

## Yioop Documentation v 3.2

1. [Overview](#)
  - a. [Getting Started](#)
  - b. [Introduction](#)
  - c. [Feature List](#)
2. [Set-up](#)
  - a. [Requirements](#)
  - b. [Installation and Configuration](#)
  - c. [Optional Server and Security Configurations](#)
  - d. [Upgrading Yioop](#)
  - e. [Summary of Files and Folders](#)
3. [Search and User Interface](#)
  - a. [Search Basics](#)
  - b. [Search Tools Page](#)
  - c. [Search Operators](#)
  - d. [Result Formats](#)
  - e. [Settings](#)
  - f. [Mobile Interface](#)
4. [User Accounts and Social Features](#)
  - a. [Registration and Signin](#)
  - b. [Managing Accounts](#)
  - c. [Managing Users, Roles, and Groups](#)
  - d. [Feeds and Wikis](#)
5. [Keyword Advertising](#)
  - a. [The Manage Advertisements Activity](#)
  - b. [Bid Mechanics](#)
  - c. [Payment Processing](#)
6. [Crawling and Customizing Results](#)
  - a. [Performing and Managing Crawls](#)
  - b. [Mixing Crawl Indexes](#)
  - c. [Classifying Web Pages](#)
  - d. [Page Indexing and Search Options](#)
  - e. [Results Editor](#)
  - f. [Search Sources](#)
  - g. [GUI for Managing Machines and Servers](#)
7. [Building Sites with Yioop](#)
  - a. [Building a Site using Yioop's Wiki System](#)
  - b. [Building a Site using Yioop as Framework](#)
  - c. [Embedding Yioop in an Existing Site](#)
  - d. [Accessing Yioop and getting and OpenSearch RSS, JSON, or JSONP Response](#)
  - e. [Accessing Yioop via the Function API](#)
  - f. [Localizing Yioop to a New Language](#)
8. [Advanced Topics](#)
  - a. [Modifying Yioop Code](#)
  - b. [Yioop Command-line Tools](#)
9. [References](#)

## OVERVIEW

### Getting Started

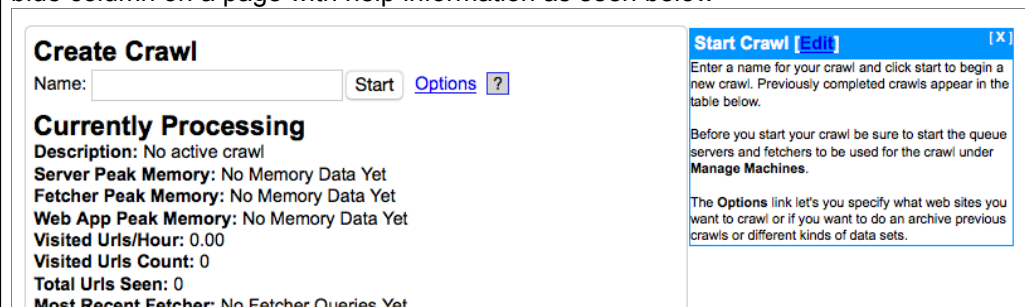
If you have downloaded Version 2.1 of the software you may prefer to use a capture of the [Version 2.10 Documentation-PDF](#) .

This document serves as a detailed reference for the Yioop search engine. If you want to get started using Yioop now, you probably want to first read the [Installation Guides](#) page and look at the [Yioop Video Tutorials Wiki](#). If you cannot find your particular machine configuration there, you can check the Yioop [Requirements](#) section followed by the more general Installation and Configuration instructions.

[Yioop.com](#) , the demo site for Yioop software, allows people to register accounts. Once registered, if you have questions about Yioop and its installation, you can join the [Yioop Software Help](#) group and post your questions there. This group is frequently checked by the creators of Yioop, and you will likely get a quick response.

Having a Yioop account also allows you to experiment with some of the features of Yioop beyond search such as Yioop Groups, Wikis, and Crawl Mixes without needing to install the software yourself. The [Search and User Interface](#) , [Managing Users, Roles, and Groups](#) , [Feeds and Wikis](#) , and [Mixing Crawl Indexes](#) sections below could serve as a guide to testing the portion of the site general users have access to on Yioop.com.

When using Yioop software and do not understand a feature, make sure to also check out the integrated help system throughout Yioop. Clicking on a question mark icon will reveal an additional blue column on a page with help information as seen below



The screenshot shows the 'Create Crawl' interface. On the left, there is a form with a 'Name:' input field, a 'Start' button, and an 'Options ?' link. Below the form, the status is 'Currently Processing' with a description: 'No active crawl'. Other statistics shown include 'Server Peak Memory: No Memory Data Yet', 'Fetcher Peak Memory: No Memory Data Yet', 'Web App Peak Memory: No Memory Data Yet', 'Visited Uris/Hour: 0.00', 'Visited Uris Count: 0', and 'Total Uris Seen: 0'. At the bottom, it says 'Most Recent Fetcher: No Fetcher Queries Yet'. On the right, a blue tooltip titled 'Start Crawl [Edit]' is visible, containing instructions: 'Enter a name for your crawl and click start to begin a new crawl. Previously completed crawls appear in the table below.' It also mentions 'Before you start your crawl be sure to start the queue servers and fetchers to be used for the crawl under Manage Machines.' and 'The Options link let's you specify what web sites you want to crawl or if you want to do an archive previous crawls or different kinds of data sets.'

