Index - Sources screen, or in the Quick Start wizard. An important factor in configuring LucidWorks Enterprise is to determine how often to revisit each data source for new or updated content.

Managing Indexes and Fields

Each document to be indexed can be broken up into several parts, depending on the structure of the document. For example, most text documents consist of a title, author, and main body. It may also contain date created and/or modified, permission information, and version data. A list of phone numbers may include personal information such as name and address in addition to the actual phone numbers. These parts are called fields, and properly parsing each document into its component parts is essential to effective indexing in order to optimize the search experience for users and minimize the size of the index.

LucidWorks Enterprise includes information on how to parse several types of documents such as Microsoft Word files, web pages, and PDF files and contains fields common to these types. Databases can be indexed by mapping columns to pre-configured fields, or new fields can be created as needed. XML files in Solr format can also be used to explicitly name parts of documents to ensure proper parsing during indexing. Lucid Imagination has provided smart defaults for how to define fields, but the Fields screen in the Administration UI allows administrators to change the default settings or to add custom 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.

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 Enterprise 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 Enterprise, 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.

Installation and Upgrade Guide

This guide covers installation, deployment, and server administration processes to get LucidWorks Enterprise installed on your server.

Installation describes how to install and uninstall LucidWorks Enterprise, including system requirements. Two installation modes are available: a wizard-like graphical interface and command line installation for headless servers. Installation time should take no longer than a few minutes.

If LucidWorks Enterprise needs to be installed on multiple servers with the same options, automation is possible by creating a script. Read more at Automating Installation Options for Installation to Multiple Environments.

The installer provides an option to start LucidWorks Enterprise. However, there are other points during operation of the application where it may be necessary to start and then restart the application. Information on how to do that can be found at Starting and Stopping LucidWorks Enterprise.

As you start to use LucidWorks Enterprise, an understanding of the directory structure, where to find logs, and how to work with the components will be invaluable. That information is described at System Directories and Logs and Working with LucidWorks Enterprise Components.

Those customers migrating from v1.7 or v1.8 of LucidWorks Enterprise now have a migration path that can be followed to move to v2.0. There are several steps to this process, which are described at Migrating from a Prior Version.

Installation

There are two ways of installing LucidWorks Enterprise:

- 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-enterprise-installer-2.0.jar.

For Windows systems, the install file is called lucidworks-enterprise-installer-2.0.exe. On Windows, the installer should be run with Administrator privileges to ensure proper installation.

Requirements

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

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. The LucidWorks Enterprise automatically installs and configures Jetty, an open source Java servlet container. You do not need to take any extra steps to configure a servlet container for LucidWorks Enterprise.

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:

- Depends on the size and number of documents to be indexed. Contact Lucid Imagination for assistance in determining the disk space needed for your implementation.

Running the Installation Wizard

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

6. Configure the components dialog box to select the LWE 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 Enterprise will be installed. 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 Summary of Choices screen appears.

10. Confirm your installation choices, then click Next. The LucidWorks Enterprise Installation begins.

11. When the installation is finished, click Next. The Start LucidWorks Enterprise screen appears. To start LucidWorks Enterprise immediately, check the Start LucidWorks box.

12. Click Next. The installer initiates the LucidWorks Enterprise 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.

13. Click Next.

14. 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 Enterprise. 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

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

1. Launch the installer with the command `java -jar <file-name.jar> -console`.

```
bash-3.2$ java -jar lucidworks-enterprise-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/LucidImagination/LucidWorksEnterprise
press 1 to continue, 2 to quit, 3 to redisplay
1
```

5. The installer installs LucidWorks Enterprise, and asks if you want to start the servers.

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


LucidWorks Enterprise has been installed.  Would you like to start LucidWorks Enterprise
now?
NOTE: LucidWorks Enterprise 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 Enterprise may take some time to finish
loading even after the installer has been closed.
  [x] Start LucidWorks Enterprise
input 1 to select, 0 to deseclect:
1
```

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

```
Install was successeful
application installed on /Applications/LucidImagination/LucidWorksEnterprise
[ Console installation done ]
```

If you specified that the installer should start LucidWorks Enterprise, 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

Generating an Installation Script from the Graphical Installer

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:

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 Enterprise 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-enterprise-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/LucidWorksEnterprise</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 Enterprise 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 onWorking With LucidWorks Enterprise Components for more information about what each component does.

Set the Installation Path

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

```
<installpath>/Applications/LucidImagination/LucidWorksEnterprise</installpath>
```


Start LucidWorks Enterprise after Script Completion

LucidWorks Enterprise 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>
```

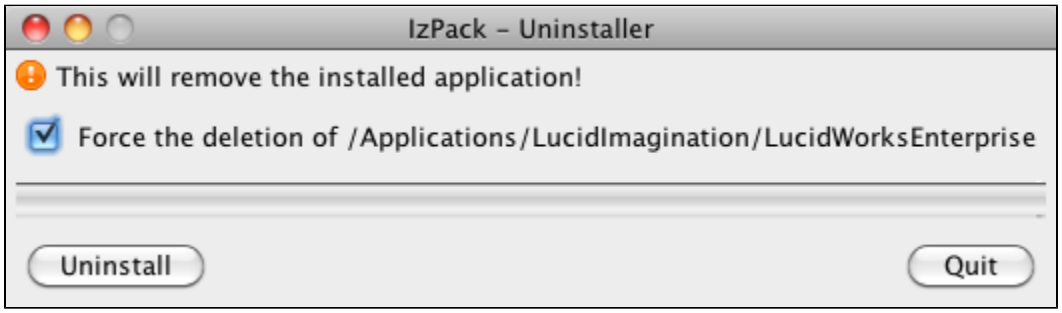
Once you have made the appropriate edits, save the file with a name such as `config-options.xml`. To use it in future installations, run the installer from the command line with a command such as (be sure to replace our sample file name with the one created locally):

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

Uninstalling Lucid Works Enterprise

You can uninstall LucidWorks Enterprise 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.

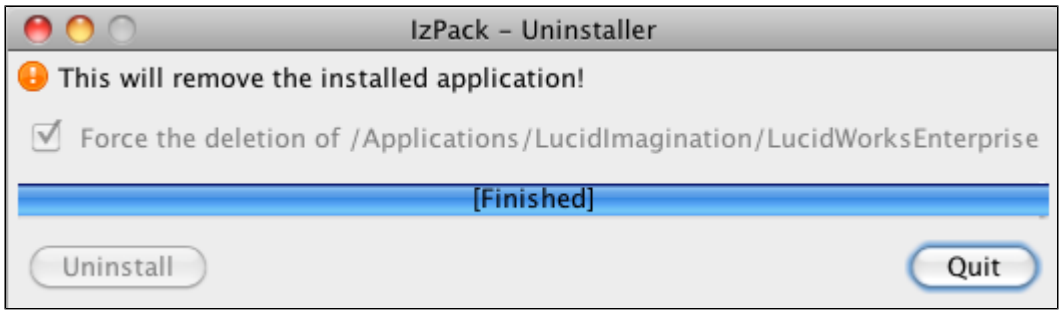
1. Launch the appropriate file. The IzPack - Uninstaller dialog box appears:



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.



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 Enterprise on a server without GUI access, stop all running LucidWorks Enterprise processes, then manually delete the parent directory. This will remove all indexes and associated data.

Working With LucidWorks Enterprise Components

Basics of LucidWorks Enterprise Components

LucidWorks Enterprise has two main components that can each be run together on a single server or deployed on separate servers if desired:

LWE Core

The LucidWorks Enterprise (LWE) 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 Enterprise and review results. It includes search features such as automatic completion of queries, spell checking, faceting, and sorting. 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.

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/

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.

Configuring the Components

If LWE Core and LWE UI 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. Both 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. Both components run on the same machine but they are started and stopped separately. This would require running the installer two 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 twice, 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 'server1' on port 8888
- Run the LWE UI component on the machine 'server2' port 8989

In this scenario, we need two installations, and our installation steps would be something 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 component so it will not be installed on server1.
    4. Change the URL for the LWE UI to http://server2:8989. Even though it 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. Advance to the Next screen, and select a path for this installation.
    6. Finish the installation.
- Second Installation, LWE UI on 'server2'
    1. Launch the installer again, and advance to the component selection screen.
    2. Deselect the "Enable" check boxes for the LWE Core component so it will not be installed on server2 but change the URL to http://server1:8888 so LWE UI configuration can talk to LWE Core.
    3. Change the URL for LWE UI to http://server2:8989.
    4. Advance to the Next screen, and select a path for this installation on server2. If you were instead installing the LWE UI on the same machine, you would enter a different installation path.
    5. Finish the installation.

Please note:

- Starting and stopping server processes will need to occur in both installations; there is no single start/stop script that will work across multiple installations.
- The `$LWE_HOME/conf/master.conf` files in both installations will continue to refer to components that were not installed. You must take care not to enable a component in the wrong installation.
- The Click Scoring Relevance Framework runs as part of the LWE Core, and requires that `click-<collectionName>.log` (generated by the Search User Interface) be available in the `$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

Locating Files and Directories

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

What	Path
Configuration Files	`$LWE_HOME/conf/`
Documentation	`$LWE_HOME/app/docs/` (PDF format) and `$LWE_HOME/lwe-core/doc/` (HTML format)
Examples	`$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_name`*`/data/`
LucidWorks Logs	`$LWE_HOME/data/solr/cores/LucidWorksLogs/data/`
Solr Home	`$LWE_HOME/data/solr/`
Solr Configuration Files	`$LWE_HOME/conf/solr/`
Solr Source Code	`$LWE_HOME/app/solr-src/`
Start/Stop Scripts	`$LWE_HOME/app/bin/`

Editing Configuration Files on Windows
LucidWorks Enterprise 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 Enterprise before editing files on Windows to be sure the edits are saved properly.

Configuring LucidWorks Enterprise Directories

After you have installed LucidWorks Enterprise, 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
```

System Logs

LucidWorks Enterprise 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	Function
`$LWE_HOME/data/logs/core.<YYYY_MM_DD>.log`	LWE Core operations, including information from crawling activities
`$LWE_HOME/data/logs/core-stderr.log`	Errors from Jetty startup
`$LWE_HOME/data/logs/core-stdout.log`	Messages from Jetty startup
`$LWE_HOME/data/logs/request.<YYYY_MM_DD>.log`	API requests from front-end systems (Search, Admin and Alerts) to the back-end LWE Core
`$LWE_HOME/data/logs/ui.log`	Information from the Rails application, which runs the Search, Admin and Alerts components
`$LWE_HOME/data/logs/ruby-stderr.log`	Errors from Ruby startup
`$LWE_HOME/data/logs/ruby-stdout.log`	Messages from Ruby startup
`$LWE_HOME/data/logs/click-<collectionName>.log`	User click data, for use in relevance boosting (if enabled)
`$LWE_HOME/data/logs/alert_trace.log`	Alert heartbeat that checks for new alerts to run
`$LWE_HOME/data/logs/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 Enterprise Core log is configured by a properties file (`$LWE_HOME/conf/log4j.xml`). The default is to create a distinct log per date (server time). Only data from the most recent `core.YYYY_MM_DD.log` file is included in query and index history calculations that appear in the Admin User Interface.

Rotation of the ui.log File
On Windows machines, log rotation of the `ui.log` file only has been disabled. The LWE Core log will still be date specific, but the `ui.log` will need to be manually rotated.

LucidWorksLogs

LucidWorks Enterprise 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 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.

Starting and Stopping LucidWorks Enterprise

Starting LucidWorks

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

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

To stop LucidWorks Enterprise, use the stop scripts at the command line. This will stop all servers 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 Enterprise
To restart LucidWorks Enterprise, first stop the servers and start them again using the processes outlined above.

Migrating from a Prior Version

It is not yet possible to perform a fully automated upgrade from one LucidWorks Enterprise version to another, but it is possible to migrate configuration and data from a prior version to v2.0. Migration is possible only from v1.7 or v1.8 to v2.0.

While the migration steps are outlined below, it is recommended that this only be attempted after consultation with Lucid Imagination Support to be sure the latest version of the tools are installed and all customizations are accounted for.

One issue to be careful of is that data source IDs may change when importing configuration data from the prior installation of LucidWorks Enterprise to the new one. This is because the Update Tool imports the configuration data and allows LucidWorks Enterprise to use it's own processes to assign new data source ids. However, the index data is simply converted, leaving documents with the same data source related fields intact. This mismatch will not destroy the system, but may cause unexpected behavior in the Admin UI and in search results (such as, document counts may not appear correctly in the Admin UI). Deleting documents may also be problematic, meaning that using the UI to delete documents from data sources may delete documents for the wrong data sources.

It is highly recommended to run through this process in a test environment to understand the likely outcome before using this in production.

The migration steps are as follows:

1. Export system configuration from the older version while LucidWorks Enterprise is running. Be sure to review the issues.txt output of the tool to understand what problems may occur during the rest of the migration.
2. Stop LucidWorks Enterprise and run the Upgrade Tool. Save the output files where they can be found later.
3. Install LucidWorks Enterprise v2.0 on a new server or in a different location from the older version.
4. Migrate the LucidWorks Enterprise configuration to the new version (with LucidWorks Enterprise running).
5. Update configuration files with customizations that are not supported by the Update Tool.
6. Upgrade the index for each collection and put the new index files in the proper places in the new installation. LucidWorks Enterprise should not be running while this occurs.
7. Re-generate auto-complete indexes for each collection, if desired.

At the current time, it is not possible to install a new version of LucidWorks Enterprise over an existing one. If upgrading the application on the same server it is currently running on, a new directory path will need to be input into the installer to continue. Also, make sure to define the component ports on the new installation as different from the existing LucidWorks Enterprise.

LucidWorks Update Tool

The LucidWorks Update Tool is a standalone utility that uses the LucidWorks REST API to enable a user to capture customized configuration information from a running LucidWorks Enterprise server, store the information in a local zip file, and update another LucidWorks Enterprise server to match the same configuration. The tool provides an export/import capability and a product upgrade capability, but it operates external to LucidWorks Enterprise rather than as an embedded feature. The tool also generates a human-readable summary of the LucidWorks Enterprise schema. Currently, this tool only collects information that is available with the REST API. Some changes, such as manual edits to `solrconfig.xml` or `schema.xml` files, will need to be migrated manually.

- How the Update Tool Works
- Configuration Fixes for This Release
- Using the Update Tool
- Examples
- Tool Output
    - Directory Structure of the Output Zip File
    - The Summary File
    - The Issues File
    - The Version Files

- Schemas for Past Releases

How the Update Tool Works

The tool collects information about and is able to update:

- collections
- data sources
- fields
- field types (will be updated only if the source and target servers support the Field Type API released in v2.0)
- search components (currently only `ad`)
- filter component configurations (without passwords, as these are not returned by the API)
- search handlers (currently only `/lucid`)
- collection settings
- stop words
- synonyms
- roles
- users
- authorizations
- alerts

It does not collect actual indexed document data. If you are unable to re-index your data, the Index Upgrade tool will permit you to migrate existing index data to a new LucidWorks Enterprise.

The tool attempts to correct the configuration as needed to accommodate any incompatibilities between releases. This can include moving from an existing LucidWorks Enterprise server to a LucidWorks Enterprise server running a new version. The tool can also display or save a readable summary of the configuration for a LucidWorks Enterprise server. You can read directly from one server and write directly to another server, or optionally save the config info into a zip file and write to a server later. You could even save the configuration for a server in a zip file and later restore it from the saved zip file. While there are numerous ways to use the exported configurations, the primary purpose and design of the tool is to ease upgrading from an older version of LucidWorks Enterprise to a newer one.

There is a preview mode option so you can display a list of changes that need to be made to a target server to get it to match an existing config without making the changes. The servers can be local or remote.

As a convenience, the Update Tool also saves a copy of all of the Solr "conf" files (`schema.xml`, `solrconfig.xml`, synonyms, stopwords, stemming rules, etc.) in the zip files that it creates to save a LucidWorks Enterprise schema. But, although these files are kept in the zip file, they are not pushed to the destination server when doing an update except in cases where LucidWorks Enterprise contains an API for a specific part of the configuration (i.e., Fields, Field Types, and some settings).

Use the Latest Version of the Update Tool
The Update Tool can capture from any release of LucidWorks Enterprise, but can only update to a LucidWorks Enterprise server running the "current" version of LucidWorks Enterprise. The tool will display an error message and exit if the target LucidWorks Enterprise server is not running the latest release.

You can use an old version of the tool to capture from an old LucidWorks Enterprise, but you need to use a newer version of the tool to update a newer LucidWorks Enterprise.

To be sure you're using the latest version of the tool itself, check our website at http://www.lucidimagination.com/products/lucidworks-search-platform/enterprise/update-tool-20 to see if we've made any changes.

Configuration Fixes for This Release

For LucidWorks Enterprise v2.0, the Update Tool will make the following changes:

- As of version 2.0, LucidWorks Enterprise no longer supports the `index_time_stopwords` setting. If a saved schema has this setting, the tool will automatically remove it and automatically insert a stop word filter into the index analyzer for the text_en field.

- The acl field will be preserved in the destination LucidWorks Enterprise even if it is not defined in the source collection. This is a new field in v2.0.
- The aclusers and aclgroups fields are ignored since the crawlers will dynamically add attr_aclusers and attr_aclgroups as dynamic fields (aclusers and aclgroups are renamed attr_aclusers and attr_aclgroups in v2.0).
- All attribute names that were defined in CamelCase in earlier versions have now been standardized to use names with underscores; the tool will automatically map these.
- Users migrated from earlier versions will be given passwords of "default-user-password". This can be overwritten with the `-D` option, as below.
- In v2.0, there is no longer a separation between Users and Authorizations (which defined what parts of the UI a user could access), in fact, the Authorizations API has been removed. Users are now given an authorization in their user account to access either "admin", allowing access to the Admin UI and Search UI, or "search", allowing access to the Search UI only. During migration, users with the former Authorization of ADMIN will be mapped to having an "admin" authorization, while those with SEARCH will be mapped to "search".
- Data source and activity schedules will be migrated, but because the APIs and prior versions of LucidWorks Enterprise allow more flexibility in setting a schedule than the v2.0 UI, the tool updates activity schedules (which are the actions that optimize the index, create an autocomplete index, and process click logs) in the following ways:
    1. By default, the tool will update any existing activity schedule in the new LucidWorks Enterprise server (the "target" server)
    2. The tool will create a single activity schedule in the target server. If there are multiples from the source server, the first one will be used and the rest discarded. The tool will report when it has done this.
    3. The user can use the `-noUniqueActivitySchedules` option to force the Update Tool to create activity schedules in the target server for every activity that was present in the source. If using this option, however, note that only one schedule is displayed in the v2.0 UI, which may be confusing for some users if there are multiples of any schedule type.

Using the Update Tool

To use the Update Tool, run `LweUpdate-2.0.jar` found in `$LWE_HOME/app/migration_tools/lwe-update-tool` with the following command:

```
java -jar LweUpdate-2.0.jar [options]
```

There are several possible options, as follows:

Option	Parameter	Description
`-r`	URL	Reads from a LucidWorks Enterprise server. Parameters can be localhost, remotehost or the fully defined URL. See below for how to set the `LWE_URL` in your environment to use remotehost as the parameter.
`-o`	file name	Outputs the schema to a file. If no file extension is specified, .zip will be used by default. Use with the `-r` option to read the configuration and output it to a file.
`-i`	file name	Installs the schema from a file to the target LucidWorks Enterprise server.
`-w`	URL	Writes the schema to the target LucidWorks Enterprise server.

`-D`	default-user-password=new-password	Overwrites the default password for all users being migrated from "default-user-password" to a user-supplied alternative. This is a Java system property and should be entered before the `-jar` portion of the command, as in: `java -Ddefault-user-password=new-password -jar LweUpdate-2.0.jar [options]`
`-preview`	none	Displays the changes that would be needed to the target LucidWorks Enterprise server, but does not make the changes.
`-update`	none	Enables changes to the target LucidWorks Enterprise server if it appears to be already configured. By default, if the target server has been configured the tool will quit.
`-empty`	none	Deletes all documents in all collections in the target LucidWorks Enterprise server. This is the default behavior if this is not specified.
`-keep`	none	If documents have been indexed in the target LucidWorks Enterprise server before the configuration has been imported, this option will retain the documents and their history information.
{{-noUniqueActivitySchedules	none	Allows multiple activity schedules to be migrated from the source to the target, if multiples existed in the source. Note that the v2.0 UI only displays one activity schedule, even if multiples exist. The default behavior is only migrate the first schedule of any type (click, autocomplete, or optimize) and discard the others with a warning.
{{-uniqueActivitySchedules	none	Enforces the default behavior to only migrate the first schedule of any type. It is not required, but is available as an additional assurance.
`-version`	none	Displays the name and version of the tool.
`-q`	none	Suppresses all INFO-level log messages.
`-ra`	URL	Defines the URL where the alerts component has been configured if it is not the default.
`-wa`	URL	Defines the URL where the alerts component should be configured in the target LucidWorks Enterprise server if it should not be the default.

If accessing a server with HTTPS configured, the following four JVM properties must also be specified before the -jar part of the java command.

- `-Dclient-keystore-url=<URL or path>`: The URL or path for the key store file.
- `-Dclient-keystore-pass=*<password>`: Password for the key store file.

- `-Dclient-truststore-url=<URL or file://path>`: URL or path for the trust store file.
- `-Dclient-truststore-pass=*<password>`: Password for the trust store file.

In addition to specifying a URL on the command line, you may define the `LWE_URL` environment variable and then simply refer to `remotehost` or `LWE_URL` where a LucidWorks Enterprise server URL is required (as with options `-r` and `-o`, for example). To do this in bash for example, you would enter the following at the command line:

```
LWE_URL=http://www.my-company.com:8888; export LWE_URL
```

Examples

1. Read user config info from a source LucidWorks Enterprise server to get a brief summary of what is on it: collections, data sources, and fields, and document counts by collection and data source.
    a. Define the `LWE_URL` and use remotehost instead of the server URL

    ```
    java -jar LweUpdate-2.0.jar -r remotehost
    ```

    b. Use localhost as the URL

    ```
    java -jar LweUpdate-2.0.jar -r localhost
    ```

    c. Define the full URL

    ```
    java -jar LweUpdate-2.0.jar -r [http://some.host:port/]
    ```

2. Read configuration from the source LucidWorks Enterprise server and save it in a local zip file.
    a. Define the file name with the extension

    ```
    java -jar LweUpdate-2.0.jar -r remotehost -o local-schema.zip
    ```

    b. Define the output filename and accept the default extension

    ```
    java -jar LweUpdate-2.0.jar -r remotehost -o local-schema
    ```

3. Read configuration from the source LucidWorks Enterprise server and save it in a local zip file, then install a new version of LucidWorks Enterprise and write the saved config info to the new server
    a. Create the output

    ```
    java -jar LweUpdate-2.0.jar -r remotehost -o local-schema.zip
    ```

    b. Import to the target server

    ```
    java -jar LweUpdate-2.0.jar -i local-schema.zip -w http://some.other.host
    ```

4. Read configuration from the source LucidWorks Enterprise server, optionally saving it in a local zip file, then update the target LucidWorks Enterprise server to match the source configuration
    a. Write the output to a file and write it to the target server

    ```
    java -jar LweUpdate-2.0.jar -r remotehost -o local-schema.zip -w http:
    //some.other.host
    ```

    b. Or, skip the step of outputting it to a file and write it directly to the target server


```
java -jar LweUpdate-2.0.jar -r remotehost -w http://some.other.host
```

5. Read configuration from the source LucidWorks Enterprise server, optionally saving it in a local zip file, then use the preview option to get a report of the changes that would be needed to another LWE server to match that user config info, but do not make the changes
    a. Preview with configuration output

```
java -jar LweUpdate-2.0.jar -preview -r remotehost -o local-save-schema.zip -w
http://some.other.host
```

    b. Or, skip the step of outputting it to a file

```
java -jar LweUpdate-2.0.jar -preview -r remotehost -w http://some.other.host
```

6. Write the saved config info from a local zip file to a target LWE server

```
java -jar LweUpdate-2.0.jar -i local-schema.zip -w remotehost
```

7. Restore a LWE server to a prior state that had been saved in a local zip file
    a. Create the output

```
java -jar LweUpdate-2.0.jar -r remotehost -o local-schema.zip
```

    b. Import to the new target server

```
java -jar LweUpdate-2.0.jar -i local-schema.zip -w remotehost
```

8. Reads a schema from the source LucidWorks Enterprise server with alerts running on port 9898 and writes it to save-schema.zip

```
java -jar LweUpdate-2.0.jar -r localhost -ra http://localhost:9898 -o save-schema
```

9. Reads a schema from the source LucidWorks Enterprise server and updates the target LucidWorks Enterprise server at 'http://someurl'. Alerts will be read from port 9898 of localhost and updated at port 7979 on the target

```
java -jar LweUpdate-2.0.jar -r localhost -ra http://localhost:9898 -w
http://someurl:8888 -wa http://someurl:7979
```

10. Reads a schema from a remote server with both an SSL key store and trust store and writes it into save-schema.zip

```
java -Dclient-keystore-url=file:///home/me/mykeystorefile -Dclient-keystore-pass=secret1
-Dclient-truststore-url=file://mytruststorefile -Dclient-truststore-pass=secret2
-jar LweUpdate-2.0.jar -r https://some-url.com:8888 -o save-schema
```

Tool Output

If using the `-r` option alone, a basic summary is shown in the terminal window as the tool runs. More detail is provided when the output is sent to a file with the `-o` option. When looking at the output, there are several files included.

Directory Structure of the Output Zip File

- `summary.txt`
- `issues.txt`
- `master.conf.json`
- `version.json`
- `version.txt`
- `collections.json`
- `collection_sizes.json`
- A directory for each collection, with the same name as the collection "instance" directory in the LucidWorks Enterprise installation:
    - `datasources.json`
    - `fields.json`
    - `fieldtypes.json`
    - `info.json`
    - `settings.json`
    - `roles.json`
    - `datasource_schedules.json`
    - `datasource_histories.json`
    - `datasource_sizes.json`
    - `search_components.json`
    - `filter_components.json`
    - `jdbc_driver-files.json`, which is a list of filenames, not the actual driver files
    - `jdbc_driver_classes.json`
    - a `conf` directory, which includes the Solr `schema.xml` and `solrconfig.xml` files, as well as all the other files from the Solr `conf` directory of each collection.
- `users.json`
- `authorizations.json`
- `collection_templates.json`
- `alerts.json`

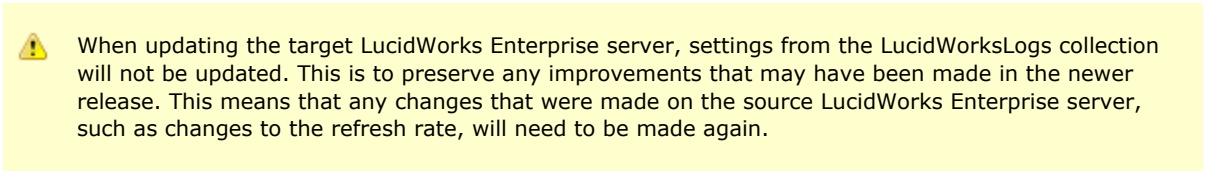
The `.json` files are structured as array blocks, pretty-printed for readability and to facilitate editing if needed.

The Summary File

The `summary.txt` file is a readable file that summarizes the entire installation that is generated in the output zip file includes:

- URL of the LucidWorks Enterprise server and the alerts URL
- Date and time when the info was captured
- The LucidWorks Enterprise release number and all the version info returned by the REST API (as raw JSON)
    - If version info is not present, the tool will heuristically determine the LWE release based on the availability or unavailability of various API features
- Number of collections, users, authorizations, and alerts
- For each collection (including the LucidWorksLogs collection):
    - Name
    - Directory name
    - Template name
    - Number of indexed documents
    - Number of data sources
    - Number of fields
    - Number of field types
    - For each data source:
        - Name
        - Id
        - Category
        - Type
        - Crawler
        - Number of indexed documents
        - Schedule
        - Status of last crawl job
    - For each field:
        - Name

        - Type
        - Whether the field is:
            - Indexed
            - Stored
            - Searched by default
            - Included in search results
            - Used as a facet
    - Summaries of fields:
        - List of fields searched by default
        - List of fields included in search results
        - List of fields used as facets
    - For each field type:
        - Name
        - Class
    - Info for the collection (directories, disk space, etc.)
    - Settings for the collection, including stop words and synonyms
    - For each role:
        - name
        - List of groups
        - List of users
        - List of filters
    - For each search component (currently hard-wired to one)
        - Handler name (e.g., "/lucid")
        - List of search component names (e.g., "filterbyrole", "query", etc.)
    - For each filter component configuration
        - Filter component name (e.g., "ad")
        - Filter component configuration (raw JSON) - excludes password
    - List of JDBC driver files
    - List of JDBC driver classes
- For each User:
    - User name
    - Last name
    - First name
    - Email address
    - Password (which, unfortunately, is always just eight asterisks)
    - Password hash which is used in place of the actual password
- For each authorization:
    - Name of the authorization (ADMIN, SEARCH, ALERTS)
    - List of group names authorized for this activity
    - List of user names authorized for this activity
- List of collection templates



The Issues File

The log output and `summary.txt` will include any potential problems that were detected by the tool, but the list of such issues is also written to `issues.txt` for quick and easy review.

The Version Files

There are two version files. The first, `version.json`, contains the JSON returned by the LucidWorks Enterprise Version API, but that API is only supported from v2.0 forward. The second file, `version.txt`, is a simple, single-line text file that contains the LucidWorks Enterprise release version number which either came from the version API or was determined heuristically by the tool.

The LWE release version numbers recognized by the tool are:

- 1.6
- 1.7
- 1.8
- 2.0

Schemas for Past Releases

The install comes with and creates a directory called `$LWE_HOME/app/migration-tools/release-schemas`, which has pre-run captures of each of the recent LucidWorks Enterprise versions. These can be used for reference and can also be used to reset a server to a fresh, blank state. The supported releases are currently:

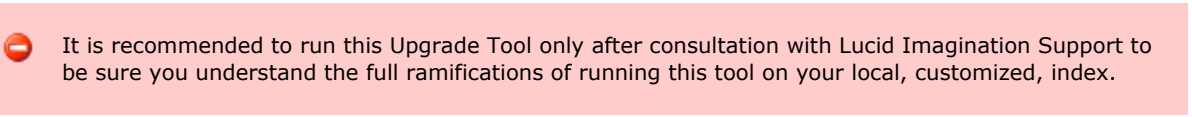
- 1.7: `$LWE_HOME/app/migration-tools/release-schemas/1-7-schema.zip`
- 1.8: `$LWE_HOME/app/migration-tools/release-schemas/1-8-schema.zip`
- 2.0: `$LWE_HOME/app/migration-tools/release-schemas/2-0-schema.zip`

There is also a set of example schemas that show what a capture looks like with collections, data sources, fields, users, roles, authorizations, and alerts already defined. These are found in `$LWE_HOME/app/migration-tools/example-schemas`, with zip files for each version (1.7, 1.8 and 2.0).

Index Upgrade Tool

In order to upgrade an index from an earlier version of LucidWorks Enterprise to v2.0, the Index Upgrade Tool should be run. This tool is found under `$LWE_HOME/migration-tools/` in a .jar file called `IndexUpgradeTool.jar`.

The tool will support upgrading an index from v1.7 or v1.8 to v2.0. Earlier versions are not supported.



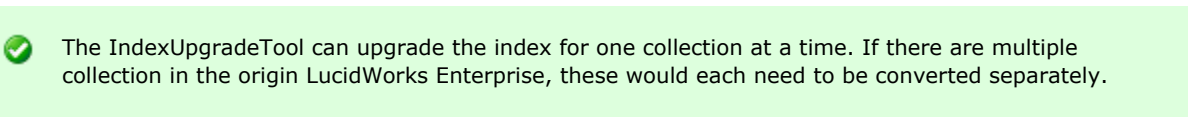
Lucid Imagination regularly reviews how our term indexing and query interpretation with our "lucid" query parser operates and may make changes from time to time. In some cases, this means that queries may not behave the same between releases. In particular, for v2.0, we have changed the way we handle indexing "CamelCase" terms from documents and processing queries entered in that format. Previously, these terms were indexed as single terms, but we have changed this so now they will be indexed as multi-term phrases. This means queries entered in CamelCase will also be interpreted as multi-term phrase queries. However, the index upgrade tool does not convert these terms to multi-term phrases, so moving forward there may be a mix of terms in the index, such as "SharePoint" and "Share Point". 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 Enterprise 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 and input the command:

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

The `<sourcedir>` parameter is the existing index that will be moved to the `<destinationdir>`. In v1.7 and v1.8, the `<sourcedir>` is found at `$LWE_HOME/solr/cores/`*`collection`*`/data/index`, where *`collection`* is the name of the collection whose indexes are being upgraded. In v2.0, the directory structure has changed, so the `<destinationdir>` would be `$LWE_HOME/data/solr/cores/`*`collection`*`/data/index`.



There is currently one option that can be used with the tool, 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.

About Spell Check and Auto-Complete Indexes From Prior Versions
Before LucidWorks Enterprise v2.0, when spell checking and/or auto-complete was enabled a separate index was created. In v2.0, the spell check feature can use data from the 'spell' field in the main index and creation of a separate index is no longer required. For this reason, the spell check index is not upgraded, although the 'spell' field from the main index will be converted as part of the main index upgrade.

The auto-complete feature still uses a separate index in v2.0. However, this index is not upgraded as part of this process. After upgrading the main index, the auto-complete index will need to be re-populated by running the `autocomplete` job with either the Admin UI or the Activities API.

LucidWorks Enterprise User Interface Guide

Error formatting macro: redirect: java.lang.NullPointerException

The LucidWorks Enterprise User Interface Guide is now located at http://lucidworks.lucidimagination.com/display/help.

You should be redirected in 5 seconds. If not, click the link above to go directly to it.

System Configuration Guide

The System Configuration Guide provides detailed information about many of the features included with LucidWorks Enterprise.

Many of the settings used by the crawlers are configured by default, and can be edited with the `$LWE_HOME/conf/lwe-core/default.yml` file. More information is available on the page Configuring Default Settings.

Content in LucidWorks Enterprise is organized into collections, which can each have different documents, data sources, fields, field types and settings. Before starting to configure LucidWorks Enterprise, it's important to understand how collections work and when you may need to create new ones. Working with Collections provides an overview of how this works. When it's time to create new collections, it is possible to create collection templates to speed the initial setup process.

When ready to start adding documents to the LucidWorks Enterprise Index, there are several items to consider such as if the fields in the documents are configured correctly and use the proper field types and if the default mapping of documents to the various fields is right. Crawling and Indexing Configuration covers these topics and other options such as synonyms and stop words, setting up a batch-only crawl (where documents are not indexed), and specific advice for particular content repositories. If you need to delete the index and start over, that's covered too.

Once documents have been indexed, the system is ready to serve queries. There are several options available here too, covered in Query and Search Configuration. Users can set up alerts, spell check helps them recover from spelling mistakes, auto-complete prompts users to use terms that are already known to exist in the index, and document highlighting shows the user's search terms in the context of the document.

One issue that is important to users is that the results are relevant to them. We've included a chapter on Understanding and Improving Relevance to understand how relevance ranking works in LucidWorks Enterprise and what can be done to adjust it if needed. In some situations, the documents that other users have previously selected (either in general or for the same query) may be considered more relevant, and LucidWorks Enterprise includes the Click Scoring Relevance Framework to take those user clicks into account.

Securing the system should be a high priority, and the section on Security and User Management describes how to integrate LucidWorks Enterprise with an LDAP or Active Directory server, how to enable SSL for the system, and how to restrict access to content.

System performance is a big factor in getting users to adopt and continue to use the search application. Performance Tips describes how to approach judging and improving performance metrics, while Integrating Monitoring Services shows how to hook LucidWorks Enterprise into JMX, Zabbix or Nagios to keep an eye on how the system is running. At some point, you may need to increase the number of servers that are indexing and serving user queries - the options are described in Expanding Capacity.

Finally, LucidWorks Enterprise is built on Solr, and instead of hiding that under the hood, we provide a way to access it natively if that is easier for your application. Solr Direct Access describes how to do that in detail.

Configuring Default Settings

You can configure many default settings in LucidWorks Enterprise in the `defaults.yml` file located in the `$LWE_Home/conf/lwe-core` directory. You must restart LucidWorks Enterprise 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:

```
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
  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
# 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, 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
  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
```


Working with Collections

LucidWorks Enterprise supports 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. There are times, however, when multiple index rules need to coexist in the same system, perhaps to index very different types of content (books vs. movies, for example), or when you need to be more confident in limiting user access to some documents.

The concept of collections in LucidWorks Enterprise is very similar to the concept of cores in Solr. Under the hood, a collection is the same as a core in Solr, however we have added functionality to the LucidWorks Enterprise GUI to make configuring collections straight-forward and navigating between collections simple.

You can configure the following items for each collection individually:

- Datasources
- 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 Administration User Interface (Admin UI). Modifying the wrong collection out of context may have unexpected consequences including poorly indexed content or an inconsistent search experience for users.

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

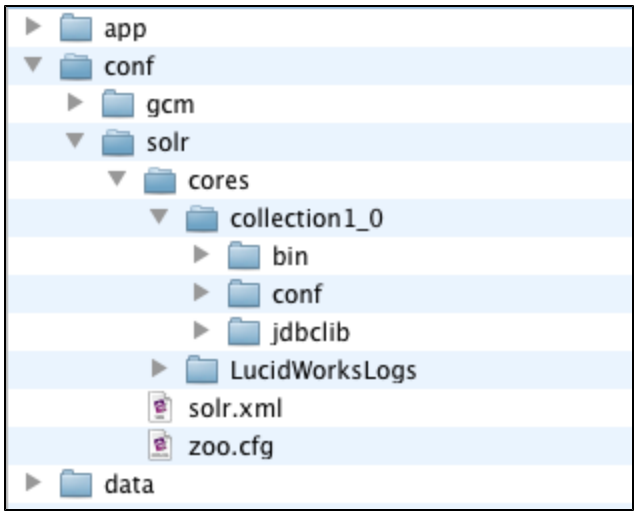
- Collection definition
- Access to user interfaces
- Users
- Alerts (although these take the collection as a parameter to limit the query)

Using Collection Templates

It is possible to create templates for collection creation by configuring one collection and creating a `.zip` file that is called during new collection creation either via the user interface or the REST API. Each template is a .zip file that consists of LucidWorks configuration files. The default LucidWorks configuration is available as `default.zip`. The default configuration can be customized as needed and put in a .zip archive for use when creating new collections in the future.

We have also created a stripped-down version of the LucidWorks Enterprise default configuration that includes only the few fields that are absolutely essential for the system to run (see Customizing the Schema for more details on our default field set). This can be used for collection creation if desired. The package is called `essential.zip`. Both templates can be found in `$LWE_HOME/app/collection_templates`.

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.

Crawling and Indexing Configuration

This section describes how to configure crawling and indexing in LucidWorks Enterprise. It includes the following topics:

Supported Filetypes

LucidWorks Enterprise 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 LucidWorks Enterprise 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.

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	
JPG Images	image/jpeg	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	

How Documents Map To Fields

When LucidWorks Enterprise (LWE) 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://lucidimagination.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://lucidimagination.com
content-type	mimeType	html/text
title	title	The Company for Apache Lucene Solr Open Source Search | Lucid Imagination
body	body	The Future Of Search

And so on.

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

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. For more information on fields in LucidWorks Enterprise, see the Table of Fields in the section Customizing the Schema.

Customizing the Schema

LucidWorks Enterprise contains a `schema.xml` file for each collection, which is used to define the fields for the index (among other things). It is the same `schema.xml` file that is used with a Solr installation, however Lucid Imagination has added fields to support various features of LucidWorks Enterprise and to make it easier for users to get up and running. Not all users will need all fields, however, so they may want to trim the `schema.xml` file so it is easier to read. The following table shows the default fields, how they are used, and if they can be removed for local installations.

One of the primary added values of LucidWorks Enterprise is the integration of content crawlers for web sites, filesystems and other repositories of content. Many of the fields added to `schema.xml` 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 that uses them crawls new content. However, if not using the LucidWorks Enterprise 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 Lucid Imagination will add new fields to the `schema.xml` file to support future functionality.

Guidelines for Removing Fields from the Schema

Essential Fields

There are five essential fields which must be retained in `schema.xml` for LucidWorks Enterprise to continue to function. These are:

- id
- data_source
- data_source_name
- data_source_type
- text_all
- timestamp

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 Enterprise 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`:

- url
- title

- body
- author
- keywords
- description
- dateCreated
- lastModified
- pageCount
- mimeType
- author_display
- keywords_display
- timestamp

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 Enterprise features. They can be removed if those features are disabled or not in use.

Feature	Fields
Click Scoring Relevance Framework	click click_terms click_val
ACL	acl
Spell Check	spell
Auto Completion	autocomplete
Enterprise Alerts	timestamp

Crawler Fields

The crawlers included with LucidWorks Enterprise create fields in `schema.xml` that begin with attr_ and are used to store document-specific metadata during the crawl processes. They are not generally used otherwise by LucidWorks Enterprise (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 Enterprise 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	Can Be Deleted

acl	string	X	X			Storing Access Control List information.	Only if never using Access Control List (ACL) query-time document security.
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.	Yes, but automatically created by LucidWorks crawlers, so will be recreated at next crawl run.
author	text_en	X	X		X	Raw author pulled from documents. Used by default in the built-in Search UI.	Only if never using 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.	Only if never using 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.	Only if never using built-in auto-complete functionality.
batch_id	string	X	X			Identifies the batch that added the document.	Yes.
bcc	text_en	X	X			Used in processing email messages.	Yes. Will be added dynamically if an indexed document contains this field.
belongsToContainer	text_en	X	X			Used to store the URL of the archive file (.zip, .mbox, etc.) which contains the file.	Yes.

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.	Only if never using built-in Search UI.
byteSize	int		X			The size of the document.	Yes. Will be added dynamically if an indexed document contains this field and was crawled by the lucid.aperture crawler (local file systems and web sites).
cc	text_en	X	X			Used in processing email messages.	Yes. Will be added dynamically if an indexed document contains this field.
characterSet	string		X			The character set used for the document. Only populated if it is declared in the document (most commonly with HTML files).	Yes. Will be added dynamically if an indexed document contains this field.
click	string	X	X			Used with the Click Scoring Relevance Framework and contains the boost value.	Only if Click Scoring will not be used.
click_terms	text_ws	X	X			Used with the Click Scoring Relevance Framework and contains the top terms associated with the document.	Only if Click Scoring will not be used.

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.	Only if Click Scoring will not be used.
contentCreated	date	X	X			The creation date for the document, if available.	Yes. Will be added dynamically if an indexed document contains this field. However, it will not be added as a date, but a string, which may cause sorting issues if the field is used again later.
crawl_uri	string	X				A copy of the URL for the document.	Yes.
creator	text_en	X	X			The creator of the document, if available.	Yes. Will be added dynamically if an indexed document contains this field.
data_source	string	X	X			The ID of the data source that crawled this document.	No. Field is essential.
data_source_name	string	X	X	X		The name of the data source that crawled this document.	No. Field is essential.
data_source_type	string	X	X		X	The type of data source that crawled this document.	No. Field is essential.
dateCreated	date	X	X		X	The date the content was created, if available.	Only if never using built-in Search UI.

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.	Only if never using built-in Search UI.
email	text_en	X	X			Not currently used by any LucidWorks Enterprise crawlers.	Yes. Will be added dynamically if an indexed document contains this field.
fileName	text_en	X	X			The name of the file.	Yes.
fileSize	int	X	X			The size of the file.	Yes.
from	text_en	X	X			Used in processing email messages.	Yes. Will be created dynamically if indexing a document that contains this field.
fullname	text_en	X	X			Data in this field is mapped to "author".	Yes.
generator	text_en	X	X			The name of the software that generated the document, if available.	Yes.
id	string	X	X		X	Unique ID for the document.	No. Field is essential.
id_highlight	text_en	X	X			No longer used by LucidWorks Enterprise and will be removed in a later version.	Yes.
incubationdate_dt	date	X	X			Used in older Solr example documents.	Yes.
keywords	text_en	X	X		X	The keyword list from a Microsoft Office document.	Only if never using built-in Search UI.

keywords_display	comma-separated	X		X		Terms from the keyword field formatted for display to users.	Only if never using built-in Search UI.
lastModified	date	X	X		X	Date the content was last modified.	Only if never using built-in Search UI.
mimeType	string	X	X	X	X	The type of document (PDF, Microsoft Office, etc.).	Only if never using built-in Search UI.
name	text_en	X	X			Data in this field is mapped to "title".	Yes.
otherDates	date	X	X			Dates other than dateCreated or lastModified would be mapped to this field.	Yes.
pageCount	int	X	X		X	The number of pages in a Microsoft Office document such as Word or PowerPoint.	Only if never using built-in Search UI.
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.	Yes.
retrievalDate	date	X	X			Not currently used, but could be used for the date a web document was retrieved from its server.	Yes.
rootElementOf	text_en	X	X			Populated only for the root or initial document of a crawl.	Yes.
signatureField	string	X	X			Part of Solr's default schema.	Yes.

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.	Only if never using built-in spelling checker.
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.	No. Field is essential.
text_medium	text_en	X	X			Not currently used.	Yes.
text_small	text_en	X	X			Not currently used.	Yes.
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.	No, field is considered essential.
title	text_en	X	X			The title of the document.	Only if never using built-in Search UI.
to	text_en	X	X			Used in processing email messages.	Yes. Will be created dynamically if indexing a document that contains this field.
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.	Yes.

url	string		X		X	The URL to access the document.	Only if never using built-in Search UI.
username	text_en	X	X			No longer used and may be removed in a later version.	Yes.
weight	float	X	X			Example field that could be used for processing e-commerce data.	Yes.

Synonyms, Stop Words, and Stemming

LucidWorks Enterprise is built on the Solr infrastructure, including Solr's schema and field-type analyzer architecture. Although there are separate analyzers for indexing of documents and querying of the index, this topics concerns only the query analyzer. The Solr analyzer architecture is quite flexible and permits the administrator to specify a tokenizer and a list of filters. The tokenizer breaks a string into tokens, typically based on white space. Each filter than performs a relatively discrete step in the processing of tokens to produce the final form of a term that is ready to be matched against terms in the index.

Typical filter steps in term analysis include, but are not limited to:

- Tokenize the text stream based on white space
- Expand synonyms
- Remove stop words
- Remove punctuation and break apart multi-word terms (for example, CD-ROM)
- Translate upper case to lower case
- Translate accented characters into un-accented equivalents
- Stemming to translate plural words to singular and possibly reduce words to their root word

Synonym Expansion

The administrator can provide a synonym definition file for each type of text field in the schema. Synonyms can be either single terms or multi-term phrases. There can be an unlimited number of terms and phrases which are defined as synonyms.

If the Lucid query parser encounters any of those terms or phrases in a query term list, additional (optional) clauses will be automatically added to the user query so that the query will match either the specified term or phrase or any of the synonym terms or phrases.

A synonym can also be defined as a direct replacement so that the replacement term or phrase will be substituted in the query for the original term or phrase.

For the format of a synonyms file, see Term Analysis File Formats.

Stop Words

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 pairs and triples of the stop words and non-stop words.

There is the special case where a term list consists only of stop words. 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

The precise list of stop words is stored in a file that can be edited by the administrator to tailor it to the application.

Additionally, there can be separate stop word files for specific field types that may have special vocabulary requirements. For more information about setting up custom stop word files, see Term Analysis File Formats.

Stemming

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 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 to be 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.

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

The Lucid simple plural stemming filter is completely controlled by rules contained in a file, so additional rules and special cases can be easily added by the administrator.

For the format of the Lucid plural stemmer rules file, see the Lucid Plural Stemmer Rules File Format section.

If an alternative stemmer is desired, such as Lucid KStem, see the Choosing an Alternate Stemmer section.

Term Analysis File Formats

While LucidWorks Enterprise provides reasonable defaults for term analysis, you may desire more customization. To further customize your term analysis defaults, you can use the term analysis filters in Solr. Some of these term analysis filters are driven by rules contained in text files. These include:

- Stopwords
- Synonyms
- Stemming rules

You can edit synonyms and stop words in the Query Settings section of the Administration User Interface. The actual files can be found in the `$LWE_HOME/conf/solr/cores/collection1_0/conf` directory (assuming the default collection of `collection1`; if using multiple collections, use the collection name that matches the collection to be changed).

Stop Words File Format

The LucidWorks Enterprise Stop Words file format is the same as the Solr stopwords file format which is one term per line, as in:

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

Synonyms File Format

The LucidWorks Enterprise Synonyms file format is the same as the Solr synonyms file format. Blank lines and lines starting with pound are comments. Explicit mappings match any token sequence on the left side of "=>" and replace with all alternatives on the right side. These types of mappings ignore the expand parameter in the schema.

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.

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
```

Lucid Plural Stemming Rules File Format

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. But usually, general patterns cover wide classes of words.

This is mostly mapping plural to singular and primarily those ending with "s", but there are also verb forms not ending with "s" that fall under the same heuristic rules.

It is understood that this simple heuristic approach will misinterpret some words that should either not be stemmed or should be stemmed differently. The rules try to avoid removing "s" endings that are not plural (or verb conjugations), such as "alias" or "business."

Input token does not need to be lower case, but stemming change will be lower case.

The filter (factory) is named `com.lucid.analysis.LucidPluralStemFilterFactory`. It has a "rules" parameter which names the rules file. The default rules file is named `LucidStemRules_en.txt` and found in `$LWE_HOME/conf/solr/cores/collection1_0/conf`. It is expected that each natural language will have its own stemming rules file. This file is also specific to each collection.

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

1. Exclamation point indicates a comment or comment line to be ignored.
2. White space is extraneous and ignored.
3. Blank lines ignored.

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

Suffix of 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.

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.
```

Example Stemming Rules File

Here is the default `LucidStemRules_en.txt` file that ships with LucidWorks Enterprise:


```
? \! 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
```

Suppressing Stop Word Indexing

By default, LucidWorks Enterprise 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 a 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. Although this is not the preferred choice, it is supported by the Lucid query parser.

There are two modes for suppressing stop word indexing:

1. Skip mode: Completely ignore or skip them, as if they were not present.
2. Position increment mode: Do not store them in the index, but increment the position counter so as to leave a hole at the position of each stop word.

When skip mode is selected, the query parser will ignore or skip stop words in quoted phrases.

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.

Examples

Given these mini 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.

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 Enterprise 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_name>/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 Enterprise should be restarted. On some Windows machines, LucidWorks Enterprise 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:

```
<!--
<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.

Position Increment Mode

There is an additional option available for indexing of stop words. You can choose to suppress the actual indexing of stop words, but still increment the position in the document being indexed to leave empty spaces at the positions of the stop words. This enables queries with stop words in phrases (and in relevancy-boosting phrase terms) to query more precisely without either the storage overhead of indexing the actual stop words or the query-time overhead of searching for occurrences of the stop words. This is called position increment mode. It is enabled with an additional parameter to the stop filter factory in the index analyzer called `enablePositionIncrements` which is set to "true".

To enable this mode, edit the `StopFilterFactory` entry of the index analyzer (which was uncommented above) in `schema.xml` so it appears 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.

Troubleshooting Document Crawling

Errors during crawling will be recorded in the `core.<date>.log` file. You can find the `core.<date>.log` file 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. You can also view log events on the Server Log page (Status -> Server Log).

Documents may be skipped because there is not an extractor available for that file type, or because the file size exceeds the maximum set during crawl configuration. Skipped documents will not be recorded in the LucidWorksLogs collection. These 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

With each of the errors below, the exact cause cannot be determined. 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; for other data sources it will be whatever you have assigned as the document URI when the data source was 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, but that is primarily for file system crawls.

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.

Batch (Split) Crawling

Batch (or "Split") crawling 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.

To enable batch crawling, 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.

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.

Not all crawler controllers support batch crawling, or all batch crawling 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`.

To work with batches and batch jobs, use the Batch Crawling API. You can find details for all relevant API calls on that page. The basic workflow is as follows:

1. Create a data source using the Admin UI or Data Sources API.
2. Configure the data source for batch crawling by setting the `indexing` parameter to `false` using the Data Sources API. You can also set the `caching` and `indexing` parameters as described above.
3. Schedule a crawl for the data source.
4. Get the `batch_id` for the data source using the Batch Crawling API: `GET http://localhost:8888/api/collections/collection1/batches`.
5. Using the Batch Crawling 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`.

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

Crawling Windows Shares with Access Control Lists

About Crawling Windows Shares with Access Control Lists

LucidWorks Enterprise can crawl Windows shares and the Access Control Lists (ACLs) associated with shared files and directories. You can use this ACL information to limit users' searches to the content they are permitted to access.

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.

Additional System Requirements

Operating System

- Microsoft Windows Server 2003
- Microsoft Windows Server 2008

Microsoft Windows Active Directory

Active Directory is required for getting the user and group data.

Network Connections

Connections from LucidWorks Enterprise to the host sharing the directory:

- Server Message Block (SMB)
  - TCP Port 445
  - UDP Port 445

Connections from LucidWorks Enterprise to the host running Active Directory

- Lightweight Directory Access Protocol (LDAP)
  - TCP Port 389

For information on setting up LDAP in LucidWorks Enterprise for use with Windows Active Directory, see LDAP Integration

Other Requirements

- 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.

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 Enterprise. 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.

How To Set Up And Crawl An SMB Data Source With ACLs

1. Create an SMB data source. LucidWorks Enterprise (LWE) crawls in a recursive fashion, so you only need to provide a starting URL and the credentials required to access the share. LWE crawls SMB data sources with the filesystem crawler, so all of the features of the filesystem crawler are available, including `include_paths`, `exclude_paths`, `crawl_depth`, `max_size`. You can specify these features in the data source configuration.
2. Configure the ACL Filtering component.

> Configuration of ACL is also possible using the Filtering Results and Search Handler Component APIs.

3. Crawl the new SMB data source.

Suggestions for External Data Source Documents

In some cases, it may not be possible to use the crawlers embedded with LucidWorks Enterprise to index content, such as an email archive or a Web repository that's best accessed via an API. It's possible to use another process, such as using SolrJ, to feed documents directly to Solr. In that situation, LucidWorks Enterprise would not 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. There is a way to create an "external" data source and add fields to the document so LucidWorks Enterprise can treat the documents the same as documents found via the embedded crawlers. The data source can be added either via the Sources screen in the Admin UI or via the Data Sources API.

> External data sources cannot be scheduled or otherwise controlled with LucidWorks Enterprise. The process for feeding documents to the index is entirely controlled outside of the application. If regular updates to documents are required, a server-side cron job or other mechanism may need to be configured manually.

Once a data source of the type "external" is added, you need to complete the link between the documents and the new data source by adding several fields to the documents. At a minimum, the id of the newly created data source should be added as a `data_source` field, which can be found by either inspecting the URL of the data source when on the Data Source Detail screen, or by using the Data Source API. In order for the documents to appear in facets, also add the `data_source_name`. Adding the `data_source_type` and a unique ID per document in the `id` field will enhance the ability of LucidWorks Enterprise to display information about these documents in the Admin UI (such as document counts and other data).

The default search UI included with LucidWorks Enterprise 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.

For example, using the 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?commit=true -H "Content-Type: text/xml"
--data-binary '
<add>
<doc>
<field name="data_source">3</field>
<field name="data_source_name">Test External #1</field>
<field name="data_source_type">external</field>
<field name="id">http://www.wikipedia.com/mytest</field>
<field name="title">My First Document</field>
<field name="author">Tom Jones</field>
<field name="body">

This is the body text of my first test document.
It is being fed directly into the Solr update handler.
That is all for now.

The end.

</field>
</doc>
</add>'
```

Integration with External Pipelines

In some situations you may want to index content that has already been preprocessed with existing external systems, often employing several steps and modules (such as Natural Language Processing tools, dictionaries, Named Entity Recognition modules, custom tokenizers, stemmers, etc).

Standard field types supported in Solr do not provide this functionality. LucidWorks Enterprise adds a field type dedicated to support such scenarios.

`PreAnalyzedField` Type

The `PreAnalyzedField` type provides a way to index a serialized token stream, optionally with a stored value of the field that will be stored independently of the provided token stream.

When using standard Solr field types when a stored value is provided then the tokenized value will be derived by applying an analysis chain. But with `PreAnalyzedField` these two values are fully independent. That is, it is possible to provide a stored value that is completely unrelated to the token stream that will be indexed. The token stream part of the field is required, but the stored field is optional and may be omitted.

Field Content Format

Tokens to be indexed as well as the optional stored value are passed using a special serialization format, described in a pseudo-Backus-Naur Form syntax below:

```
content ::= (stored)? tokens
stored ::= "=" text "="
tokens ::= (token ((" ") + token)*)*
token ::= text ("," attrib)\*
attrib ::= name '=' value
name ::= text
value ::= text
```

There are some special characters that must be properly escaped in the "text" values using a back-slash '\' character. The following escape sequences are recognized (listed below with added double quotes for clarity):

- "\=" - literal equals sign character. This is the only escape sequence recognized in the stored part of the field, other escape sequences in this part are treated literally.
- "\ " - space character
- "\," - comma character
- "\/" - back-slash character
- "\n" - newline character
- "\r" - carriage-return character
- "\t" - horizontal tab character

Java Unicode escape sequences (e.g., \u0001) are not supported.

Supported Token Attributes

The following token attributes are supported, and identified with short symbolic names:

- i - position increment (integer)
- s - token offset, start position (integer)
- e - token offset, end position (integer)
- t - token type (string)
- f - token flags (hexadecimal integer)
- p - payload (bytes in hexadecimal format)

Token offsets are tracked and implicitly added to the token stream - the start and end offsets consider only the term text and whitespace, and exclude the space taken in the serialization format by token attributes.

Example Token Streams

This example contains only the token stream part (stored value is omitted). Note that token offsets are tracked even though they were not specified.

```
'one two three'
- stored: 'null'
- tok: '(term=one,startOffset=0,endOffset=3)'
- tok: '(term=two,startOffset=4,endOffset=7)'
- tok: '(term=three,startOffset=8,endOffset=13)'
```

In this example the additional unescaped whitespace is skipped and is not treated as a part of the token, but it affects token offsets.

```
' one  two   three '
  - stored: 'null'
  - tok: '(term=one,startOffset=1,endOffset=4)'
  - tok: '(term=two,startOffset=6,endOffset=9)'
  - tok: '(term=three,startOffset=12,endOffset=17)'
```

This example shows how to explicitly set starting and ending offsets, as well as position increments.

```
'one,s=123,e=128,i=22  two three,s=20,e=22'
- stored: 'null'
- tok: '(term=one,positionIncrement=22,startOffset=123,endOffset=128)'
- tok: '(term=two,positionIncrement=1,startOffset=5,endOffset=8)'
- tok: '(term=three,positionIncrement=1,startOffset=20,endOffset=22)'
```

This example presents various combinations of escape sequences and their interpretation. Of particular interest may be the first token, with value of "one ," and type equal to ",", as well as a series of invalid escapes and unescaped special characters that are ignored

```
'\ one\ \,,i=22,t=\, two\=

\n,\ =\   \'
- stored: 'null'
- tok: '(term= one ,,positionIncrement=22,startOffset=0,endOffset=6)'
- tok: '(term=two=


,positionIncrement=1,type=,,startOffset=7,endOffset=15)'
- tok: '(term=\,positionIncrement=1,startOffset=17,endOffset=18)'
```

This example shows an unusual case of empty tokens with non-empty attributes.

```
',i=22 ,i=33,s=2,e=20 , '
  - stored: 'null'
  - tok: '(term=,positionIncrement=22,startOffset=0,endOffset=0)'
  - tok: '(term=,positionIncrement=33,startOffset=2,endOffset=20)'
  - tok: '(term=,positionIncrement=1,startOffset=2,endOffset=2)'
```

This example illustrates the use of the optional stored value part.

```
'=This is the stored part with \=
\n    \t escapes.=one two three\ '
- stored: 'This is the stored part with =
\n    \t escapes.'
- tok: '(term=one,startOffset=0,endOffset=3)'
- tok: '(term=two,startOffset=4,endOffset=7)'
- tok: '(term=three ,startOffset=8,endOffset=13)'
```

Empty (non-null) stored value, and no token stream.

```
'=='
- stored: ''
- (no tokens)
```

Example of passing just the stored value. If the Solr schema indicates that this field should be analyzed then this value will be passed as usual through the declared analysis chain in order to produce a token stream.

```
'=this is a test.='
  - stored: 'this is a test.'
  - (no tokens)
```

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 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>
```

Query and Search Configuration

This section describes how to configure query and search options in LucidWorks Enterprise. It includes the following topics:

Enterprise Alerts

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

Overview of Alerts Functionality

1. The user does a search, and clicks the link under the search box to "Create new alert".
    a. The user configures the alert and notification settings, including how often to run the alert (`period`) and an email address to send alert notifications.
    b. LucidWorks Enterprise 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 Alerts API documentation.

Types of Alerts

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.

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. The default LucidWorks Enterprise schema already defines this timestamp. If you are using or extending the default LucidWorks Enterprise schema, you have met the timestamp requirements for alerts.