### Introduction

The Yioop search engine is designed to allow users to produce indexes of a web-site or a collection of web-sites. The number of pages a Yioop index can handle range from small site to those containing tens or hundreds of millions of pages. In contrast, a search-engine like Google maintains an index of tens of billions of pages. Nevertheless, since you, the user, have control over the exact sites which are being indexed with Yioop, you have much better control over the kinds of results that a search will return. Yioop provides a traditional web interface to do queries, an rss api, and a function api. It also supports many common features of a search portal such as user discussion group, blogs, wikis, and a news aggregator. In this section we discuss some of the different search engine technologies which exist today, how Yioop fits into this eco-system, and when Yioop might be the right choice for your search engine needs. In the remainder of this document after the introduction, we discuss how to get and install Yioop; the files and folders used in Yioop; the various crawl, search, social portal, and administration facilities in the Yioop; localization in the Yioop system; building a site using the Yioop framework; embedding Yioop in an existing web-site; customizing Yioop; and the Yioop command-line tools.

Since the mid-1990s a wide variety of search engine technologies have been explored. Understanding some of this history is useful in understanding Yioop capabilities. In 1994, Web Crawler, one of the earliest still widely-known search engines, only had an index of about 50,000 pages which was stored in an Oracle database. Today, databases are still used to create indexes for small to medium size sites. An example of such a search engine written in PHP is [Sphider](#). Given that a database is being used, one common way to associate a word with a document is to use a

unique words per page, this table's size would need to be in the hundreds of millions for even a million page index. This edges towards the limits of the capabilities of database systems although techniques like table sharding can help to some degree. The Yioop engine uses a database to manage some things like users and roles, but uses its own web archive format and indexing technologies to handle crawl data. This is one of the reasons that Yioop can scale to larger indexes.

When a site that is being indexed consists of dynamic pages rather than the largely static page situation considered above, and those dynamic pages get most of their text content from a table column or columns, different search index approaches are often used. Many database management systems like [MySQL/MariaDB](#), support the ability to create full text indexes for text columns. A faster more robust approach is to use a stand-alone full text index server such as [Sphinx](#). However, for these approaches to work the text you are indexing needs to be in a database column or columns, or have an easy to define "XML mapping". Nevertheless, these approaches illustrate another common thread in the development of search systems: Search as an appliance, where you either have a separate search server and access it through either a web-based API or through function calls.

Yioop has both a search function API as well as a web API that can return [Open Search RSS](#) results or a JSON variant. These can be used to embed Yioop within your existing site. If you want to create a new search engine site, Yioop provides all the basic features of web search portal. It has its own account management system with the ability to set up groups that have both discussions boards and wikis with various levels of access control. The built in Public group's wiki together with the GUI configure page can be used to completely customize the look and feel of Yioop. Third party display ads can also be added through the GUI interface. If you want further customization, Yioop offers a web-based, model-view-adapter (a variation on model-view-controller) framework with a web-interface for localization.

By 1997 commercial sites like Inktomi and AltaVista already had tens or hundreds of millions of pages in their indexes [ [P1994](#) ] [ [P1997a](#) ] [ [P1997b](#) ]. Google [ [BP1998](#) ] circa 1998 in comparison had an index of about 25 million pages. These systems used many machines each working on parts of the search engine problem. On each machine there would, in addition, be several search related processes, and for crawling, hundreds of simultaneous threads would be active to manage open connections to remote machines. Without threading, downloading millions of pages would be very slow. Yioop is written in [PHP](#). This language is the 'P' in the very popular [LAMP](#) web platform. This is one of the reasons PHP was chosen as the language of Yioop. Unfortunately, PHP does not have built-in threads. However, the PHP language does have a multi-curl library (implemented in C) which uses threading to support many simultaneous page downloads. This is what Yioop uses. Like these early systems Yioop also supports the ability to distribute the task of downloading web pages to several machines. As the problem of managing many machines becomes more difficult as the number of machines grows, Yioop further has a web interface for turning on and off the processes related to crawling on remote machines managed by Yioop.

There are several aspects of a search engine besides downloading web pages that benefit from a distributed computational model. One of the reasons Google was able to produce high quality results was that it was able to accurately rank the importance of web pages. The computation of this page rank involves repeatedly applying Google's normalized variant of the web adjacency matrix to an initial guess of the page ranks. This problem naturally decomposes into rounds. Within a round the Google matrix is applied to the current page ranks estimates of a set of sites. This operation is reasonably easy to distribute to many machines. Computing how relevant a word is to a document is another task that benefits from multi-round, distributed computation. When a document is processed by indexers on multiple machines, words are extracted and a stemming algorithm such as [ [P1980](#) ] or a character n-gramming technique might be employed (a stemmer would extract the word jump from words such as jumps, jumping, etc; converting jumping to 3-grams would make terms of length 3, i.e., jum, ump, mpi, pin, ing). For some languages like Chinese, where spaces between words are not always used, a segmenting algorithm like reverse maximal match might be used. Next a statistic such as BM25F [ [ZCTSR2004](#) ] (or at least the non-query time part of it) is computed to determine the importance of that word in that document compared to that word amongst all other documents. To do this calculation one needs to compute global statistics concerning all documents seen, such

might take one or more merge rounds to compute these statistics based on partial computations on many machines. Hence, each of these computations benefit from allowing distributed computation to be multi-round. Infrastructure such as the Google File System [ [GGL2003](#) ], the MapReduce model [ [DG2004](#) ], and the Sawzall language [ [PDGQ2006](#) ] were built to make these multi-round distributed computation tasks easier. In the open source community, the [Hadoop Distributed File System](#), [Hadoop MapReduce](#), and [Pig](#) play an analogous role [ [W2009](#) ]. Recently, a theoretical framework for what algorithms can be carried out as rounds of map inputs to sequence of key value pairs, shuffle pairs with same keys to the same nodes, reduce key-value pairs at each node by some computation has begun to be developed [ [KSV2010](#) ]. This framework shows the map reduce model is capable of solving quite general cloud computing problems -- more than is needed just to deploy a search engine.

Infrastructure such as this is not trivial for a small-scale business or individual to deploy. On the other hand, most small businesses and homes do have available several machines not all of whose computational abilities are being fully exploited. So the capability to do distributed crawling and indexing in this setting exists. Further high-speed internet for homes and small businesses is steadily getting better. Since the original Google paper, techniques to rank pages have been simplified [ [APC2003](#) ]. It is also possible to approximate some of the global statistics needed in BM25F using suitably large samples. More details on the exact ranking mechanisms used by Yioop and be found on the [Yioop Ranking Mechanisms](#) page.

Yioop tries to exploit these advances to use a simplified distributed model which might be easier to deploy in a smaller setting. Each node in a Yioop system is assumed to have a web server running. One of the Yioop nodes web app's is configured to act as a coordinator for crawls. It is called the **name server** . In addition to the name server, one might have several processes called **queue servers** that perform scheduling and indexing jobs, as well as **fetcher** processes which are responsible for downloading pages and the page processing such as stemming, char-gramming and segmenting mentioned above. Through the name server's web app, users can send messages to the queue servers and fetchers. This interface writes message files that queue servers periodically looks for. Fetcher processes periodically ping the name server to find the name of the current crawl as well as a list of queue servers. Fetcher programs then periodically make requests in a round-robin fashion to the queue servers for messages and schedules. A schedule is data to process and a message has control information about what kind of processing should be done. A given queue server is responsible for generating schedule files for data with a certain hash value, for example, to crawl urls for urls with host names that hash to queue server's id. As a fetcher processes a schedule, it periodically POSTs the result of its computation back to the responsible queue server's web server. The data is then written to a set of received files. The queue server as part of its loop looks for received files and merges their results into the index so far. So the model is in a sense one round: URLs are sent to the fetchers, summaries of downloaded pages are sent back to the queue servers and merged into their indexes. As soon as the crawl is over one can do text search on the crawl. Deploying this computation model is relatively simple: The web server software needs to be installed on each machine, the Yioop software (which has the the fetcher, queue server, and web app components) is copied to the desired location under the web server's document folder, each instance of Yioop is configured to know who the name server is, and finally, the fetcher programs and queue server programs are started.

As an example of how this scales, 2010 Mac Mini running a queue server program can schedule and index about 100,000 pages/hour. This corresponds to the work of about 7 fetcher processes (which may be on different machines -- roughly, you want 1GB and 1core/fetcher). The checks by fetchers on the name server are lightweight, so adding another machine with a queue server and the corresponding additional fetchers allows one to effectively double this speed. This also has the benefit of speeding up query processing as when a query comes in, it gets split into queries for each of the queue server's web apps, but the query only "looks" slightly more than half as far into the posting list as would occur in a single queue server setting. To further increase query throughput, the number queries that can be handled at a given time, Yioop installations can also be configured as "mirrors" which keep an exact copy of the data stored in the site being mirrored. When a query

multi-word queries can actually be a major bottleneck if you don't have many machines and you do have a large index. To further speed this up, Yioop uses a hybrid inverted index/suffix tree approach to store word lookups. The suffix tree ideas being motivated by [ [PTSHVC2011](#) ].

Since a multi-million page crawl involves both downloading from the web rapidly over several days, Yioop supports the ability to dynamically change its crawl parameters as a crawl is going on. This allows a user on request from a web admin to disallow Yioop from continuing to crawl a site or to restrict the number of urls/hours crawled from a site without having to stop the overall crawl. One can also through a web interface inject new seed sites, if you want, while the crawl is occurring. This can help if someone suggests to you a site that might otherwise not be found by Yioop given its original list of seed sites. Crawling at high-speed can cause a website to become congested and unresponsive. As of Version 0.84, if Yioop detects a site is becoming congested it can automatically slow down the crawling of that site. Finally, crawling at high-speed can cause your domain name server (the server that maps [www.yioop.com](http://www.yioop.com) to 173.13.143.74) to become slow. To reduce the effect of this Yioop supports domain name caching.