Spell Check

Integrated query spell checking is bundled with LucidWorks Enterprise, with the option to integrate third-party enhanced spell checking capabilities. Spell checking and auto-complete 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 the all-too-familiar case of working with messy data written by a variety of authors of varying quality. One author may spell a word one way, while another author spells it a different way. Meanwhile, 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.

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.

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

1. Enable spell checking by accessing the Query 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 if preferred.
3. Crawl your content.

What makes for a good spell-checking field?
A good spell checking 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 is probably not.

Using the Spell Checker Results

The spell checker is integrated into LucidWorks Enterprise via Solr's `SearchComponent` plugin framework. Specifically, it is set up via the `SpellCheckComponent`. Thus, when enabled, spell suggestions are returned with query results inline in the query response, as documented on the SpellCheckComponent page.

Auto-Complete of User Queries

Integrated query auto-complete is bundled with LucidWorks Enterprise as 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.

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.

Because the creation of auto-complete indexes may take some system resources, LucidWorks Enterprise does not create this index by default. Auto-Complete indexing jobs must be scheduled using the Settings interface under the Indexing tab of the Admin UI (or via the Activities API) before query suggestions will appear for users. You can configure LucidWorks Enterprise to create this index automatically after crawling content.

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

1. Enable auto-complete by accessing the Querying Settings tab 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 tab and choosing "Index for autocomplete". The Fields API can be used instead if that is your working mode.
3. After crawling some content, create the "autocomplete" index by accessing the Index Settings tab of 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.

What makes for a good auto-complete field?
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 is probably not.

Automatic Creation of Auto-Complete Indexes

By default, LucidWorks Enterprise does not build the indexes for auto-complete each time documents are added to the index (e.g., from crawling new content) because doing so may have performance implications in a production environment with a large index. However, LucidWorks Enterprise 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.

For Auto-Complete, find the following section in the `solrconfig.xml` file for the 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>
```

Auto-Complete data is persisted in a directory defined in the `storeDir` property (relative to the collection's `dataDir` directory), so the default setting of `buildOnCommit` to false may be sufficient, especially if the source of the auto-complete data is not the main index (empty `sourceLocation` property) but an external file that changes rarely.

Document Highlighting

The document highlighting service (RequestHandler) implemented by `DocumentHighlighterRequestHandler` performs a similar function to the built-in Solr highlighter. The key difference is that while the Solr highlighter operates on a list of results returned from the index in response to a query, the Document Highlighter described here can perform highlighting of arbitrary content in one of many supported formats, and the content can be either submitted as a part of request or it may be retrieved from the current index.

Document Highlighter can return the original text, the highlighted text, and offsets of each matching span of terms, according to request parameters.

Overview of Request Parameters

Below is a list of parameters accepted by this RequestHandler. Parameters are mandatory unless marked as optional, in which case a default value will be used.

- `dochl.q`: the query to use for finding highlights. Note: the query parser can be defined using the regular `qt=` parameter, for example: `qt=lucene`.
- `dochl.mode={highlight | offsets | both}`: (optional, default is `highlight`). When mode is set to `highlight`, only the text with highlighted sections will be returned. If mode is set to `offsets`, only the matching term spans and their offsets will be returned. When mode is set to `both`, both highlights and offsets will be returned.
- `dochl.source={solrcell | stored | xml | text}`: the source of the text to be highlighted.

- `solrcell`: retrieve text from a specified content stream (required) using `ExtractingRequestHandler`, commonly referred to as SolrCell, and optionally use a subset of extracted data specified in an XPath expression (see below for more details). All parameters specific to SolrCell, such as field mapping, can be used in combination with this option.
- `stored`: obtain text from stored fields of a specified document in the current index (see below for additional parameters).
- `xml`: use the text content of the provided XML content stream, and optionally use a subset of the XML selected by an XPath expression.
- `text`: use the provided plain text content stream.

LucidWorks Enterprise (LWE) performs text analysis based on the field names in the supplied content. The default field name, if none are present, is `body`. The default field type, which determines the analysis chain, is obtained based on this field name from the current schema. It can be also overridden using `dochl.ft.NAME` parameters, for example `dochl.ft.body=text_en`. A special name of `dochl.ft.*` can be used to set an override for any other unspecified field.

- `dochl.xpath`: (optional, default is none, which selects the whole document). This XPath expression is used to select matching elements in the input XML document (either provided directly as XML content stream, or extracted using SolrCell). Only text content from the selected elements will be processed. In the case of multiple matches, LWE processes each match separately, and returns text, highlight, and offset nodes arranged in the order of matching.
- `dochl.filename`: (optional, default is none) can be used to help with content extraction.
- `dochl.includeOrigText`: (optional, default is true when `dochl.mode=offsets` and false otherwise). If true then returns also the original source text.
- `dochl.beginMarker`: (optional, default is) characters to use as the start of the highlighted span.
- `dochl.endMarker`: (optional, default is) characters to use as the end of the highlighted span.
- `dochl.docId`: (required when `dochl.mode=stored`) unique key (without field name) that identifies the document to obtain the stored fields from.
- `dochl.fl`: (optional, considered when `dochl.mode=stored`) if present, specifies a comma-separated list of fields to highlight. If absent, the regular `f` parameter is considered. If both are absent then LWE uses all stored fields.

Examples

Request
`curl "http://localhost:8888/solr/collection1/dochl?dochl.q=test&dochl.source=text& / stream.body=This+is+a+test+example&dochl.mode=both&dochl.includeOrigText=true &dochl.beginMarker=&dochl.endMarker="`

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <lst name="responseHeader">
    <int name="status">0&</int>
    <int name="QTime">91</int>
  </lst>
  <lst name="results">
    <lst name="0">
      <lst name="offsets">
        <str name="test">10-14</str>
      </lst>
      <str name="highlighted">This is a &lt;span&gt;test&lt;/span&gt; example<str>
      <str name="text">This is a test example</str>
    </lst>
  </lst>
</response>
```

Request

```
curl "http:
//localhost:8888/solr/collection1/dochl?dochl.q=test&amp;dochl.source=solrcell&amp;dochl.mode=bot
; \-F "myfile=test.pdf"
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader">
<int name="status">0</int>
<int name="QTime">61</int>
</lst>
<lst name="results">
<lst name="body">
<lst name="offsets">
<str name="test">10-14</str>
<str name="test">40-44</str>
</lst>
<str name="highlighted">This is a <b>test</b> example.  This is a second <b>test</b> example.
</str>
</lst>
</lst>
</response>
```

Request

```
curl "http:
//localhost:8888/solr/collection1/dochl?dochl.q=test&amp;dochl.source=xml&amp;dochl.mode=both&str
version='1.0'+encoding='UTF-8'?&gt;&lt;top&gt;&lt;p&gt;first+test&lt;/p&gt;&lt;p&gt;second+test+e
\-H
"Content-Type: text/xml"
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader">
<int name="status">0</int>
<int name="QTime">76</int>
</lst>
<lst name="results">
<lst name="body">
<lst name="offsets">
<str name="test">17-21</str>
</lst>
<str name="highlighted">first testsecond <b>test</b> example</str>
</lst>
</lst>
</response>
```

Search User Interface Customization

Client Interaction with LucidWorks Enterprise

LucidWorks Enterprise is built upon Solr and supports it natively. You may find it helpful to review the Solr Reference Guide for Solr 1.4 if you plan to interact directly with Solr.

Lucid Imagination has defined a special Solr search request handler, `/lucid`, for LucidWorks Enterprise. This request handler is preconfigured, and you can see the default settings returned in each response's responseHeader/params data structure. These parameters can be overridden by the client request, but overriding some parameters may adversely affect the expected search results.

To search using the LucidWorks Enterprise request handler, simply point your HTTP client/browser to http://localhost:8888/solr/collection1/lucid?q=some+query. LucidWorks Enterprise 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.

LucidWorks Enterprise uses Solr's default SearchHandler, adding a custom query parser, role filtering, spelling checking, unsupervised feedback, and so on. Solr's common query parameters apply, along with Solr's built-in highlighting and other built-in search component handling.

Any request sent to Solr must include the collection name. In the above example URLs, "collection1" refers to the default LucidWorks Enterprise collection. If you have configured multiple collections, replace "collection1" with the appropriate collection name.

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 Solr's wt (writer-type) documentation.

Debug Mode

To enable detailed debug information in the LucidWorks Enterprise Search UI, add `&debug=on` to the request. Debug information includes detailed query parse information and Lucene explanations.

Integrating LucidWorks Enterprise with PHP

You can interact with LucidWorks Enterprise using PHP with Solr's serialized PHP format. Here is some sample code to issue the request and produce a response:

```
<?php
  $url = "http://localhost:8888/solr/collection1/lucid?q=query&wt=phps";
$serializedResult = file_get_contents($url);
  $response = unserialize($serializedResult);
?>
```

The $response variable holds a nested associative array. The complete structure can be dumped and seen clearly using `print_r($response);`

The total number of search results is available as `$response['response']['numFound']`. An array of documents, in sort order, is `$response['response']['docs']`.

You can reverse engineer Solr request syntax by looking at LucidWorks Enterprise queries written to the `lucid.YYYY_MM_DD.log`. Look for lines like this:

```
INFO: [collection1] /lucid
wt=xml&start=0&sort=random1+desc&q=example&version=2.2 0 17
```

The `wt=xml&start=0&sort=random1+desc&q=example&version=2.2` string represents the parameters sent to Solr. For PHP integration, use `wt=phps` instead of `wt=xml`.

Highlighter Usage

The `/lucid` request handler defaults the highlighter `pre` and `post` query term markers with special unique tokens in order to ensure HTML escaping is done properly. If you are using the LucidWorks Enterprise built-in search interface, these markers are handled automatically. If you use a custom client with LucidWorks Enterprise against the `/lucid` request handler, you will need to account for these special markers by either replacing them in the client tier or overriding these parameters. The default markers are `LWE_BEGIN_HIGHLIGHT` and `LWE_END_HIGHLIGHT`. To override these parameters, set the `hl.simple.pre` and `hl.simple.post` on the client request.

Performing a Search Against LucidWorks Enterprise

LucidWorks Enterprise has its own Search UI, but if you are going to build your own user interface, or your own application to access the data stored in LucidWorks Enterprise, you will need to access the underlying engine directly.

LucidWorks Enterprise 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 Enterprise 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=userScreenName&hl=
true&hl.fl=text
```

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 Enterprise 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=userScreenName&hl=
true&hl.fl=text
```

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=userScreenName:cloudExpo
```

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.

Understanding and Improving Relevance

As mentioned earlier, relevance is one of the most complex components of a search engine implementation. Users may have differing opinions about what the best match for their query is, but this is generally because most user's queries are too short for the system to gain a full understanding of the intention behind the query. However, there are some ways to assess relevance and adjust how documents are scored to improve ranking.

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 Enterprise 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.

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 Enterprise currently ships with 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 Enterprise 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 Stemming and Lucid Plural

Stemmer Rules File Format sections.) 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 at http://localhost:8888/solr/collection1/admin/analysis.jsp.

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 Solr Wiki section on the Schema. This currently can be set up by editing the `schema.xml` file and restarting LucidWorks Enterprise.

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.

Stopwords

As discussed previously, removing stopwords (such as a, the, of, and so on) 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 Enterprise can remove stopwords at indexing time, it does not by default. Instead, LucidWorks Enterprise 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 recall (the ability to get more relevant documents) rather than relevance ranking or the exclusion of irrelevant documents. For instance, a user query contain "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 Enterprise, 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 Enterprise 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 Enterprise 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. For details on how to configure the component, see the Solr Wiki section titled QueryElevationComponent. It can be configured by editing `solrconfig.xml` manually, or using the Settings REST API.

QueryElevationComponent Requires Restart
Configuring the QueryElevationComponent requires restarting LucidWorks Enterprise.

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.

The first thing to do when debugging is to run the query in debug mode by appending `&debug=on` to the request. In the default LucidWorks Enterprise Search UI, links will appear under each search result to "toggle explain"; clicking that will show the scoring of each document for the query.

By turning on debug mode, it is then possible to see why an individual result scored the way it did, as is the case here:

```
1: 2.4688563 = (MATCH) boost((title:lucene^5.0|text_all:lucene)~0.01,1000.0/(1.0*float
(rord(lastModified))+1000.0))^2^, product of:
2:   2.6342697 = (MATCH) max plus 0.01 times others of:
       2.6313386 = (MATCH) weight(title:lucene^5.0 in 103), product of:
         0.99999946 = queryWeight(title:lucene^5.0), product of:
3:         5.0 = boost
           5.26268 = idf(docFreq=2, numDocs=213)
           0.03800343 = queryNorm
         2.63134 = (MATCH) fieldWeight(title:lucene in 103), product of:
4:         1.0 = tf(termFreq(title:lucene)=1)
5:         5.26268 = idf(docFreq=2, numDocs=213)
6:         0.5 = fieldNorm(field=title, doc=103)
       0.2931021 = (MATCH) weight(text_all:lucene in 103), product of:
         0.102523014 = queryWeight(text_all:lucene), product of:
           2.6977305 = idf(docFreq=38, numDocs=213)
           0.03800343 = queryNorm
         2.8588905 = (MATCH) fieldWeight(text_all:lucene in 103), product of:
           6.78233 = tf(termFreq(text_all:lucene)=46)
7:         2.6977305 = idf(docFreq=38, numDocs=213)
           0.15625 = fieldNorm(field=text_all, doc=103)
8:   0.9372071 = 1000.0/(1.0*float(rord(lastModified)=67)+1000.0)
```

[1] The overall score for the document given the expanded query.

[2] The score from the main query

[3] A boost of 5.0 is applied to the title.

[4] The term frequency (tf) of the word in the Document. The word "lucene" occurs once in the title Field of the Document.

[5] The IDF (inverse document frequency) for the word in the title Field. The word "lucene" occurs in the title Field in 2 Documents out of a total of 213 Documents.

[6] Apply length normalization to the Field.

[7] The IDF for the word in the text_all Field. The word occurs in 38 documents out of 213.

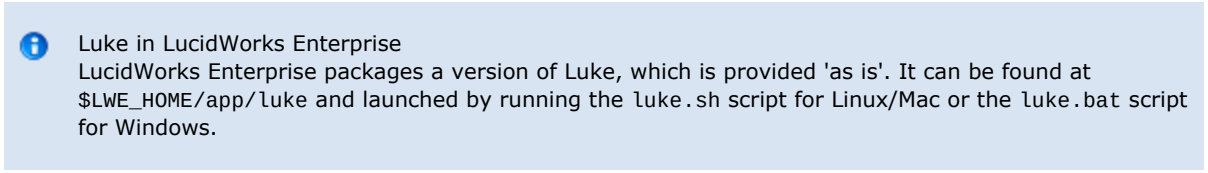
[8] The default query for LucidWorks Enterprise can boost more recent documents higher than older documents.

A Word on Scoring
Scores for results for a given query are only useful in comparison to other results for that exact same query. Trying to compare scores across queries or trying to understand what the actual score means (for example, 2.34345 for a specific document) may not be an effective exercise. However, understanding the components that were factored in to make that score is a different story.

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 Enterprise, launch it using the script located in the `luke` directory under the installation.

Once Luke is launched, point it at the LucidWorks Enterprise index directory (`$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.



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.

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.

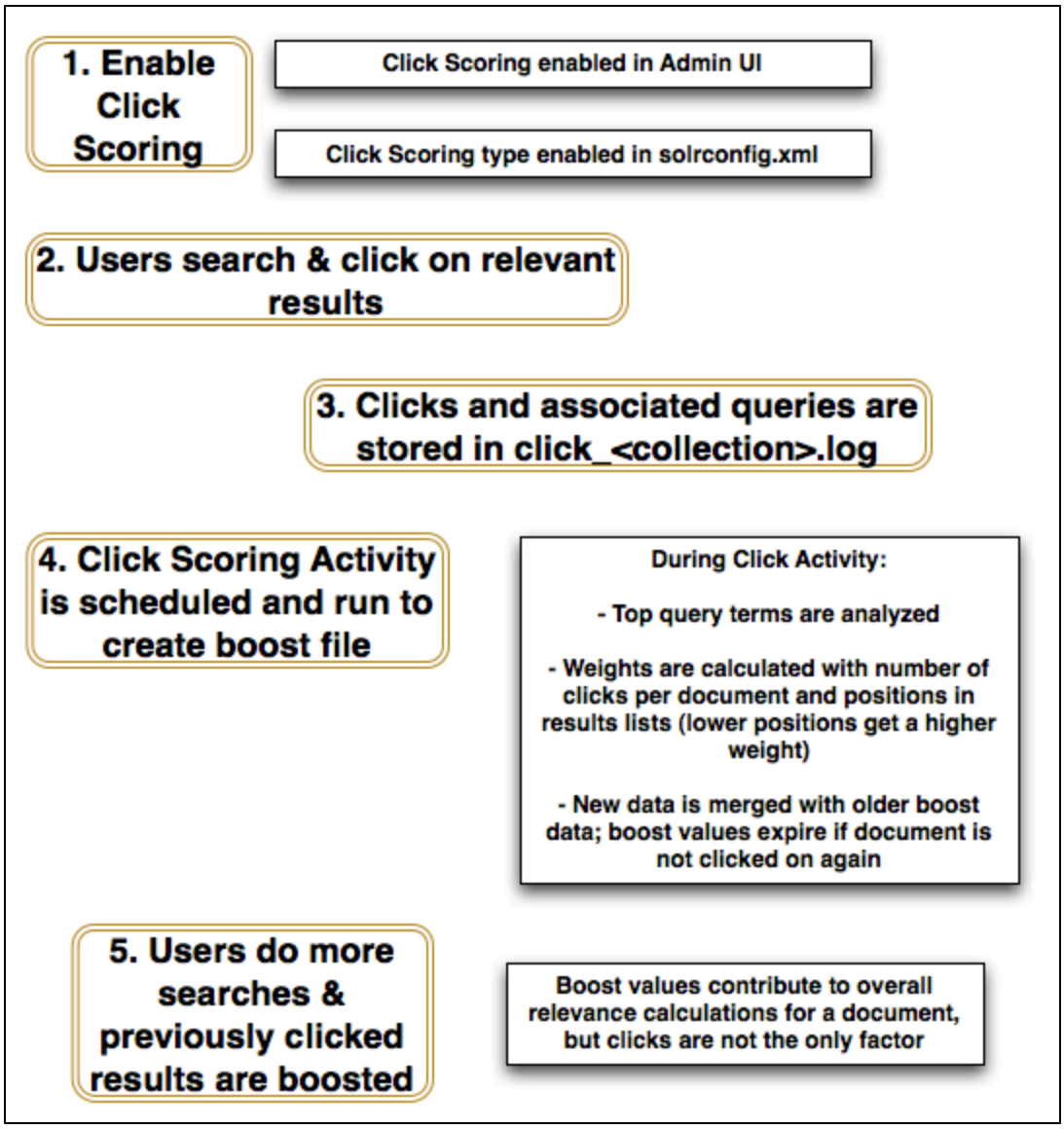
LucidWorks Enterprise includes a component that enables administrators to add this type of information to the index. This component is referred to as the Click Scoring Relevance Framework (or Click Scoring, for short). The framework includes tools for query log collection, processing of logs and robust calculation of the data used to boost certain documents. It is possible to supply boost data prepared without Click Scoring tools included with LucidWorks Enterprise, 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 Querying Settings section of the Admin UI.

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:



This reinforcement of ranking and terms needs to be counterbalanced by "expiration" of past history of click-through events, to avoid situations when documents that are selected many times start to permanently dominate the list of results and 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 Enterprise Search User Interface. If LucidWorks Enterprise is used with a custom search user interface that directly uses the LucidWorks Enterprise REST API and the underlying Solr API, this part of Click Scoring needs to be reimplemented. 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-<collectionName>.log`, and the name of the log file corresponds to the collection name (for example, `click-collection1_0.log` for the default collection in LucidWorks Enterprise).

When using Index Replication this log data is not replicated to slave nodes. However, the latest version of the aggregated boost data (see below) 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.

Maintenance of Historical Click Data

Each time click processing runs it saves a copy of the click log (by default this is in `$LWE_HOME/data/solr/cores/<collectionName>/data/click-data/`). Other data produced during click processing is also stored in that location.

Over time the amount of data collected there could be significant. LucidWorks Enterprise 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.

Collection of User Clicks
If Click Scoring is not enabled, LucidWorks Enterprise does not gather information about user document clicks.

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 scheduled to run periodically using the Administration User Interface by setting a recurring activity in the Indexing Settings screen of the Admin UI. It results in the creation of calculated click boost data, which is by default located in `$LWE_HOME/data/solr/cores/<collectionName>/data/click-data/current`.

Integration of click boost data with main index

If Click Scoring is enabled and boost data exists, then when new documents are indexed, an index optimization is run, or a full re-index is executed, the boost data is integrated on the fly with the main index. 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 Enterprise is configured to use Click Scoring data as an additional field in a `lucid` query parsing (and extended variant of `dismax`). 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 Administration User Interface, 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

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/<collectionName>/data/click-data/current/boost.data`.