Despite its simpler one-round model, Yioop does a number of things to improve the quality of its search results. While indexing, Yioop can make use Lasso regression classifiers [ [GLM2007](#) ] using data from earlier crawls to help label and/or rank documents in the active crawl. Yioop also takes advantage of the link structure that might exist between documents in a one-round way: For each link extracted from a page, Yioop creates a micropage which it adds to its index. This includes relevancy calculations for each word in the link as well as an [ [APC2003](#) ]-based ranking of how important the link was. Yioop supports a number of iterators which can be thought of as implementing a stripped-down relational algebra geared towards word-document indexes (this is much the same idea as Pig). One of these operators allows one to make results from unions of stored crawls. This allows one to do many smaller topic specific crawls and combine them with your own weighting scheme into a larger crawl. A second useful operator allows you to display a certain number of results from a given subquery, then go on to display results from other subqueries. This allows you to make a crawl presentation like: the first result should come from the open crawl results, the second result from Wikipedia results, the next result should be an image, and any remaining results should come from the open search results. Yioop comes with a GUI facility to make the creation of these crawl mixes easy. To speed up query processing for these crawl mixes one can also create materialized versions of crawl mix results, which makes a separate index of crawl mix results. Another useful operator Yioop supports allows one to perform groupings of document results. In the search results displayed, grouping by url allows all links and documents associated with a url to be grouped as one object. Scoring of this group is a sum of all these scores. Thus, link text is used in the score of a document. How much weight a word from a link gets also depends on the link's rank. So a low-ranked link with the word "stupid" to a given site would tend not to show up early in the results for the word "stupid". Grouping also is used to handle deduplication: It might be the case that the pages of many different URLs have essentially the same content. Yioop creates a hash of the web page content of each downloaded url. Amongst urls with the same hash only the one that is linked to the most will be returned after grouping. Finally, if a user wants to do more sophisticated post-processing such as clustering or computing page rank, Yioop supports a straightforward architecture for indexing plugins.

There are several open source crawlers which can scale to crawls in the millions to hundred of millions of pages. Most of these are written in Java, C, C++, C#, not PHP. Three important ones are [Nutch](#)/ [Lucene](#)/ [Solr](#) [ [KC2004](#) ], [YaCy](#), and [Heritrix](#) [ [MKSR2004](#) ]. Nutch is the original application for which the Hadoop infrastructure described above was developed. Nutch is a crawler, Lucene is for indexing, and Solr is a search engine front end. The YaCy project uses an interesting distributed hash table peer-to-peer approach to crawling, indexing, and search. Heritrix is a web crawler developed at the [Internet Archive](#). It was designed to do archival quality crawls of the web. Its ARC file format inspired the use of WebArchive objects in Yioop. WebArchives are Yioop's container file format for storing web pages, web summary data, url lists, and other kinds of data used by Yioop. A WebArchive is essentially a linked-list of compressed, serialized PHP objects, the last element in this list containing a header object with information like version number and a total count of objects

can be used to store auxiliary data structures into the list if desired. One nice aspect of serialized PHP objects versus serialized Java Objects is that they are humanly readable text strings. The main purpose of Web Archives is to allow one to store many small files compressed as one big file. They also make data from web crawls very portable, making them easy to copy from one location to another. Like Nutch and Heritrix, Yioop also has a command-line tool for quickly looking at the contents of such archive objects.

The [ARC format](#) is one example of an archival file format for web data. Besides at the Internet Archive, ARC and its successor [WARC format](#) are often used by TREC conferences to store test data sets such as [GOV2](#) and the [ClueWeb 2009 / ClueWeb 2012](#) Datasets. In addition, it was used by grub.org (hopefully, only on a temporary hiatus), a distributed, open-source, search engine project in C#. Another important format for archiving web pages is the XML format used by Wikipedia for archiving MediaWiki wikis. [Wikipedia](#) offers [creative common-licensed downloads](#) of their site in this format. The [Open Directory Project](#) makes available its [ODP data set](#) in an RDF-like format licensed using the Open Directory License. Thus, we see that there are many large scale useful data sets that can be easily licensed. Raw data dumps do not contain indexes of the data though. This makes sense because indexing technology is constantly improving and it is always possible to re-index old data. Yioop supports importing and indexing data from ARC, WARC, database queries results, MediaWiki XML dumps, and Open Directory RDF. Yioop further has a generic text importer which can be used to index log records, mail, Usenet posts, etc. Yioop also supports re-indexing of old Yioop data files created after version 0.66, and indexing crawl mixes. This means using Yioop you can have searchable access to many data sets as well as have the ability to maintain your data going forward. When displaying caches of web pages in Yioop, the interface further supports the ability to display a history of all cached copies of that page, in a similar fashion to Internet Archives interface.

Another important aspect of creating a modern search engine is the ability to display in an appropriate way various media sources. Yioop comes with built-in subsearch abilities for images, where results are displayed as image strips; video, where thumbnails for video are shown; and news, where news items are grouped together and a configurable set of news/twitter feeds can be set to be updated on an hourly basis.

This concludes the discussion of how Yioop fits into the current and historical landscape of search engines and indexes.

## Feature List

Here is a summary of the features of Yioop:

### • General

- Yioop is an open-source, distributed crawler and search engine written in PHP.
- Crawling, indexing, and serving search results can be done on a single machine or distributed across several machines.
- The fetcher/queue server processes on several machines can be managed through the web interface of a main Yioop instance.
- Yioop installations can be created with a variety of topologies: one queue server and many fetchers or several queue servers and many fetchers.
- Using web archives, crawls can be mirrored amongst several machines to speed-up serving search results. This can be further sped-up by using memcache or filecache.
- Yioop can be used to create web sites via its own built-in wiki system. For more complicated customizations, Yioop's model-view-adapter framework is designed to be easily extendible. This framework also comes with a GUI which makes it easy to localize strings and static pages.
- Yioop can be used as a package in other project using [Composer](#) .
- Yioop search result and feed pages can be configured to display banner or skyscraper ads through an Site Admin GUI (if desired).
- Yioop search result and feed pages can also be configured to use Yioop's built-in keyword

- Yioop has been optimized to work well with smart phone web browsers and with tablet devices.
- **Social and User Interface**
  - Yioop can be configured to allow or not to allow users to register for accounts.
  - If allowed, user accounts can create discussion groups, blogs, and wikis.
  - Blogs and wiki support attaching images, videos, and files and also support including math using LaTeX or AsciiMathML.
  - Yioop comes with two built in groups: Help and Public. Help's wiki pages allow one to customize the integrated help throughout the Yioop system. The Public Groups discussion can be used as a site blog, its wiki page can be used to customize the look-and-feel of the overall Yioop site without having to do programming.
  - Wiki pages support different types such as standard wiki page, page alias, media gallery, and slide presentation.
  - Video on wiki pages and in discussion posts is served using HTTP-pseudo streaming so users can scrub through video files. For uploaded video files below a configurable size limit, videos are automatically converted to web friendly mp4 format, provided the distributed version of the media updater is in use.
  - Wiki pages can be configured to have auto-generated tables of contents, to make use of common headers and footers, and to output meta tags for SEO purposes.
  - Users can share their own mixes of crawls that exist in the Yioop system.
  - If user accounts are enabled, Yioop has a search tools page on which people can suggest urls to crawl.
  - Yioop has three different captcha'ing mechanisms that can be used in account registration and for suggest urls: a standard graphics-based captcha, a text-based captcha, and a hash-cash-like catpha.
  - Password authentication can be configured to either use a standard password hash based system, or make use of Fiat Shamir zero-knowledge authentication.
- **Search**
  - Yioop supports subsearches geared towards presenting certain kinds of media such as images, video, and news. The list of video and news sites can be configured through the GUI. Yioop has a media updater process which can be used to automatically update news feeds hourly.
  - News feeds can either be RSS or Atom feed or can be scraped from an HTML page using XPath queries. What image is used for a news feed item can also be configured using XPath queries.
  - Several hourly processes such as news updates, video conversion, and mail delivery can be configured to be distributed across several machines automatically.
  - Yioop determines search results using a number of iterators which can be combined like a simplified relational algebra.
  - Yioop can be configured to display word suggestions as a user types a query. It can also suggest spell corrections for mis-typed queries. This feature can be localized.
  - Yioop can also make use of a thesaurus facility such as provided by WordNet to suggest related queries.
  - Yioop supports the ability to filter out urls from search results after a crawl has been performed. It also has the ability to edit summary information that will be displayed for urls.
  - A given Yioop installation might have several saved crawls and it is very quick to switch between any of them and immediately start doing text searches.
  - Besides the standard output of a web page with ten links, Yioop can output query results in Open Search RSS format, a JSON variant of this format, and also to query Yioop data via a function api.
- **Indexing**
  - Yioop is capable of indexing small sites to sites or collections of sites containing low hundreds of millions of documents.
  - Yioop uses a hybrid inverted index/suffix tree approach for word lookup to make multi-word queries faster on disk bound machines.

scheme called Modified9 is used.

- Yioop has a web interface which makes it easy to combine results from several crawl indexes to create unique result presentations. These combinations can be done in a conditional manner using "if:" meta words.
- Yioop supports the indexing of many different filetypes including: HTML, Atom, BMP, DOC, DOCX, ePub, GIF, JPG, PDF, PPT, PPTX, PNG, RSS, RTF, sitemaps, SVG, XLSX, and XML. It has a web interface for controlling which amongst these filetypes (or all of them) you want to index. It supports also attempting to extract information from unknown filetypes.
- Yioop supports extracting data from zipped formats like DOCX even if it only did a partial download of the file.
- Yioop has a simple page rule language for controlling what content should be extracted from a page or record.
- Yioop has three different kinds of text summarizers which can be used to further affect what words are index: a basic web page scraper, a graph-base summarizer, and a centroid algorithm summarizer. The latter can be used to generate word clouds of crawled documents.
- Indexing occurs as crawling happens, so when a crawl is stopped, it is ready to be used to handle search queries immediately.
- Yioop Indexes can be used to create classifiers which then can be used in labeling and ranking future indexes.
- Yioop comes with stemmers for English, Arabic, Dutch, French, German, Hindi, Italian, Persian, Portuguese, Russian, and Spanish, and a word segmenter for Chinese. It uses char-gramming for other languages. Yioop has a simple architecture for adding stemmers for other languages.
- Yioop uses a web archive file format which makes it easy to copy crawl results amongst different machines. It has a command-line tool for inspecting these archives if they need to be examined in a non-web setting. It also supports command-line search querying of these archives.
- Yioop supports an indexing plugin architecture to make it possible to write one's own indexing modules that do further post-processing.
- **Web and Archive Crawling**
  - Yioop supports open web crawls, but through its web interface one can configure it also to crawl only specific site, domains, or collections of sites and domains. One can customize a crawl using regex in disallowed directives to crawl a site to a fixed depth.
  - Yioop uses multi-curl to support many simultaneous downloads of pages.
  - Yioop obeys robots.txt files including Google and Bing extensions such as the Crawl-delay and Sitemap directives as well as \* and \$ in allow and disallow. It further supports the robots meta tag directives NONE, NOINDEX, NOFOLLOW, NOARCHIVE, and NOSNIPPET and the link tag directive rel="canonical". It also supports anchor tags with rel="nofollow" attributes. It also supports X-Robots-Tag HTTP headers. Finally, it tries to detect if a robots.txt became a redirect due to congestion.
  - Yioop comes with a word indexing plugin which can be used to control how Yioop crawls based on words on the page and the domain. This is useful for creating niche subject specific indexes.
  - Yioop has its own DNS caching mechanism and it adjusts the number of simultaneous downloads it does in one go based on the number of lookups it will need to do.
  - Yioop can crawl over HTTP, HTTPS, and Gopher protocols.
  - Yioop supports crawling TOR networks (.onion urls).
  - Yioop supports crawling through a list of proxy servers.
  - Yioop supports crawling Git Repositories and can index Java and Python code.
  - Yioop supports crawl quotas for web sites. I.e., one can control the number of urls/hour downloaded from a site.
  - Yioop can detect website congestion and slow down crawling a site that it detects as congested.
  - Yioop supports dynamically changing the allowed and disallowed sites while a crawl is in



the active crawl.