```
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 Enterprise 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 Enterprise is configured to use Index Replication the `boost.data` file will also be automatically replicated. Due to the internal limitations of the 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.

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>
```

Multilingual Indexing and Search

LucidWorks Enterprise 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 using Lucene's analyzers package. Additionally, by purchasing the Basis Technology add-on plugin, it is easy to add even more advanced capabilities for working with languages.

Multilingual capabilities are provided by Lucene's analysis process (see http://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters). Since Lucene is built on Java, which is Unicode enabled, many multilingual issues are handled automatically by LucidWorks Enterprise and Solr. In fact, the main issues with multilingual search are mostly the same issues for working with any single/native 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 a decision to be made on 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 Enterprise, 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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise as described in the Solr wiki section on SolrPlugins.

Basis Technology Multilingual Capabilities

The optional Basis Technology Rosette Linguistics Platform (RLP) provides extensive multilingual capabilities, many of which have been integrated into LucidWorks Enterprise. RLP provides rich algorithms for morphological analysis, segmentation and other language needs. RLP is currently available for most, but not all, LucidWorks Enterprise platforms (see http://basistech.com/products/platforms/ for a list of RLP platforms and system requirements in this Guide for a list of LucidWorks Enterprise platforms.)

Obtaining the Rosette Linguistics Platform (RLP)

To purchase the Basis Technology RLP package, contact Lucid Imagination. Contact Lucid Support for instructions on installation and configuration.

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 Enterprise implements security for each of these as follows:

Network access: Because the components of LucidWorks Enterprise (LWE-Core, Admin UI, Search UI, and Alerts) 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 can be open while blocking LWE-Core and Alerts ports can be easily blocked. The chapter Securing LucidWorks Enterprise describes this process in more detail. Note that if you are using LucidWorks Enterprise's document authorization features, this step is particularly important, as direct access to the underlying Solr application can circumvent these measures.

User authentication: LucidWorks Enterprise supports LDAP binding for user authentication, so an administrator can create roles or groups on an external LDAP server, then use them within LWE to control access to UI functionality or sets of documents. The chapter LDAP Integration describes how to configure LDAP for LucidWorks Enterprise.

UI authorization: LucidWorks Enterprise controls access to the Admin UI, the Search UI, and Alerts. The chapter LDAP Integration discusses how to use the Admin UI to configure these access levels in order to give different LDAP users and groups authorization to use each of these different functions.

Document authorization: LucidWorks Enterprise 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 Enterprise 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 Enterprise

Restricting Access to LucidWorks Enterprise

LucidWorks Enterprise consists of two components: LWE Core and LWE UI.

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 Enterprise-enhanced capabilities such as job scheduling, system status, and indexing file, web, and database sources. 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 or the HTTP Connector's `address` attribute for the Tomcat container.

You can also restrict direct access to LucidWorks Enterprise 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 by the application, just as one would do with a typical relational database. If you are using LucidWorks Enterprise's built-in search filters or document-level authentication, you must prevent access to LWE 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 Enterprise User Interfaces

LucidWorks Enterprise has two built-in authorizations to control user access:

- ADMIN allows users to access any part of the LucidWorks Enterprise 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 LucidWorks Enterprise's Access

The privileges of the LucidWorks Enterprise process and the rights that process has to access documents for indexing are crucial to its proper operation. Generally, you want LucidWorks Enterprise to be able to access all documents within a particular folder or from a particular web site. The built-in LucidWorks Enterprise file and web crawling data sources will index any specified document, as long as the LucidWorks Enterprise 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.

In the case of a database data source, the LucidWorks Enterprise database integration requires that you provide credentials in the form of a JDBC connection string. These credentials determine the visibility of data for LWE.

Enabling SSL

LucidWorks Enterprise

To configure the LWE-Core and LWE-UI components for SSL, set each component to `https://` and specify the port you wish during the installation process. If you have already installed LucidWorks Enterprise, you can set these values in the `$LWE_HOME/conf/master.conf` file:

```
# COMPONENT LWE-Core - LWE-Solr + LWE REST API.
# -----
lwecore.enabled=true
lwecore.address=https://127.0.0.1:8443

 ...

# COMPONENT LWE-UI - Admin and Search UI as well as Alerts
# -----
lweui.enabled=true
lweui.address=https://127.0.0.1:8443
```

After configuring the Core and UI components to `https`, set `ssl.auth_require_secure: true` in the `$LWE_HOME/conf/lwe-core/defaults.yml` file.

Jetty

Both the LWE-Core and LWE-UI components run under Jetty, which can be configured to use SSL. Each component needs to be enabled separately, although the process for each component is the same.

LWE-Core

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 `keystore`, `password`, `keyPassword`, `truststore` and `trustPassword` parameters should be configured.

```
<Configure id="Server" class="org.mortbay.jetty.Server">
<!--
  <Call name="addConnector">
    <Arg>
      <New class="org.mortbay.jetty.security.SslSocketConnector">
        <Set name="Port">8443</Set>
        <Set name="maxIdleTime">30000</Set>
        <Set name="keystore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-core/etc/keystore</Set>
        <Set name="password">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
        <Set name="keyPassword">OBF:1u2u1wml1z7s1z7a1wnl1u2g</Set>
        <Set name="truststore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-core/etc/truststore</Set>
        <Set name="trustPassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
        <Set name="NeedClientAuth">true</Set>
      </New>
    </Arg>
  </Call>
-->
</Configure>
```

LWE-UI

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 `keystore`, `password`, `keyPassword`, `truststore` and `trustPassword` parameters should be configured.

```
<Configure id="Server" class="org.mortbay.jetty.Server">
<!--
  <Call name="addConnector">
    <Arg>
      <New class="org.mortbay.jetty.security.SslSocketConnector">
        <Set name="Port">8443</Set>
        <Set name="maxIdleTime">30000</Set>
        <Set name="keystore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-ui/etc/keystore</Set>
        <Set name="password">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
        <Set name="keyPassword">OBF:1u2u1wml1z7s1z7a1wnl1u2g</Set>
        <Set name="truststore"><SystemProperty name="lucidworksConfHome"
/>/jetty/lwe-ui/etc/truststore</Set>
        <Set name="trustPassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
        <Set name="NeedClientAuth">true</Set>
      </New>
    </Arg>
  </Call>
-->
</Configure>
```

For more information about configuring Jetty to use SSL, see http://docs.codehaus.org/display/JETTY/How+to+configure+SSL.

Configuring Mutually Authenticated SSL

LucidWorks Enterprise 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 LWE portions of SSL functionality can be configured by using the SSL Configuration API.

When configuring LucidWorks Enterprise 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

Reviewing logging events from the LucidWorks Enterprise log files (either `core.YYYY_MM_DD.log` or `ui.log`) may provide some hints about what is going on.

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 easily used to verify ssl configuration:

```
curl --cacert <ca.crt> --key <host.key> --cert <client.crt>
https://localhost:8443/solr/admin
```

If you can see the HTML for the Solr Admin page after running that command then SSL is properly set up.

Certificate Management

LucidWorks Enterprise 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 Enterprise. 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 Enterprise and potentially repeated on all nodes where LucidWorks Enterprise is running.

Restricting Access to Content

LucidWorks Enterprise provides two ways to restrict access to content through based on user identity: search filters, and document-based access.

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 Enterprise Administration User Interface 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.

Document-based Authorization

An application can enforce document visibility controls in front of LucidWorks Enterprise simply by adding fields to each document that represent either 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 Enterprise 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>
```

LDAP Integration

LucidWorks Enterprise 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.

Self-signed Certificates
LucidWorks Enterprise LDAP functionality is not compatible with self-signed or custom certificates.

To configure LucidWorks Enterprise to use LDAP for user authentication, edit the LDAP Configuration File. To enable LDAP in LucidWorks Enterprise, set the environment variable `lweui.ldap.enabled=true` in the `master.conf` file (`$LWE_HOME/conf/master.conf`).

Map a Valid LDAP User/Group to Authorization Before Enabling LDAP
Because LucidWorks Enterprise's 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.

For standard LDAP integration, the LDAP administrative user only needs permissions to query the LDAP server for users and groups. Lucid Imagination strongly recommends you create an LDAP admin user with only the necessary minimal user and group querying permissions for use with LucidWorks Enterprise.

LucidWorks Enterprise (LWE) also allows you to authenticate users without LDAP administrative credentials. This method is called "queryless" authentication, because LWE does not query the LDAP directory for user information. Rather, LWE 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 LWE cannot find members of a group, only individual users.

LDAP Configuration File

The main configuration file for configuring LDAP is `$LWE_HOME/conf/ldap.yml`. The default settings must be modified as needed for LucidWorks Enterprise to connect to the LDAP server and query the database for user authentication.

After the file has been edited, restart the server.

Here is the default content of the `ldap.yml` file. Note that the file includes sample configurations for standard LDAP authentication, queryless authentication, and Microsoft ActiveDirectory integration for use with Windows Shares data sources.

```
##########################################################
# 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

# Attribute Definitions
#
# (see below for sample configs)
# 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:
#     $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: login to use for searching the directory - not used with query-less
authentication
# admin_password: password to use for searching the directory - not used with query-less
authentication
# ssl: enable/disable SSL
# group_base: Search base for group queries. Not used with query-less 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:
#     $ATTR will be substituted with the value of 'group_name_attribute' from above
#     $USER will be substituted with the logged-in user's fully-qualified LDAP DN
#   search is performed using 'group_base' as a search base
#   Note: The default query ($ATTR=$USER) does not specify the object type for groups
#     several different group object types are common, e.g. group, groupOfNames,
groupOfUniqueNames.
#     Therefore, non-group objects may also match if they contain an a matching attribute

# Sample Configurations
#
```

```
# Basic Configuration:
# production:
#   host: localhost
#   port: 389 # 636 for SSL
#   attribute: uid
#   base: dc=xyz,dc=corp,dc=com
#   admin_user: cn=Manager,dc=xyz,dc=corp,dc=com
#   admin_password: secret
#   ssl: false
#   group_base: ou=groups,dc=xyz,dc=corp,dc=com
#   group_membership_attribute: uniqueMember # often this is just 'member'
#   group_name_attribute: cn
#
# Basic Queryless Authentication:
# Notes:
#   Disable "Use admin credentials in UI /settings/edit page", and restart
#   Group lookup is not possible in this mode
# production:
#   host: localhost
#   port: 389 # 636 for SSL
#   attribute: uid
#   base: ou=users,dc=xyz,dc=corp,dc=com
#   ssl: false
#   # all other attributes are invalid with queryless authentication
#
# Microsoft ActiveDirectory
# production:
#   host: localhost
#   port: 389 # 636 for SSL
#   attribute: userPrincipalName # AD uses userPrinciapalName for email address, e.g.
fred@domain.com
#   base: dc=domain,dc=com
#   admin_user: cn=Manager,dc=corp,dc=com
#   admin_password: secret
#   ssl: false
#   group_base: ou=groups,dc=corp,dc=com
#   group_membership_attribute: member
```

```
#   group_name_attribute: name
```

The attribute definitions included in the `ldap.yml` file are as follows:

Attribute	Definition
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

LWE supports two different methods of mapping users to groups:

- Listing users as attributes in group directory entries
- Listing groups as attributes in user directory entries

You should only use one of these methods at a time. Your configuration should contain only one of the two blocks of LDAP user/group mapping settings.

Manual User Management

LucidWorks Enterprise 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.

Solr Direct Access

LucidWorks Enterprise 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 Enterprise exposes the Solr interface directly. This means that applications can leverage both Solr's power and openness and LucidWorks Enterprise's ease of use. This chapter examines common needs of application developers in getting custom data into LucidWorks Enterprise/Solr and searching documents that have been indexed. There is a wealth of information about Solr at Solr's wiki which we intentionally do not duplicate within the LucidWorks Enterprise documentation.

Solr Version
For information about the Solr version included in this release of LucidWorks Enterprise, see the SOLR_VERSION.txt file in `$LWE_HOME/app/solr-src/SOLR_VERSION.txt`. You can also get detailed Solr version information for all releases of LucidWorks Enterprise from our public github fork here: https://github.com/lucidimagination/lucene-solr. To see information for a specific release, select the tag for that release from the Switch Tags drop-down list.

The primary difference in accessing Solr's example instance and LucidWorks Enterprise is the base URL. Solr's example application is accessed by default at `http://localhost:8983/solr/`, whereas the LucidWorks Enterprise default collection instance of Solr is rooted at `http://localhost:8888/solr/collection1/`. If using multiple collections, replace `collection1` with the correct collection name.

Indexing Solr XML

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>
```

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

One way to integrate LucidWorks Enterprise 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 Enterprise 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 Enterprise 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'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 Enterprise.

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

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

Searching Solr

Search results can be readily obtained through an HTTP GET request to Solr. For LucidWorks Enterprise, it is as straightforward as this URL: `http://localhost:8888/solr/collection1/lucid?q=my+query&role=DEFAULT`.

The "collection1" part of that path is the Solr core name of the default LucidWorks Enterprise collection. If using another collection, modify the URL to replace "collection1" with the correct collection name. The "lucid" part of the request is a custom Solr request handler mapping that LucidWorks Enterprise has preconfigured (called the Lucid query parser). The "role" part of the request specifies the role by which results are filtered and returned. If you do not include a role in the request, LucidWorks Enterprise will return not search results. Entering that URL in a browser results in XML being returned and (generally, depending on browser) rendered somewhat legibly. Any HTTP client can be used.

There are many options to requesting search results from Solr (and thus LucidWorks Enterprise). For more information, start with Solr's standard request handler and common query parameters.

To ensure that your query is indexed for UI reporting, include the `req_type=main` parameter in your query URL.

Programmatic access to Solr/LucidWorks Enterprise

Solr's open access to search results over HTTP makes interaction very easy, but a web application still needs to parse the response (which may be XML or other formats such as JSON, Ruby, Python, or binary Java-to-Java) and provide navigation. There are native libraries in several languages that abstract the HTTP request and response parsing to make Solr integration extremely straightforward.

This section covers only a fraction of the ways to integrate with Solr programmatically. See Integrating Solr for more information and examples.

LucidWorks Enterprise on Java

Solr comes with a Java API, SolrJ, to natively interact with Solr. Here is an example program to search:

```
import org.apache.solr.client.solrj.SolrServer;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.client.solrj.impl.CommonsHttpSolrServer;
import org.apache.solr.common.SolrDocument;

import java.net.MalformedURLException;

public class SolrJExample {
     public static void main(String[] args) throws MalformedURLException, SolrServerException
{
      SolrServer solr = new CommonsHttpSolrServer("http://localhost:8888/solr/collection1");
QueryResponse response = solr.query(new SolrQuery("my query"));
      for (SolrDocument doc : response.getResults()) {
        System.out.println(doc.getFieldValue("id") + ": " + doc.getFieldValue("title"));
        }
      }
    }
```

For more information on SolrJ, see the SolrJ wiki page.

LucidWorks Enterprise on Ruby

Using the solr-ruby library (installed with gem install solr-ruby) for searching LucidWorks Enterprise is simple and elegant:

```
require 'solr'
solr = Solr::Connection.new("http://localhost:8888/solr/collection1")
results = solr.query("my query")
results.each {|hit| puts "#{hit['id']}: #{hit['title']}"}
```

For more information on the solr-ruby library, see Solr-Ruby.

Query Parsers

All of query parsers included with Solr are available for use, in addition to the enhanced parser included with LucidWorks Enterprise.

Name	id	availability	Description
Lucene or Solr	lucene	Solr	The Lucene Query Parser, with some Solr enhancements. More information can be found at the Solr Wiki page for Solr query syntax.
Function	func	Solr	Parses a FunctionQuery which calculates a function over field values. More information available at the Solr Wiki page for function queries.
DisMax	dismax	Solr	Search across multiple fields, allow +, -, and phrase queries while escaping most other Lucene syntax to avoid syntax errors. More information available at the Solr Wiki page for the DisMaxRequestHandler.
Extended DisMax	edismax	Solr	A version of the Extended DisMax parser developed by Lucid and donated to the Apache Software Foundation for inclusion in Solr.
Lucid	lucid	LucidWorks Enterprise	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
field	field	Solr	Single-purpose micro query parser. Generates a query on a single field.
prefix	prefix	Solr	Single-purpose micro query parser. Generates a prefix query on a single field.
raw	raw	Solr	Single-purpose micro query parser. Generates a raw unanalyzed term query.
Boosted	boost	Solr	Single-purpose micro query parser. Generates a BoostedQuery which boosts a Query by a FunctionQuery.
Nested	query	Solr	Single-purpose micro query parser. Delegates to another query parser.

Performance Tips

A number of configuration items can be manipulated for better performance when benchmarking LucidWorks Enterprise. 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.

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.
- 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`
- Optimize the index occasionally to minimize the number of segments in the index. When benchmarking, the number of segments in the index should be noted.
- Review the section on Wildcards at Start of Terms if leading wildcards have been enabled for important performance considerations.

Expanding Capacity

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.

Index Replication shows how to configure multiple shards for a master-slave environment.

Distributed Search and Indexing discusses how to distribute search and indexing processes across multiple servers or shards for peak performance.

Solr Cloud covers the new cloud-capable features of Solr and how to use them with LucidWorks Enterprise.

Index Replication

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.

When the Click Scoring Relevance Framework is enabled LucidWorks Enterprise ensures that also the click boost data is replicated together with index files.

LucidWorks Enterprise supports index replication, but it is not configured through the Administration User Interface. Instead, replication configuration requires editing XML configuration files in the Solr release included with LucidWorks Enterprise. This chapter 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.

Note
Remember that LucidWorks Enterprise uses Solr multi-core technology, with "collection1" as the default core, causing paths to be of the form `http://host:port/solr/collection1/*` instead of `http://host:port/solr/*` as shown in the Solr documentation.

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 ':'.

More Information

For more information about configuring index replication, see Chapter 10 of the LucidWorks for Solr Certified Distribution Guide.

Distributed Search and Indexing

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 Enterprise server containing a complete index that can be queried independently, but which only contains a fraction of the complete search collection.

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="distrib">
  <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.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
```

Indexing Documents

If distributed indexing has been configured as above, then any indexing initiated from the LucidWorks Enterprise 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/solr&
shards=localhost:8983/solr,localhost:7574/solr,localhost:8888/solr
```

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 Enterprise), 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/select?shards=localhost:8983/solr,localhost:7574/solr,localhost:8888/solr&q
super
```

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 Enterprise 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.

Solr Cloud

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

Solr Cloud Examples

Solr Cloud is currently an expert level API in development, and the current state is documented on the SolrCloud wiki page. The documentation for the specific SolrCloud version included with this version of LucidWorks Enterprise is here.

LucidWorks Enterprise command line start and stop is a bit different than the example server included in Apache Solr. Some changes are needed to follow along with the Solr examples.

- Instead of `java -jar start.jar`, use `./start.sh` or `./start.bat` and use the command line paramater -lwe_core_java_opts to pass additional command line options to the JVM.
- We recommend that you only install LWE using the installer application; copying the LucidWorksEnterprise directory to another directory to create another server may cause conflicts with ports.
- The Solr config directory in LucidWorks Enterprise is `./conf/solr/cores/collection1_0/conf` as opposed to `./solr/conf`
- Queries should include an additional query parameter `&distrib=true` to submit the search to multiple shards.

Here is how one would set up the SolrCloud Example A: Simple two shard cluster.

Because we need two servers for this example, we will make two installations of LucidWorks Enterprise, one in `example` and the other in `example2` . The installation in `example` should use port 8983 for LWE Core (selected in the installer). The installation in `example2` should use the default port (8888) for LWE Core. If enabling other components, be sure to run them on different ports in each installation as well.