- o Yioop has a web form that allows a user to control the recrawl frequency for a page during a crawl.
- o Yioop keeps track of ETag: and Expires: HTTP headers to avoid downloading content it already has in its index.
- o Yioop supports importing data from ARC, WARC, database queries, MediaWiki XML, and ODP RDF files. It has generic importing facility to import text records such as access log, mail log, usenet posts, etc., which are either not compressed, or compressed using gzip or bzip2. It also supports re-indexing of data from WebArchives.

[Return to table of contents](#) .

## Set-up Requirements

The Yioop search engine requires: (1) a web server, (2) PHP 5.4 or better, (3) Curl libraries for downloading web pages, (4) mod\_rewrite enabled. To be a little more specific Yioop has been tested with Apache 2.2 and I've been told Version 0.82 or newer works with lighttpd. It should work with other web servers, although it might take some finessing. For PHP, you need a build of PHP that incorporates multi-byte string (mb\_ prefixed) functions, Curl, Sqlite (or at least PDO with Sqlite driver), the GD graphics library and the command-line interface. If you are using Mac OSX Snow Leopard or newer, the version of Apache2 and PHP that come with it suffice. For Windows, Mac, and Linux, another easy way to get the required software is to download a Apache/PHP/MySQL suite such as [XAMPP](#).

On any platform, make sure that the Apache mod\_rewrite module is enabled and that .htaccess files work for the directory in question (often this is the case by default with packages such as Xampp, but please check the Yioop install guide for the platform in question in order to see if anything is required). It is possible to get Yioop to work without mod\_rewrite, you can use the src directory location as the url for Yioop, but URLs used by your installation will look uglier.

On Windows machines, find the the php.ini file under the php folder in your Xampp folder and change the line:

```
;extension=php_curl.dll
```

to

```
extension=php_curl.dll
```

The php.ini file has a post\_max\_size setting you might want to change. You might want to change it to:

```
post_max_size = 32M
```

Yioop will work with the post\_max\_size set to as little as two megabytes bytes, but will be faster with the larger post capacity. If you intend to make use of Yioop Discussion Groups and Wiki and their ability to upload documents. You might want to consider also adjusting the value of the variable *upload\_max\_filesize* . This value should be set to at most what you set post\_max\_size to.

If you are using WAMP, similar changes as with XAMPP must be made, but be aware that WAMP has two php.ini files and both of these must be changed.

If you are using the Ubuntu-variant of Linux, the following lines would get the software you need:

```
sudo apt-get install curl
sudo apt-get install apache2
sudo apt-get install php5
sudo apt-get install php5-cli
sudo apt-get install php5-sqlite
sudo apt-get install php5-curl
sudo apt-get install php5-gd
```

For both Mac and Linux, you might want to alter the post\_max\_size variable in your php.ini file as in the Windows case above.

feature in Yioop, you might need to do some additional configuration. Namely, some sites disable the PHP fopen, fclose, and exec functions, and so might have to edit your php.ini file to enable these functions if you want this activity to work. The Manage Machines activity allows you through a web interface to start/stop and look at the log files for each of the QueueServer's, and Fetchers that you want Yioop to manage. It also allows you to start and stop the Media Updater process/processes. If it is not configured then these task would need to be done via the command line. **Also, if you do not use the Manage Machine interface your Yioop site can make use of only one QueueServer.**

As a final step, after installing the necessary software, **make sure to start/restart your web server and verify that it is running.**

### Memory Requirements

In addition, to the prerequisite software listed above, Yioop allows specifies for its process certain upper bounds on the amounts of memory they can use. By default src/executables /QueueServer.php's limit is set to 2500MB, src/executables/Fetcher.php's limit is set to 1200MB. You can expect that index.php might need up to 500MB. These values are set near the tops of each of these files in turn with a line like:

```
ini_set("memory_limit","2500M");
```

For the index.php file, you may need to set the limit at well in your php.ini file for the instance of PHP used by your web server. If the value is too low for the index.php Web app you might see messages in the Fetcher logs that begin with: "Trouble sending to the scheduler at url..."

Often in a VM setting these requirements are somewhat steep. It is possible to get Yioop to work in environments like EC2 (be aware this might violate your service agreement). To reduce these memory requirements, one can manually adjust the variables NUM\_DOCS\_PER\_GENERATION, SEEN\_URLS\_BEFORE\_UPDATE\_SCHEDULER, NUM\_URLS\_QUEUE\_RAM, MAX\_FETCH\_SIZE, and URL\_FILTER\_SIZE in the configs/config.php file. Experimenting with these values you should be able to trade-off memory requirements for speed.

[Return to table of contents](#) .

### Installation and Configuration

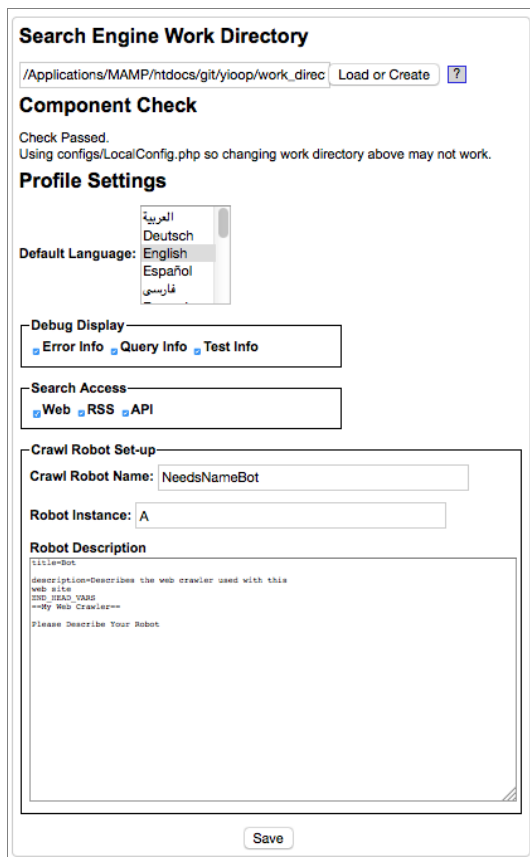
The Yioop application can be obtained using [the download page at seekquarry.com](#) . After downloading and unzipping it, move the Yioop search engine into some folder under your web server's document root. Point your browser to the location where you have just unzipped the file. You should see the default Yioop landing page:



in the [Requirements section](#) . The most common sources of problems are that mod\_rewrite package is not loaded for Apache, the .htaccess file of Yioop is not being processed because Apache is configured so as not to allow overrides on that directory, or the webserver doesn't have write permissions on YIOOP\_DIR/configs/Configs.php and the YIOOP\_DIR/work\_directory folder. If you have gotten the above screen, then congratulations, Yioop is essentially installed! In this section, we will give a first pass at how to customize some general aspects of your site. These will include configuring your site for development versus production, describing your crawler to the websites you will crawl, customizing the icons, color scheme, and timezone settings of your site. The next section, describes more advanced configurations such as user registration and advertisement serving.

### Configurations for Development, Production, Testing, and Crawling

Click on the Sign In link to log on to the administrative panel for Yioop. The default username is root with an empty password. Under the **System Settings** activity group on the side of the screen, click on the **Configure Activity** . This activity page looks like:



The screenshot shows the 'Search Engine Work Directory' configuration page. At the top, there is a text input field containing the path '/Applications/MAMP/htdocs/git/yioop/work\_direct' with 'Load or Create' and a help icon to its right. Below this is a 'Component Check' section with the text 'Check Passed. Using configs/LocalConfig.php so changing work directory above may not work.' The 'Profile Settings' section includes a 'Default Language' dropdown menu with options for العربية, Deutsch, English, Español, and فارسي. There are three expandable sections: 'Debug Display' with checkboxes for Error Info, Query Info, and Test Info; 'Search Access' with checkboxes for Web, RSS, and API; and 'Crawl Robot Set-up' which contains a 'Crawl Robot Name' field with the placeholder 'NeedsNameBot', a 'Robot Instance' field with the value 'A', and a 'Robot Description' text area. The text area contains a code block with a description of the crawler and a note to describe the robot. A 'Save' button is located at the bottom of the form.

The Search Engine Work Directory form lets you specify the name of a folder where all of the data specific to your copy of Yioop should be stored. This defaults to YIOOP\_DIR/work\_directory, and you probably don't want to change this, but you can customize its locations to where you want with this form. The web server needs permissions to be able to write this directory for Yioop to work. Notice under the text field there is a heading "Component Check" and it says "Checks Passed". Yioop does an automatic check to see if it can find all the common built-in classes and functions it needs to run, making sure they have not been disabled. If it says something other than "Checks passed", you will probably need to edit your php.ini file to get Yioop to be able to do crawls.

In the above Configure Screen image, there is a Profile Settings form beneath the Search Engine Work Directory form. The Profile Settings form allows you to configure the debug, search access, database, queue server, and robot settings.

The **Debug Display** fieldset has three checkboxes: Error Info, Query Info, and Test Info. Checking Error Info will mean that when the Yioop web app runs, any PHP Errors, Warnings, or Notices will be

production environment. The second checkbox, Query Info, when checked, will cause statistics about the time, etc. of database queries to be displayed at the bottom of each web page. The last checkbox, Test Info, says whether or not to display automated tests of some of the systems library classes if the browser is navigated to [http://YIOOP\\_INSTALLATION/tests/](http://YIOOP_INSTALLATION/tests/). None of these debug settings should be checked in a production environment.

The **Search Access** fieldset has three checkboxes: Web, RSS, and API. These control whether a user can use the web interface to get query results, whether RSS responses to queries are permitted, or whether or not the function based search API is available. Using the Web Search interface and formatting a query url to get an RSS response are describe in the Yioop [Search and User Interface](#) section. The Yioop Search Function API is described in the section [Embedding Yioop](#) , you can also look in the examples folder at the file SearchApi.php to see an example of how to use it. **If you intend to use Yioop in a configuration with multiple queue servers (not fetchers), then the RSS checkbox needs to be checked.**

The **Crawl Robot Set-up** fieldset is used to provide websites that you crawl with information about who is crawling them. The field Crawl Robot Name is used to say the name of your robot. You should choose a common name for all of the fetchers in your set-up, but the name should be unique to your web-site. It is bad form to pretend to be someone else's robot, for example, the googlebot. As Yioop crawls it sends the web-site it crawls a User-Agent string, this string contains the url back to the bot.php file in the Yioop folder. bot.php is supposed to provide a detailed description of your robot. The contents of textarea Robot Description is supposed to provide this description and is inserted between <body> </body> tags on the bot.php page.