The original Apache Solr instructions look like this:

```
cd example
java -Dbootstrap_confdir=./solr/conf -Dcollection.configName=myconf -DzkRun -jar start.jar
```

Equivalent LucidWorks Enterprise instructions for Linux-based systems:

```
cd example
bin/start.sh -lwe_core_java_opts "-Dbootstrap_confdir=./solr/cores/collection1_0/conf
-Dcollection.configName=myconf -DzkRun"

cd ../example2
bin/start.sh -lwe_core_java_opts "-DhostPort=8888 -Dcollection.configName=myconf
-DzkHost=localhost:9983"
```

For Windows-based systems:

```
cd example
bin\start.bat -lwe_core_java_opts "-Dbootstrap_confdir=./solr/cores/collection1_0/conf
-Dcollection.configName=myconf -DzkRun"

cd ../example2
bin\start.bat -lwe_core_java_opts "-DhostPort=8888 -Dcollection.configName=myconf
-DzkHost=localhost:9983"
```

Browse to http://localhost:8983/solr/collection1/admin/zookeeper.jsp to see the state of the cluster (the zookeeper distributed filesystem).

Integrating Monitoring Services

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 Enterprise provides additional capabilities to integrate application level statistics information into these monitoring tools.

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 Enterprise 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 Enterprise

By default JMX is enabled in LucidWorks Enterprise 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.

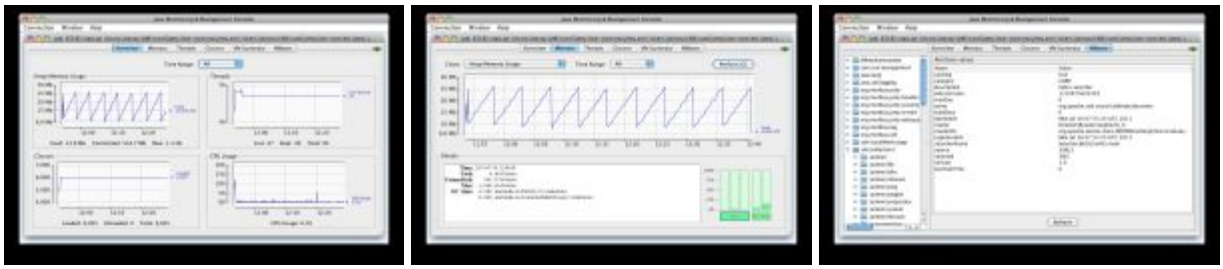
You might want to secure remote JMX access either by configuring a software or hardware firewall to allow connections to specified port only from your hosts/network or by configuring password authentication and/or SSL encryption. For more information about various security options please refer to the JMX documentation.

JMX Clients

There are number of various JMX clients out there you can use to connect to LucidWorks Enterprise 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.



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 Enterprise provides number of useful JMX MBeans, some in Solr and some in LucidWorks Enterprise:

Solr MBeans

Domain	Objects	Available attributes
solr/<collection_name>	type=updateHandler,id=org.apache.solr.update.DirectUpdateHandler2	cumulative_adds, cumulative_deletesI cumulative_deletesI cumulative_errors, commits, autocomm optimizes, rollbacks docsPending, etc

solr/<collection_name>	type=/update,id=org.apache.solr.handler.XmlUpdateRequestHandler	request, errors, avgTimePerRequest
solr/<collection_name>	type=/lucid,id=org.apache.solr.handler.StandardRequestHandler	requests, errors, tin avgTimePerRequest

solr/<collection_name>	type=searcher,id=org.apache.solr.search.SolrIndexSearcher	numDocs, warmupT

solr/<collection_name>	type=filterCache,id=org.apache.solr.search.LRUCache	cumulative_eviction cumulative_hitratio, cumulative_hits, cumulative_inserts, cumulative_lookups warmupTime, etc
solr/<collection_name>	type=queryResultCache,id=org.apache.solr.search.LRUCache	cumulative_eviction cumulative_hitratio, cumulative_hits, cumulative_inserts, cumulative_lookups warmupTime, etc
solr/<collection_name>	type=documentCache,id=org.apache.solr.search.LRUCache	cumulative_eviction cumulative_hitratio, cumulative_hits, cumulative_inserts, cumulative_lookups

LucidWorks Enterprise 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 Enterprise 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 Enterprise server with popular open source monitoring tools - [Zabbix|#Zabbix] and [Nagios|#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 Enterprise with Zabbix and the best approach depends on the Zabbix release version.

Post-2.0 releases

Post-2.0 releases (currently it's in beta release stage) comes with built-in support for monitoring Java applications (Zabbix Java proxy). For more information please see the JMX Monitoring section of the Zabbix manual.

Pre-2.0 releases

If you are handy with scripting and command line tools then you can also gather and send all the JMX information

using either:

- UserParameter: You can configure the Zabbix system agent to send custom monitored items using UserParameter configuration parameter. For retrieving JMX statistics you can use either cmdline-jmxclient or jmxterm command line clients.

```
UserParameter=jvm.maxthreads, java -jar cmdline-jmxclient.jar localhost:3000
java.lang:type=Threading PeakThreadCount
```

- zabbix_sender tool: 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 it 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 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
...
```

How to integrate with Zabbix 2.0 (1.9.x)

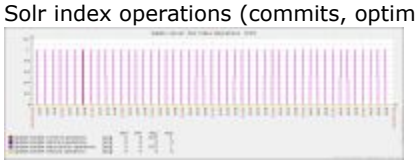
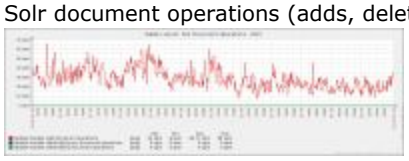
This section covers step by step guide how to integrate LucidWorks Enterprise product with the Zabbix 2.0 (1.9.x) release. This won't work with previous releases (1.8.x) because they lack built-in JMX support.

1. Download and install 2.0 (1.9.x) release according to official documentation. 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`.
2. After `make install` you should 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 as well. To do that you should add `<install_dir>/sbin/zabbix_java/startup.sh` and `<install_dir>/sbin/zabbix_java/shutdown.sh` calls to the corresponding options in `init.d`.
3. Configure JMX proxy in `/etc/zabbix/zabbix_server.conf` (see `JavaProxy`, `JavaProxyPort` and `StartJavaPollers` parameters). Verify that you're using the same port configured in `<install_dir>/sbin/zabbix_java/settings.sh` file. It is also recommended to enable JMX proxy verbose logging (edit `<install_dir>/sbin/zabbix_java/lib/logback.xml` file and change `file` element to point to your log file directory and set level attribute to debug level).
4. Import the sample Zabbix templates found in `$LWE_HOME/app/examples/zabbix` called `lwe_zabbix_templates.xml` (there are 3 in that file).
5. Install the Zabbix agent to the server where LucidWorks Enterprise is installed and configure it to connect to

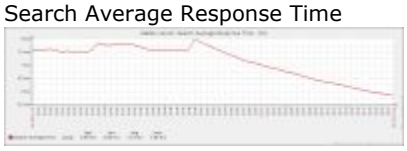
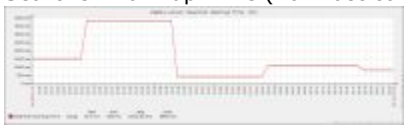
the Zabbix server.
6. Add Zabbix host and assign proper template for the OS (linux, freebsd, etc.).
7. Assign the imported templates (Template_JVM, Template_Solr, Template_LWE) to that host.
8. Enable JMX monitoring in LucidWorks Enterprise and allow the Zabbix server connect to JMX interface over the network.
9. Add JMX interface to host where LucidWorks Enterprise is installed.
10. Start any activity in the LucidWorks Enterprise server (crawling, indexing, serving) and check out graphs for monitored host (see screenshots below).

Example graphs

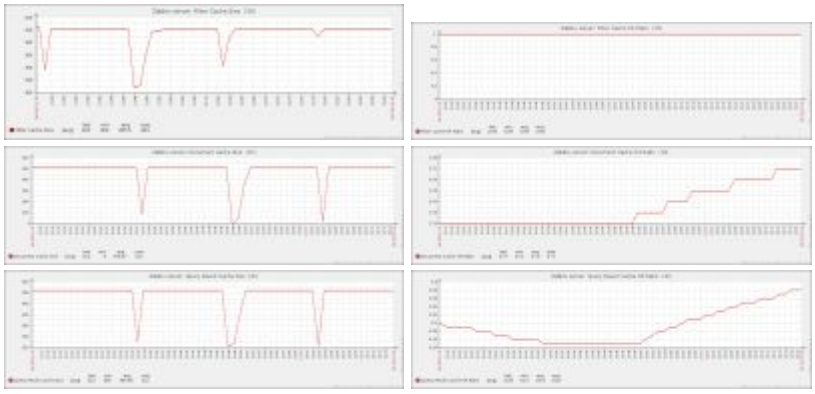
- Total number of documents in search index

- 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:

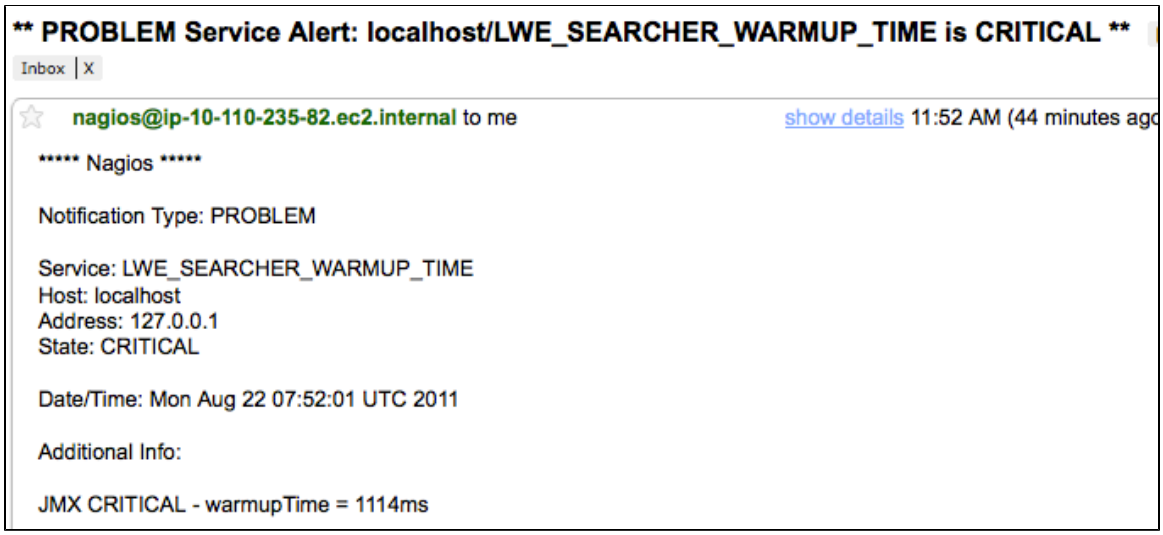
```
# LWE searcher warmup time is no more than 1) 1 second - warning state 2) 2 seconds - critical
state
define service {
        hostgroup_name                  all
        service_description             LWE_SEARCHER_WARMUP_TIME
        check_command                   check_jmx!3000!-O
"solr/collection1:type=searcher,id=org.apache.solr.search.SolrIndexSearcher" -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



Helpful tips

- OS file system cache: One of the frequent problems with LucidWorks Enterprise 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.

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 Enterprise 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 catenation 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.

h3. 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.

h3. 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 Enterprise 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 LucidWorks Enterprise 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 Enterprise 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 Enterprise, so that one term (either) can be inclusive while the other is exclusive.

Another LucidWorks Enterprise 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 Enterprise 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 Enterprise 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.

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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise query parser will catch the exception and simply ignore the date term.

Non-Text, Date, Numeric Field Queries

The LucidWorks Enterprise 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 Enterprise query parser has a special feature that selects all documents as efficiently as possible using a special Lucene API. The syntax is simply `*:*`.

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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise, 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"`

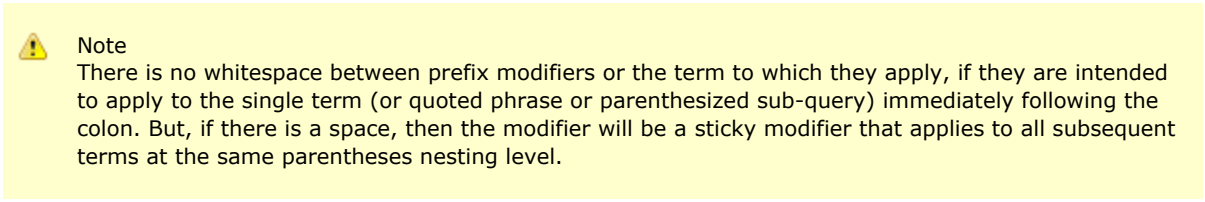
The precise quoted phrase will match the exact phrasing so will not catch the statement if it is reworded as "Employees attended the company meeting." But the first query will match that rewording, and rank it reasonably high due to the match on the trigram "The company meeting".

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`

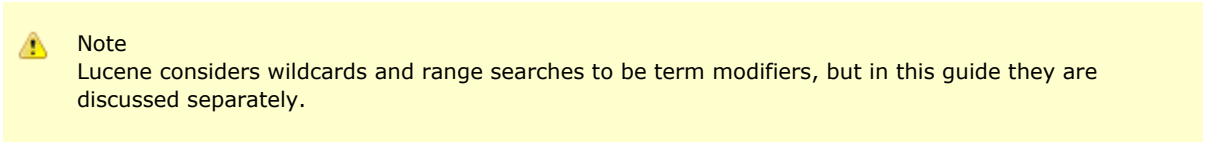


Note
There is no whitespace between prefix modifiers or the term to which they apply, if they are intended to apply to the single term (or quoted phrase or parenthesized sub-query) immediately following the colon. But, if there is a space, then the modifier will be a sticky modifier that applies to all 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 boot 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 Enterprise generates a disjunction maximum query if multiple fields are specified for the default query fields.

The LucidWorks Enterprise 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 Enterprise 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 Enterprise 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 Enterprise 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 Enterprise 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)
!
^
~
(
)
{
}
[
]
:
"
==
<
>
  (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

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 Enterprise should be restarted. On some Windows systems, it may be necessary to stop LucidWorks Enterprise 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>
```

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. Any arbitrary stemming filter can be specified, but Lucid KStem is a typical alternative.

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"/> -->
```

Be sure that you have chosen 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.

Other alternative stemming filters, such as Snowball and Porter, can be used by using a similar technique as described above.

LucidWorks REST API Reference

Overview

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, the LucidWorks Search Platform provides programmatic, remote access to most high-level aspects of configuration and operation through the REST API.

Most web developers are familiar with the idea of manipulating HTTP requests. For example, your programmatic access to search itself is by making GET requests to Solr itself, as documented in the Solr Wiki; you may even be familiar with the idea of indexing data by making a POST request directly to Solr. In the case of the REST API, the LucidWorks Search Platform uses GET and POST, as well as PUT and DELETE requests, to enable you to programmatically manage tasks such as creating and managing data sources and alerts, as well as monitoring the ongoing activities of your LucidWorks system.

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. REST requests work as follows:

GET: Used to get information on a single object, or to get a list of objects in a group.
POST: Used to create a new object.
PUT: Used to update an object or group of objects.
DELETE: As expected, used to delete 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) in question. The system then sends back the results as a JSON object. (Currently, LucidWorks only recognizes JSON requests; this may change in the future.)

About Server Addresses in Examples
The LucidWorks Search Platform REST API uses the Core component, installed at http://localhost:8888/ by default in LucidWorks Enterprise and many examples in this Guide use this as the server location. If you changed this location on install, be sure to change the destination of your REST requests. If using LucidWorks Cloud, the server call must include the Access Key, which can be found on the My Search Server page. Example URLs for API calls would then be changed from `http://localhost:8989/api/...` to `http://<server address>/<access key>/api/....`

Quick Start

Getting Started Indexing

Error Response Format

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.
  - 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.

    - Fields: How data from a single document is organized in LucidWorks indexes.
    - Field Types: Create new field types or modify existing types.
    - JDBC Drivers: Load required JDBC drivers for database indexing.
    - Settings: Many different query- and index-time settings.
    - Collection Templates: Get information about templates that can be used when creating new collections.
    - Roles: Configure search filters to control access to documents.
    - Filtering Results: Use Access Control Lists to filter results for Windows Shares.
    - Search Handler Components: List active search components for a particular search handler.
    - Collection Index: Delete the entire index or only data from a single data source.
- 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.

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.

Getting Started Indexing

How to Find Your Server Address
In these examples, URLs are shown with `http://<server address>/...`. If you are using LucidWorks Cloud, the server address is found on the My Search Servers page, and includes an Access Key that is required for all client interactions. If you are using LucidWorks Enterprise, the server address is where the Core component was installed.

Perhaps the most common usage of the REST API is 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, you 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\\LucidWorksEnterprise\\bin\\..\\solr\\cores\\collection1_0\\da
,
    "index_size": "1.3 MB",
    "index_directory": {
        "directory":
"C:\\Users\\Nick\\LucidImagination\\LucidWorksEnterprise\\solr\\cores\\collection1_0\\data\\index
,
        "lockID": "lucene-87258 8020481afad83452b87f7517d46",
        "readChunkSize": 104857600,
        "lockFactory": {
            "lockDir":
"C:\\Users\\Nick\\LucidImagination\\LucidWorksEnterprise\\solr\\cores\\collection1_0\\data\\index
,
            "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.

Advanced Operations Using the REST API

LucidWorks Enterprise 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 LWE, 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 '{"url" : "http://www.grantingersoll.com","crawl_depth" : "2", "type" : "web", "name" :
"Sample Site"}'
'http://localhost:8888/api/collections/collection1/datasources'
```

The response includes the id value:

```
{
   "id":7,
   "include_paths":null,
   "collect_links":false,
   "name":"Sample Site",
   "exclude_paths":null,
   "type":"web",
   "url":"http://www.grantingersoll.com",
"crawl_depth":2
}
```

The next step is to tell LWE to run the index by updating the schedule for the data source. That means updating the schedule for source number 7 using a `PUT` request:

```
curl -X PUT -H "Content-Type: application/json"
-d '{"active":true, "type":"index", "start_time": "now", "period":0}'
'http://localhost:8888/api/collections/collection1/datasources/7/schedule'
```

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/7/status'
```

This request tells us that the data source is still being indexed:

```
{
   "id":7,
   "description:":"Crawling Documents",
   "status":
       {"crawl_started":"2011-02-27T02:48:41+0000",
        "num_unchanged":0,
        "num_updated":0,
        "num_new":26,
        "num_removed":0},
    "running":true
}
```

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 -d '{
  "username": "smiller",
  "last_name": "miller",
  "email": "me@here.com",
  "first_name": "john",
  "password": "superman"
}' -H 'Content-type: application/json' 'http://localhost:8888/api/users'
```

Once you have defined a user, you can then use the REST API to perform authentication:

```
curl -d 'superman' -H 'Content-type: application/json' 'http:
//localhost:8888/api/users/smiller/authenticate'
```

This API is simple, taking just the password and returning just a `true` or `false` value, depending on whether the username in the URL and the password passed match:

```
true
```

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 -d '{"name":"Robot Alert", "query":"robot"}'
-H "Content-type:application/json"
'http://localhost:8989/alerts/users/smiller/alerts'
```

(Note that Enterprise Alerts do not run on the Solr port; they're handled by the LWE Core component, typically installed on port 8989.)

The request returns a JSON representation of the object, which includes the ID:

```
{
   "alert_at":null,
   "collection_url":"",
   "created_at":"2011-02-27T03:35:28Z",
   "id":2,
   "min_alert_interval":0,
   "name":"Robot Alert",
   "query":"robot",
   "results_since":"2011-02-27T03:35:28Z",
   "retrieved_at":"2011-02-27T03:35:28Z",
   "update_interval":null,
   "updated_at":"2011-02-27T03:35:28Z",
   "user_identifier":"smiller",
   "properties":{}
}
```

You can then use that ID to check the status of the alert:

```
curl 'http://localhost:8989/alerts/users/smiller/alerts/2'
```

In this case, we have not yet seen any new results;


```
{
   "alert_at":null,
   "collection_url":"",
   "created_at":"2011-02-27T03:35:28Z",
   "id":2,
   "min_alert_interval":0,
   "name":"Robot Alert",
   "query":"robot",
   "results_since":"2011-02-27T03:35:28Z",
   "retrieved_at":"2011-02-27T03:35:28Z",
   "update_interval":null,
   "updated_at":"2011-02-27T03:35:28Z",
   "user_identifier":"smiller",
   "properties":{}
}
```


The LucidWorks Enterprise documentation lists all available functions for the REST API.

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

`/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

```
{
 "solr" : {
      "solr.svn.rev" : "1088021",
      "build.os" : "Windows Vista",
      "svn.repo" : "http://svn.apache.org/repos/asf/lucene/dev/trunk",
"git.commit" : "ec6045ebc1118237917f973d9b779752a9719dab",
      "build.host" : "BIGBOY",
      "build.date" : "2011/04/04 12:57",
      "build.user" : "rmuir",
      "git.repo" : "git@github.com:lucidimagination/lucene-solr.git"
   },
 "lucidworks" : {
      "hudson.build.number" : "0",
      "build.os" : "Linux",
      "version" : "0.0Canop",
      "git.commit" : "39408ca4688e635d1ce53cb8e5a8a95a9171c74c",
      "environment" : "production",
      "build.host" : "bester",
      "build.date" : "2011/04/13 18:59",
      "build.user" : "hossman"
   }
}
```

Collections

Data in LucidWorks 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.

API Entry Points

`/api/collections`: Get a list of available collection names

`/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

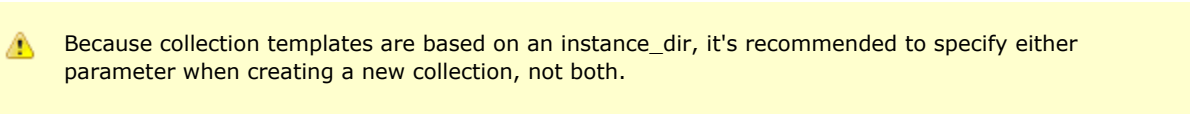
Query Parameters

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. 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`.



Output

Output Content

JSON representation of new collection

Key	Type	Description

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 stored with the other collections in the `$LWE_HOME/data/solr/cores` directory, but in a directory called "socialdata".

Input

```
curl -H 'Content-type: application/json' -d '{"name": "social", "instance_dir":"socialdata"}'
     'http://localhost:8888/api/collections'
```

Output

```
{
    "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 (LucidWorks Enterprise only) 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.

Collection Info

The Collections Info API can be used to retrieve some statistics about a particular collection. You can use this API to retrieve all information, or just a specific key, such as the `data_dir`.

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\\collection1_0
,
    "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
}
```

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 performs these activities periodically. However, activities can also be set for a single point in time. 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	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.
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.

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.
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.)

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	Indicates if the data source is currently being indexed
type	string	The activity type

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. It provides a data source's history as a list of history instances.

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 access method. This description is then used to create a crawl job to be executed by a specific crawler implementation (called "crawler controllers").

At present, the LucidWorks Search Platform 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 system • Web
DataImportHandler-based JDBC crawler	lucid.jdbc	• JDBC database
SolrXML crawler	lucid.solrxml	• Solr XML files
Google Connector Manager-based crawler	lucid.gcm	• Microsoft SharePoint (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 sharing) filesystem • Hadoop Distributed File System (HDFS) • Amazon S3 file system (also known as "S3 native") • HDFS over Amazon S3
External data	lucid.external	• Externally generated data

API Entry Points

`/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 fields to values. The exact set of fields depends on the kind of data source. Commonly used kinds of data sources use the following symbolic names: file (files on a local file system), web (HTTP/HTTPS web sites), jdbc (JDBC databases), solrxml (files in Solr XML format), and sharepoint (Microsoft SharePoint). All return:

Key	Type	Description
id	32-bit integer	The numeric ID for this data source.
type	string	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 • s3n for a native S3 filesystem • s3 for a Hadoop-over-S3 filesystem • external for an externally-managed data source
crawler	string	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, when the filesystem is local • lucid.fs for file, smb, hdfs, s3n, and s3 types, when the filesystem is remote • lucid.gcm for sharepoint type • lucid.jdbc for jdbc type • lucid.solrxml for solrxml type • lucid.external for external type
collection	string	The name of the document collection that documents will be indexed into.
name	string	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	The category of this data source: Web, FileSystem, Jdbc, SolrXml, Sharepoint, External, or Other. For informational purposes only.

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 key "mapping" contains a JSON map with the following keys and values:

Key	Type	Description
mappings	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.

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.
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
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).

unique_key	string	Defines the name of the field in the current schema that is a unique key. Filled in from the current schema. 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.
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.
literals	JSON string-string	An optional map that can specify static pairs of keys and values to be added to output documents.

The following fields are optional and are supported across all data source types listed here:

Key	Type	Description
commit_within	integer	Number of milliseconds that defines the maximum interval between commits while indexing documents.
commit_on_finish	boolean	When true (the default), then commit will be invoked at the end of crawl.

The following fields control the batch processing, and are also optional. Note: some crawler controllers don't support batch processing, or support only a subset of options. See also the Batch Crawling API and more information on Batch (Split) Crawling.

Key	Type	Description
parsing	boolean	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	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	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.

Below are specific fields for each data source type.

lucid.aperture / File system:

Key	Type	Description
path	string	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 Enterprise can access the path, and will return an error if it cannot.
follow_links	boolean	Use true to instruct the crawler to follow symbolic links in the file system.

bounds	string	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.
include_paths	list of strings	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.
exclude_paths	list of strings	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.
crawl_depth	32-bit integer	How many path levels to descend. Use '-1' to indicate unlimited depth, which is also the default if left empty.
max_bytes	long	Defines the maximum size of any crawled file. The default is -1, which is 10Mb per document.
url	string	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.
fail_unsupported_file_types	boolean	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.
warn_unknown_mime_types	boolean	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.

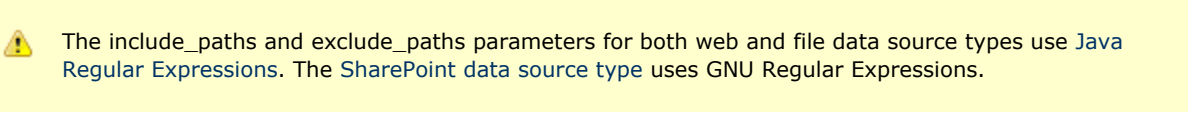
Example file data source

```
{
    "crawl_depth": 5,
    "follow_links": true,
    "name": "LucidWorks Documentation",
    "path": "D:\\lwe\\docs\\lucidworks",
    "type": "file",
    "bounds":"tree",
    "crawler": "lucid.aperture"
}
```

lucid.aperture / Web:

Key	Type	Description
url	string	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.

bounds	string	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.
include_paths	list of strings	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\.lucidimagination\.com/.*` if you want to crawl all pages under the URL `http://www.lucidimagination.com`.
exclude_paths	list of strings	Regular expression patterns that URLs may not match. If not empty then a file is excluded if any pattern matches its URL.
crawl_depth	32-bit integer	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.
max_bytes	long	Defines the maximum size of any crawled file. The default is -1, which is 10Mb per document.
max_docs	long	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.



Robots.txt
This data source obeys most of the robots.txt standard, with the exception of the Crawl-Delay directive.

HTTP Proxy
This data source supports communication via an HTTP proxy, either open or authenticated. The following data source properties determine the use of the proxy:

Key	Type	Description
proxy_host	string	host name of the proxy. If null or absent then direct access is assumed, and all other proxy-related parameters are ignored.
proxy_port	string	port number of the proxy
proxy_username	string	optional username credential for the proxy
proxy_password	string	optional password credential for the proxy

Authentication
At this time, only the Basic and Digest types of HTTP authentication methods are supported. NTLMv1 and NTLMv2 authentication are not supported in this release.

A single data source may have multiple sets of credentials for different sites, or for different realms within a single site. This authentication data is passed within the `auth` property of the data source as a list of JSON maps, each map with the following properties:

Key	Type	Description
host	string	host name (or host:port) where this authentication should be used; may be null to indicate any host
realm	string	HTTP Realm where this authentication should be used; may be null to indicate any realm
username	string	user name; must not be null or empty
password	string	password; must not be null

The HTTP connector first tries to access resources without any authentication. When it receives an HTTP code "401 Authentication Required" the authentication method (Basic, Digest or NTLMv1) is selected automatically, and the closest matching authentication tuple is selected from the ones configured for the data source.

Example web data source

```
{
    "id": 2,
    "collect_links": true,
    "crawl_depth": 2,
    "exclude_paths": [
        "http://www\\.lucidimagination\\.com/blog/tag/.*",
"http://www\\.lucidimagination\\.com/search\\?.*"
],
    "include_paths": [
        "http://www\\.lucidimagination\\.com/.*"
],
    "auth": [
        {
            "host": "www.lucidimagination.com:443",
            "realm": "Test realm",
            "username": "user1",
            "password": "test1"
        },
        {
            "host": "www.lucidimagination.com",
            "realm": null,
            "username": "user2",
            "password": "test2"
        }
    ],
    "name": "Lucid Imagination Website",
    "type": "web",
    "crawler": "lucid.aperture",
    "url": "http://www.lucidimagination.com/"
}
```

lucid.jdbc / JDBC:

JDBC databases must have a valid JDBC driver available to the crawler that is appropriate for your RDBMS. To load a driver, use the JDBC Drivers API.

Key	Type	Description
driver	string	The class name of the JDBC driver.
username	string	The username of a database account that should be used to access the data.

password	string	The password of the account that should be used to access the data.
url	string	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.
sql_select_statement	string	The select statement to use to generate data.
primary_key	string	The column name of the primary key.
delta_sql_query	string	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 "$" character, the query will fail.
nested_queries	list of strings	If you want to index one-to-many or many-to-many relations in addition to plain rows then you can specify an arbitrary number of additional SQL "nested" queries. For example, if you want to index the list of assigned tags for your documents you can specify a query in the following form `"SELECT tag FROM tag INNER JOIN document_tag ON document_tag.tag_id=tag.id WHERE document_tag.doc_id=$"`. 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.

Example jdbc data source

```
{
    "id": 2,
    "name": "Test database",
    "type": "jdbc",
    "crawler": "lucid.jdbc",
    "driver": "com.mysql.jdbc.Driver",
    "username": "root",
    "password": "pass",
    "url": "jdbc:mysql://localhost/test",
"sql_select_statement": "select * from document",
    "primary_key": "id",
    "delta_sql_query": "select id from document where last_modified > $",
    "nested_queries": ["select category from document_category where doc_id=$",
                       "select tag from document_tag where doc_id=$"]
}
```

lucid.solrxml / Solr XML file

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.

Key	Type	Description

file	string	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 Enterprise server and will return an error if it is not.
include_datasource_metadata	boolean	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.
generate_unique_key	boolean	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	An array of URL patterns to include. Used when the data source has been configured to traverse a directory of possible Solr XML files.
exclude_paths	list of strings	An array of URL patterns to exclude. Used when the data source has been configured to traverse a directory of possible Solr XML files.
url	string	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.

Example solr xml data source

```
{
    "id": 2,
    "name": "Solr example XML documents",
    "type": "solrxml",
    "crawler": "lucid.solrxml",
    "file": "D:\\lucene_solr\\solr\\example\\exampledocs",
    "include_paths" : [ "*\.xml" ],
    "include_datasource_metadata": true
}
```

lucid.gcm / SharePoint:
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 Platform does not support crawling a SharePoint repository running on Microsoft Portal SharePoint Server 2003 or Microsoft Windows SharePoint Services 2.0.

The SharePoint crawler will index all content in the repository, including public and personal sites, files, discussion boards, calendars, contacts, and images. In order to index SharePoint content, Google Services for SharePoint must be installed to be able to fully use the embedded APIs that crawl a SharePoint server. The files are included with LucidWorks Enterprise, but should be moved to the SharePoint server as described below.

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, path of this directory would be C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI\ for SharePoint 2007 and ...\14\ISAPI for SharePoint 2010

3. Copy the following files found from `$LWE_HOME` into the ISAPI folder as specified in previous step:

```
ext/sharepoint_service/Bulk Auth/SharePoint 2007/GSBulkAuthorization.asmx
ext/sharepoint_service/Bulk Auth/SharePoint 2007/GSBulkAuthorizationdisco.aspx
ext/sharepoint_service/Bulk Auth/SharePoint 2007/GSBulkAuthorizationwsdl.aspx

ext/sharepoint_service/Site Discovery/SharePoint 2007/GSSiteDiscovery.asmx
ext/sharepoint_service/Site Discovery/SharePoint 2007/GSSiteDiscoverydisco.aspx
ext/sharepoint_service/Site Discovery/SharePoint 2007/GSSiteDiscoverywsdl.aspx

ext/sharepoint_service/Acl/GssAcl.asmx
ext/sharepoint_service/Acl/GssAcl.aspx
ext/sharepoint_service/Acl/GssAcl.aspx
```

Key	Type	Description
sharepoint_url	string	The fully qualified URL for the SharePoint site.
username	string	Username with authorization to crawl the SharePoint repository.
password	string	Password for the username above.
domain	string	The domain where the user is authenticated .
kdcserver	string	Kerberos KDC Hostname.
my_site_base_url	string	Used for MOSS 2007 only. The MySite base URL is used to determine the complete MySite URL, so http://server.domain/personal/administrator/default.aspx would be entered as http://server.domain. The credentials provided will allow the lucid.gcm crawler to complete the MySite URL and crawl the content.
included_urls	string	The directories on the server that should be crawled for indexing. If left blank, all paths will be followed, even if they lead away from the original URL entered. To limit crawling to a specific site, repeat the URL in this site with a regular expression to indicate all pages from the site. The SharePoint data source uses GNU regular expressions, which may be different from the Java regular expressions used for Web and file system data sources. More information on the syntax can be found in the GNU regular expression documentation.
excluded_urls	string	Directories on the server that should not be crawled and that should be excluded from the index. The same regular expression syntax can be used to specify excluded_urls as is used for included_urls.
use_sp_search_visibility	boolean	When true (default) then SharePoint search visibility options will be respected. Google Services for SharePoint must be installed to use this feature.
aliases	map of string to string	Allows mapping of source URL patterns to aliases that are used to rewrite URLs before indexing.
authorization	string	Should always be set to 'content', which is the default.

Example SharePoint repository data source

```
{
    "id": 2,
    "name": "Sharepoint crawl",
    "type": "sharepoint",
    "crawler": "lucid.gcm",
    "sharepoint_url": "http://my.sharepoint.host.com/",
"username": "user",
    "password": "secret",
    "domain": "myDomain",
    "included_urls": ".*",
    "use_sp_search_visibility": true,
    "authorization": "content"
}
```

lucid.fs / Remote or Pseudo Filesystems

The following properties are common to all remote and pseudo file systems (HDFS, S3, S3n, SMB):

Key	Type	Description
url	string	Root URL formats vary by file system type. For details on the root URL format, see the file-system-specific information below.
type	string	One of supported data source types, must be consistent with the root URL's protocol. The following values are supported: file, smb, hdfs, s3n, s3
max_bytes	long	Optional, default is -1, which is 10Mb per document.
bounds	string	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.
includes	list of strings	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/*`.
excludes	list of strings	Regular expression patterns that URLs must not match. If not empty then a file is excluded if any pattern matches its URL.
crawl_depth	32-bit integer	How many path levels to descend. Use '-1' to indicate unlimited depth which is also the default if left blank.

lucid.fs / File
There are no protocol-specific properties for File (`file://`) data sources.

lucid.fs / HDFS

Key	Type	Description
url	string	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: `hdfs://namenode:9000/path/to/crawl`.
username	string	

password	string	

lucid.fs / S3n

Key	Type	Description
url	string	For S3n (Amazon S3 native), the root URL is a fully-qualified URL that starts with the `s3n` protocol, the name of the bucket, and the path inside the bucket. 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		Your Amazon S3 AccessKeyId
password		Your Amazon S3 SecretAccessKey

lucid.fs / S3

Key	Type	Description
url	string	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. 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		Your Amazon S3 AccessKeyId
password		Your Amazon S3 SecretAccessKey

lucid.fs / SMB or CIFs (Windows Share)
Information about how LucidWorks interprets Access Control Lists for a Windows Share, see Crawling Windows Shares with Access Control Lists.

Key	Type	Description
url	string	For SMB (Windows Shares) filesystems, the root URL includes the protocol (`smb`), the host address, and the path to crawl: `smb://host/path/to/crawl`. 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	A username with READ and ACL READ permissions for accessing the Windows Share. It's recommended to create a special user for this purpose.
password	string	Password for a user with READ and ACL READ permissions for accessing the Windows Share.
windows_domain	string	The optional Windows Domain of the shared file system.

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 Enterprise is needed to make the changes visible.

Example remote filesystem (SMB) source

```
{
   "id":10,
   "name":"<datasource name>",
   "type":"smb",
   "crawler":"lucid.fs",
   "url":"smb://<host>/<path>/",
"username":"username",
   "password":"password",
   "windows_domain":"<your windows domain>"
    }
'http://localhost:8888/api/collections/<collection>/datasources'
```

lucid.external/Externally controlled data sources
External data sources are those which do not use any of LucidWorks Enterprise crawler features. They instead feed documents directly to Solr via another process, such as SolrJ. The purpose of adding it as a LucidWorks Enterprise data source is to link LucidWorks Enterprise features, such as the Admin UI or display of data sources as facets, to the documents. Once the data source is created, several fields should be added to each document to complete the link; see more information on this at Suggestions for External Data Source Documents.

External data sources cannot be scheduled or otherwise controlled with LucidWorks Enterprise. The process for feeding documents to the index is entirely controlled outside of the application. If regular updates to documents are required, a server-side cron job or other mechanism may need to be configured manually.

Key	Type	Description
name	string	The name of the data source.
source	string	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	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 source

```
{
   "id":45,
   "name":"Test External #1",
   "type":"external",
   "crawler":"lucid.external",
   "source_type":"Raw SolrXML",
   "source":"Solr update"
}
```

Response Codes

200: Success OK

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/datasources'
```

Output


```
[
    {
        "max_bytes": 10485760,
        "include_paths": [],
        "collect_links": true,
        "exclude_paths": [],
        "mapping": {
            "multi_val": {
                "fileSize": true,
                "author": true,
                "body": true,
                "title": true,
                "keywords": true,
                "description": false,
                "subject": true,
                "fileName": true,
                "dateCreated" : false,
                "attr": true,
                "creator": true
            },
            "default_field": null,
            "mappings": {
                "content-type ": "mimeType",
                "slide-count": "pageCount",
                "body": "body",
                "slides": "pageCount",
                "subject": "subject",
                "plaintextmessagecontent": "body",
                "lastmodifiedby": "author",
                "content-encoding": "characterSet",
                "date": null,
                "type": null,
                "creator": "creator",
                "author": "author",
                "mimetype": "mimeType",
                "title": "title",
                "plaintextcontent": "body",
                "created": "dateCreated",
                "contributor": "author",
                "description": "description",
                "contentcreated": "dateCreated",
                "pagecount": "pageCount",
                "name": "title",
                "filelastmodified": "lastModified",
                "fullname": "author",
                "fulltext": "body",
                "last-modified": "lastModified" ,
                "messagesubject": "title",
                "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.grantingersoll.com/",
"crawler": "lucid.aperture",
        "id": 1,
        "bounds": "tree",
        "category": "Web",
        "name": "Sample Site",
        "crawl_depth": 2
    },
    {
        "max_bytes": 21474836 47,
        "include_paths": [],
        "exclude_paths": [],
        "mapping": {
            "multi_val": {
                "fileSize": true,
                "body": true,
                "author": true,
                "title": true,
                "keywords": true,
                "subject": true,
                "descripti on": 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",
                "plaintextmessagecontent": " body",
                "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",
                "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"
        },
        "follow_links": true,
        "collection": "collection1",
        "type": "file",
        "crawler": "lucid.aperture ",
        "id": 2,
        "bounds": "tree",
        "category": "FileSystem",
        "name": "Small Test Collection",
        "path": "C:\\ Users\\Nick\\Documents\\Business",
```

```
        "crawl_depth": 2147483647
    }
]
```

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 fields. The ID field, if present, will be ignored. See fields in section on getting a list of data sources .

Output

Output Content

JSON representation of new data source. Fields 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 Lucid Imagination 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 '
{
    "crawl_depth": 2,
    "exclude_paths": [
        "http://www\\.lucidimagination\\.com/blog/tag/.*",
"http://www\\.lucidimagination\\.com/search\\?.*"
],
    "include_paths": [
        "http://www\\.lucidimagination\\.com/.*"
],
    "name": "Lucid Imagination Website",
    "type": "web",
    "crawler": "lucid.aperture",
    "url": "http://www.lucidimagination.com/"
}' 'http://localhost:8888/api/collections/collection1/datasources'
```

Output


```
{
    "id": 6,
    "max_bytes": 10485760,
    "include_paths": [
        "http://www\\.lucidimagination\\.com/.*"
] ,
    "collect_links": true,
    "exclude_paths": [
        "http://www\\.lucidimagination\\.com/blog/tag/.*",
"http://www\\.lucidimagination\\.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": "title",
            "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.lucidimagination.com/",
"crawler": "lucid.aperture",
    "bounds": "tree",
    "category": "Web" ,
```

```
    "name": "Lucid Imagination Website",
    "crawl_depth": 2
}
```

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

Fields 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\\.lucidimagination\\.com/.*"
] ,
    "collect_links": true,
    "exclude_paths": [
        "http://www\\.lucidimagination\\.com/blog/tag/.*",
"http://www\\.lucidimagination\\.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.lucidimagination.com/",
"crawler": "lucid.aperture",
    "bounds": "tree",
    "category": "Web" ,
```

```
    "name": "Lucid Imagination 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 fields or just those that need updating. Data source type, crawler type, and ID cannot be updated. Other fields are listed in the section on getting a list of data sources.

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

`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.

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 `index`.
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

```
{
  "period": 0,
  "type": "index",
  "start_time": "2010-03-29T12:10:32-0700",
  "active": true
}
```

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	Description
start_time	date string	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 'T' is entered without quotes. The */'` is for the time zone relative to UTC, and also entered without quotes.
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 `index` (required).
active	boolean	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": 3,
  "type": "index",
  "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

`/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
id	The unique id of the datasource.
num_total	The total number of documents found during the last crawl.

num_failed	The number of documents that could not be parsed.
num_deleted	The number of documents that were removed from the index.
num_new	The number of documents considered "new".
num_updated	The number of documents that were updated.
num_unchanged	The number of documents that were not changed.
crawl_started	The date and time the crawl started.
crawl_stopped	The date and time the crawl stopped.
crawl_state	The current state of the job. Entries are FINISHED, STOPPED, or RUNNING.
batch_job	If false, the content crawled will be added to the index.
job_id	The ID of the job itself.

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/8/job'
```

Output

```
{"id":8,
"crawl_started":"2011-08-15T13:06:34+0000",
"num_total":27,
"num_unchanged":0,
"crawl_state":"RUNNING",
"crawl_stopped":null,
"job_id":"8",
"batch_job":false,
"num_updated":0,
"num_new":23,
"num_failed":4,
"num_deleted":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
id	The unique id of the datasource.
num_total	The total number of documents found during the last crawl.
num_failed	The number of documents that could not be parsed.
num_deleted	The number of documents that were removed from the index.
num_new	The number of documents considered "new".
num_updated	The number of documents that were updated.
num_unchanged	The number of documents that were not changed.
crawl_started	The date and time the crawl started.
crawl_stopped	The date and time the crawl stopped.
crawl_state	The current state of the job. Entries are FINISHED, STOPPED, or RUNNING.
batch_job	If false, the content crawled will be added to the index.
job_id	The ID of the job itself.

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

```
[
{"id":3,
"crawl_started":null,
"num_total":0,"num_unchanged":0,
"crawl_state":"IDLE",
"crawl_stopped":null,"job_id":"3",
"batch_job":false,"num_updated":0,
"num_new":0,
"num_failed":0,
"num_deleted":0}

{"id":4,
"crawl_started":"2011-08-15T13:06:34+0000",
"num_total":27,
"num_unchanged":0,
"crawl_state":"RUNNING",
"crawl_stopped":null,
"job_id":"4",
"batch_job":false,
"num_updated":0,
"num_new":23,
"num_failed":4,
"num_deleted":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

`/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
id	The unique id of the datasource.
num_total	The total number of documents found during the last crawl.
num_failed	The number of documents that could not be parsed.
num_deleted	The number of documents that were removed from the index.
num_new	The number of documents considered "new".
num_updated	The number of documents that were updated.
num_unchanged	The number of documents that were not changed.
crawl_started	The date and time the crawl started.
crawl_stopped	The date and time the crawl stopped.
crawl_state	The current state of the job. Entries are FINISHED, STOPPED, or RUNNING.
batch_job	If false, the content crawled will be added to the index.
job_id	The ID of the job itself.

Examples

Input

```
curl 'http://localhost:8888/api/collections/collection1/datasources/2/status'
```

Output
While the data source is being processed:

```
{
    "num_total":24,
   "id":3,
   "crawl_stopped":null,
   "num_failed":0,
   "num_updated":0,
   "num_unchanged":0,
   "crawl_started":"2011-09-12T20:51:55+0000",
   "num_deleted":0,
   "batch_job":false,
   "num_new":24,
   "crawl_state":"RUNNING",
   "job_id":"3"
}
```


After processing is finished, and the data source is idle:


```
{
   "num_total":44,
   "id":3,
   "crawl_stopped":"2011-09-12T20:53:05+0000",
   "num_failed":1,
   "num_updated":0,
   "num_unchanged":0,
   "crawl_started":"2011-09-12T20:51:55+0000",
   "num_deleted":0,
   "batch_job":false,
   "num_new":43,
   "crawl_state":"FINISHED",
   "job_id":"3"
}
```


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

`/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

`/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 Crawling API

The Batch Crawling API allows you to work with batches and batch crawling jobs. For information about batch crawling, including how to configure a data source for batch crawling, see Batch (Split) Crawling.

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":"00000000-0000-0000-0000-05c21a42057b",
 "parsed":true,
 "description":"HTTP-AUTH",
 "finish_time":0,
 "start_time":1313524874268,
 "parsed_docs":115,
 "ds_id":"58",
 "collection":"collection1",
 "crawler":"lucid.aperture"
},
{
 "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":"00000000-0000-0000-0000-05a6e ac33e56",
 "parsed":true,
 "description":"HTTP-AUTH",
 "finish_time":0,
 "start_time":1313524757507,
 "parsed_docs":61,
 "ds_id":"58",
 "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/00000000-0000-0000-0000-0
```

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/00000000-0000-0000-0000-0
```

Output

None.

Fields

The Fields API allows for accessing, modifying, or adding field definitions to a Collection 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

None

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 <string>	A list of field names that this field will be copied to.
default_boost	float	How much to boost the field when it is not explicitly included in a user's query. 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. This allows a default value to 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.
editable	boolean	Set to true if an external client is able to edit the field.
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`. 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. If this setting is changed, we recommend that you recreate the spelling check index as described in Auto-Complete of User Queries.

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 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.
num_facets	integer	Shows the number of facets that will be displayed if _facet_is true.
omit_tf	boolean	The omit_tf attribute sets the 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 the 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 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 omit field norms, moderate to boost moderately with the Sweet Spot Similarity implementation, 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 unanalyzed text will be stored. A field 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.
synonym_expansion	boolean	Set to true if the lucid query parser will 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
 }
]
```


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.

Key	Type	Required	Default	Description
name	string	Yes	No default	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 <string>	No	null	A list of field names that this field will be copied to.
default_boost	float	No	1.0	How much to boost the field when it is not explicitly included in a user's query. 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	No	null	A default text value for the field. This allows a default value to be entered if the field is empty in the document.
editable	boolean	No	No default	Set to true for an external client to be able to edit the field.
facet	boolean	No	false	Set to true if the lucid request handler will facet on this field. 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	Yes	No default	A valid field type defined in `schema.xml`. 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	No	false	Set to true if the lucid request handler should 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	No	false	Set to true if the lucid request handler should 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	No	false	Set to true if this field should 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	No	false	Set to true if this field should 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. If this setting is changed, we recommend that you recreate the spelling check index as described in Auto-Complete of User Queries.

indexed	boolean	No	true	Set to true if this field should 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	No	false	Set to true if this field should 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.
num_facets	integer	No	5	Set to the number of facets that should be displayed if the facet attribute is true.
omit_tf	boolean	No	false	The omit_tf attribute sets the 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.
omit_positions	boolean	No	false	The omit_positions attribute sets the 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	No	false	Set to true if the lucid query parser should intelligently remove stop words at query time. This will require LucidWorks 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	No	false	Set to true if this field should 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	No	high	Valid values are: none to omit field norms, moderate to boost moderately with the Sweet Spot Similarity implementation, 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	No	true	Set to true if the original unanalyzed text should be stored. A field 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.
synonym_expansion	boolean	No	false	Set to true if the lucid query parser should expand synonyms at query time.
term_vectors	boolean	No	false	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	No	false	Set to true if the contents of this field should be used when doing document de-duplication.
use_in_find_similar	boolean	No	false	Set to true if this field should 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.

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 settings "Search by Default" and "Enable Query 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 will query for synonym terms only in those fields that are both 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

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/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
}
```


Update a Field

`PUT /api/collections/*collection*/fields/*name*`

Input

Path Parameters

Key	Description
collection	The collection name.
name	The 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 "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. (Check field properties to confirm changes.)

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.

Query Parameters

none

Input content

None

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. This field will still be included in the list of fields returned.

FieldTypes

The FieldType API allows creating, updating and deleting field types that are used by LucidWorks 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 Enterprise 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.

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. 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 an 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 Enterprise 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.

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'
```

JDBC Drivers

If you want to index database content then you need to provide an appropriate JDBC driver for your RDBMS. The JDBC Drivers API allows you to upload drivers to LucidWorks. Drivers used with LucidWorks must be compliant with the 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=@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"
]
```

Settings

The Settings API allows for accessing and modifying settings for a given collection. Note that editing of some properties is disallowed by default, and cannot be done through this API.

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 LucidWorks Enterprise or LucidWorks Cloud default search interface. Note that this also requires setting the auto-complete activity to run at regular intervals. For more information, see Auto-Complete of User Queries.
boosts	Solr function query	Defines the boost to apply to each query. The default boost for 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 Enterprise only).
click_boost_data	string	The path to Click boost data (LucidWorks Enterprise only).
click_boost_field	string	The field name prefix used by Click fields (LucidWorks Enterprise only).
click_index_location	string	The path to Click boost index (LucidWorks Enterprise only).
de_duplication	string	The valid values are: off, do not de-duplicate; overwrite duplicate documents; tag duplicated with a unique signature.
default_sort	string	Default sort method - valid values are: relevance, date, random.
display_facets	boolean	Is true if the LucidWorks Enterprise or LucidWorks Cloud default search interface should display facets.
display_fields	string	Defines the fields to use for display of results to users. Primarily used to add pseudo-fields to documents, but could be used with "real" fields also. This parameter only applies when using the `lucid` handler (query parser).
query_parser	string	Which query parser the lucid search request handler will use - valid values are: lucid, dismax, extended dismax, lucene.
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 only be used if the 'lucid' query parser is selected as the default or used in the query request.

search_server_list	list:string	A list of Solr core URLs that the lucid request handler will use for distributed search - pass an empty list to disable distributed 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 Enterprise or LucidWorks Cloud default search interface should suggest spelling corrections.
stopword_list	list:string	A list of stopwords that will be used if 'query_time_stopwords' is enabled.
unsupervised_feedback	boolean	Is true if unsupervised feedback is enabled
unsupervised_feedback_emphasis	string	Defines if unsupervised feedback should emphasize "relevancy" which does an "AND" of the original query which neither includes nor excludes additional documents, or "recall" which does an "OR" of the original query which permits the feedback terms to expand the set of documents matched - default is "relevancy".
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.
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 chain will use for distributed updates and 'self' should either be null if this server will not receive updates, or it should be a string value containing this server address if this server will receive updates - pass an empty list of servers to disable distributed update.
elevations	complex	Advanced: A format that is roughly a JSON equivalent to http://wiki.apache.org/solr/QueryElevationComponent#elevate.xml Displays the elevate.xml file in the Collection's Solr `conf` directory (by default, `$LWE_HOME/solr/cores/collection1_0/conf`). To enable/disable elevation, configure the QueryElevationComponent and set it to use this elevate.xml file (see Input, below). Solr does not currently support the QueryElevationComponent with distributed search.

Response Codes

200: OK

Examples

Get the existing settings for the collection:

Input

```
curl 'http://localhost:8888/api/collections/collection1/settings'
```

Output

```
{
    "unsupervised_feedback_emphasis": "relevancy",
    "spellcheck": true,
    "search_server_list": [],
    "display_facets": true,
    "update_server_list": null,
    "display_fields" : ["id","url","author","data_source_type","dateCreated","description",
"keywords","lastModified",
       "mimeType","pageCount","title"],
    "unsupervised_feedback": false,
    "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"],
    "unknown_type_handling": "text_en",
    "boosts": ["recip(rord(lastModified),1,1000,1000)"],
    "de_duplication": "off",
    "query_time_stopwords": true,
    "click_boost_field": "click",
    "click_boost_data": "click-data",
    "query_parser": "lucid",
    "default_sort": "relevance",
    "auto_complete": true,
    "boost_recent": true,
    "click_enabled": false,
    "synonym_list": ["lawyer, attorney", "one, 1", "two, 2", "three, 3", "ten, 10","hundred,
100","thousand, 1000","tv, television"],
    "show_similar": true,
    "query_time_synonyms": true,
    "elevations": {},
}
```

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 LucidWorks Enterprise or LucidWorks Cloud default search interface.
boost		
boost_recent	boolean	Is true if the lucid request handler should boost recent documents.
click_enabled	boolean	Is true if Click is enabled (LucidWorks Enterprise only).
click_boost_data	string	The path to Click boost data (LucidWorks Enterprise only).
click_boost_field	string	The field name prefix used by Click fields (LucidWorks Enterprise only).
click_index_location	string	The path to Click boost index (LucidWorks Enterprise only).
de_duplication	string	The valid values are: off, do not de-duplicate; overwrite duplicate documents; tag duplicated with a unique signature.
default_sort	string	Default sort method - valid values are: relevance, date, random.
display_facets	boolean	Is true if the LucidWorks Enterprise or LucidWorks Cloud default search interface should display facets.

display_fields	string	Defines the fields to use for display of results to users. Primarily used to add pseudo-fields to documents, but could be used with "real" fields also. This parameter only applies when using the `lucid` handler (query parser).
query_parser	string	Which query parser the lucid search request handler will use - valid values are: lucid, dismax, extended dismax, lucene.
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 only be used if the 'lucid' query parser is selected as the default or used in the query request.
search_server_list	list:string	A list of Solr core URLs that the lucid request handler will use for distributed search - pass an empty list to disable distributed 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 Enterprise or LucidWorks Cloud default search interface should suggest spelling corrections.
stopword_list	list:string	A list of stopwords that will be used if 'query_time_stopwords' is enabled.
unsupervised_feedback	boolean	Is true if unsupervised feedback is enabled
unsupervised_feedback_emphasis	string	Defines if unsupervised feedback should emphasize "relevancy" which does an "AND" of the original query which neither includes nor excludes additional documents, or "recall" which does an "OR" of the original query which permits the feedback terms to expand the set of documents matched - default is "relevancy".
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.
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 chain will use for distributed updates and 'self' should either be null if this server will not receive updates, or it should be a string value containing this server address if this server will receive updates - pass an empty list of servers to disable distributed update.

elevations	complex	Advanced: A format that is roughly a JSON equivalent to http://wiki.apache.org/solr/QueryElevationComponent#elevate.xml Rewrites the elevate.xml file in the Collection's Solr `conf` directory (by default, `$LWE_HOME/solr/cores/collection1_0/conf`) and triggers a core reload. To enable/disable elevation, configure the QueryElevationComponent and set it to use this elevate.xml file. Solr does not currently support the QueryElevationComponent with distributed search. Example: `{ "elevations": { "dinner jacket": [ {"doc": "sku_8040489"} ], "shoes": [ {"doc": "sku_460030"}, {"doc": "sku_415090"} ] } }`

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.

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.

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"]
```

Roles

A role is a way to specify a group of users that may have specific privileges. In the case of LucidWorks, roles can also be 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 is also available in the Admin UI, where it is called "[Search Filters]".

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 object

`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.

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 Users group membership within Active Directory.

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 component configuration(s)

`GET /api/collections/`*`collection`*`/filtering`

Output Contents

A list of JSON hashes of instances configuration:

```
{
  "instance-name": {
    <instance configuration>
  }
}
```

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 Filtering Component Instance

`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

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 new Filtering component configuration

`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 the Configuration of 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 the Configuration of a Filtering Component Instance

`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.

- 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.

Collection Index Delete

The Collections Index API can currently be used to delete a collection's main index.

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 of 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 iacceptherisk

Output

Output Content

None

Response Codes

204: No Content

Examples

Input

```
curl -X DELETE http://localhost:8888/api/collections/collection1/index?key=iacceptherisk
```

Output

None.

Delete Indexed Data for a Data Source

`DELETE /api/collections/`*`collection`*`/datasources/`*`datasource`*`/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 be configured to include 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
datasource	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.

Alerts API

Enterprise Alerts are a way for users to create a search and save it so that LucidWorks Enterprise 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.

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 Enterprise, 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/user/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

204: No Content

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

Use the Users API to manage the built-in list of users in LucidWorks. You do not need to use this API if your installation is configured to pull user information from an LDAP server.

Unlike most of the other APIs in LucidWorks Enterprise, 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' role can access all parts of the LucidWorks Enterprise UI; users with 'search' role 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' role can access all parts of the LucidWorks Enterprise UI; users with 'search' role 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' role can access all parts of the LucidWorks Enterprise UI; users with 'search' role 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 role 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

The SSL Configuration API allows you to work with some of the LucidWorks Enterprise SSL 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. There are two features that can be configured through this API: configuring secure/insecure client access to LucidWorks Enterprise core API and configuring the Solr client SSL properties.

The configuration `keyauth_require_secure` controls secure/insecure access to the core API and Solr. Once this parameter is set to true only secure connections (SSL) are allowed. When `auth_require_secure` is set to false only insecure access is allowed. It is possible to have two connectors configured on the Servlet container level at the same time (one SSL and one non SSL) and you can control the access with this api.

In addition to toggling secure/insecure access it is also possible to configure LWE 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 LWE 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

`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_require_secure	boolean	no	null	Enforces SSL to be used when communicating to REST or Solr APIs. If false only HTTP is allowed.
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"],
  "auth_require_secure":false
}
```

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_require_secure	boolean	no	null	Enforces SSL to be used when communicating to REST or Solr APIs. If false only HTTP is allowed.
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

Accept only secure communications:

Once you set `auth_require_secure` to `true`, LWE only accepts secure (HTTPS) communications to the REST API, so make sure you have a properly configured Jetty SSL connector before setting this to true.

Input

```
curl -X PUT -H 'Content-type: application/json' -d
'{"auth_require_secure": true}'
 http://127.0.0.1:8888/api/config/ssl
```

Output

None.

Configure LWE 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",
  "auth_require_secure":true
}' http://localhost:8888/api/config/ssl
```

Output

None.

Example Clients

Example client code demonstrating how to communicate with LucidWorks Enterprise from a variety of programming languages can be found in the `$LWE_HOME/app/examples/` directory of your LucidWorks Enterprise Installation.

Example .Net Clients

The `$LWE_HOME/app/examples/csharp` directory contains utilities demonstrating some of the LucidWorks Enterprise REST API features from C# code. These utilities can be used to assist people in managing their LucidWorks Enterprise installation, or as an example of how to write C# code as part of customer applications that will interact with LucidWorks Enterprise and Solr.

Dependencies

All of these tools require that the "Json.NET" DLL be installed.

All of these tools assume that the main URL for LucidWorks Enterprise is "http://localhost:8888". If LucidWorks Enterprise 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 Enterprise REST API features from Perl code. These utilities can be used to assist people in managing their LucidWorks Enterprise installation, or as an example of how to write Perl code as part of customer applications that will interact with LucidWorks Enterprise 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 LWE is "http://localhost:8888" and that the URL for the UI is "http://localhost:8989"

If LWE 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 LWE
2. Create a data source using files on the same server as LucidWorks Enterprise:

```
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 Enterprise
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 Enterprise
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 Enterprise
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 Enterprise REST API features from Python code. These utilities can be used to assist people in managing their LucidWorks Enterprise installation, or as an example of how to write Python code as part of customer applications that will interact with LucidWorks and Solr.

Dependencies

All of these tools require that the "httplib2" library be available.

All of these tools assume that the main URL for LWE is "http://localhost:8888"
and that the URL for the UI is "http://localhost:8989"

If LWE 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 Enterprise
2. Create a data source using files on the same server as LWE:

```
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 Enterprise
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 Enterprise
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.

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 Enterprise, 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 Enterprise that has been designed to stand alone or can be run independently from other components. LucidWorks Enterprise has two main components: LWE Core, which runs Solr, indexing, and other critical application functions 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 web pages from internal or external web 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 Enterprise.

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/java/2_3_2/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 Enterprise, the master continues to manage updates to the index, while queries are handled by the slaves. This approach enables LucidWorks Enterprise to properly manage query load and ensure responsiveness.

REST API
An alternative way of controlling LucidWorks Enterprise 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 Enterprise, 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 Enterprise, 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`.