#### **Changing the Global Appearance of Yioop**

To change the global appearance of Yioop, gnder the **System Settings** activity group on the side of the screen, click on the **Appearance** . This activity page looks like:

**Account Access**

- Manage Account
- Manage Users
- Manage Roles

**Crawls**

- Manage Crawl
- Manage Classifiers
- Page Options
- Results Editor
- Search Sources

**Social**


- Manage Groups
- Feeds and Wikis
- Mix Crawls

**System Settings**

- Manage Machines
- Manage Locales
- Server Settings
- Security
- Appearance
- Configure

Use Wiki Public Main Page as Landing Page


Background Color:


Background Image:  [Drag or ... Choose](#)


Foreground Color:


Top Bar Color:

Side Bar Color:

Site Logo:  [Drag or ... Choose](#)

Mobile Logo:  [Drag or ... Choose](#)

Favicon:  [Drag or ... Choose](#)

Search Toolbar  [Drag or ... Choose](#)

Timezone:

Web Cookie Name:

Web Token Name:

Auxiliary Style Directives

[Reset Customizations](#)

The **Use Wiki Public Main Page as Landing Page** checkbox lets you set the main page of the Public wiki to be the landing page of the whole Yioop site rather than the default centered search box landing page. Several of the text fields in Site Customizations control various colors used in drawing the Yioop interface. These include **Background Color** , **Foreground Color** , **Top Bar Color** , **Side Bar Color** . The values of these fields can be any legitimate style-sheet color such as a # followed by an red, green, blue value (between 0-9 A-F), or a color word such as: yellow, cyan, etc. If you would like to use a background image, you can either use the picker link or drag and drop one into the rounded square next to the **Background Image** label. Various other images such as the **Site Logo** , **Mobile Logo** (the logo used for mobile devices), and **Favicon** (the little logo that appears in the title tab of a page or in the url bar) can similarly be chosen or dragged-and-dropped.

A **Search Toolbar** is a short file that can be used to add your search engine to the search bar of a browser. You can drag such a file into the gray area next to this label and click save to set this for your site. The link to install the search bar is visible on the Settings page. There is also a link tag on every page of the Yioop site that allows a browser to auto-discover this as well. As a starting point, one can try tweaking the default Yioop search bar, yioopbar.xml, in the base folder of Yioop.

The three fields **Timezone** , **Web Cookie Name** , and **Web Token Name** control respectively, the timezone used by Yioop when it does date conversions, the name of the cookie it sets in a browser's cookie cache, and the name used for the tokens to prevent cross-site request forgery that appear in Yioop URLs when one is logged-in.

Finally, if one knows cascading stylesheets (CSS) and wants greater control of the the look and feel of the site, then one can enter standard stylesheet command in the **Auxiliary Style Directives** textarea.

The configuration and appearance activities just described suffices to set up Yioop for a single server crawl and do simple customization on how your Yioop site looks. If that is all that you are interested in, you may want to skip ahead to the section on the [Yioop Search Interface](#) to learn about the different search features available in Yioop or you may want to skip ahead to [Performing and Managing Crawls](#) to learn about how to perform a crawl. In this section, we describe the Server Settings and Security activities which might be useful in a multi-machine, multi-user setting and which might also be useful for crawling hidden websites or crawling through proxies. We also describe the form used to configure advertisements in Yioop.

The Server Settings activity looks like:

The screenshot shows the 'Servers and Registration' configuration page in Yioop. It is divided into several sections:

- Name Server Set-up:** Includes fields for 'Server Key' (1234), 'Name Server URL' (http://localhost/g2/yioop/), and a 'Use Filecache' checkbox.
- Database Set-up:** Includes 'Database System' (SQLite3) and 'Database Name' (seek\_query).
- Account Registration:** Includes 'Email Activation' (dropdown), 'Sender Email' (chris@pollatt.org), 'Send Mail From Media Updater' (checkbox), 'Use PHP mail() function' (checkbox), 'SMTP Server' (pollatt.org), 'Server Port' (587), 'Mail Username' (cpollatt), 'Mail Password' (masked), and 'Mail Security' (tls).
- Proxy Servers:** Includes 'Tor Proxy (onion urls only)' (127.0.0.1:9150) and a 'Crawl via Proxies' checkbox.
- Ad Server Configuration:** Includes 'Advertising Source' (External Ad Server), 'Ad Location' (radio buttons for Top, Side, Both, None), and two text areas for 'Global Element Ad Script', 'Top Element Ad Script', and 'Side Element Ad Script'.

A 'Save' button is located at the bottom of the form.

The **Name Server Set-up** fieldset is used to tell Yioop which machine is going to act as a name server during a crawl and what secret string to use to make sure that communication is being done between legitimate queue servers and fetchers of your installation. You can choose anything for your secret string as long as you use the same string amongst all of the machines in your Yioop installation. The reason why you have to set the name server url is that each machine that is going to run a fetcher to download web pages needs to know who the queue servers are so they can request a batch of urls to download. There are a few different ways this can be set-up:

1. If the particular instance of Yioop is only being used to display search results from crawls that you have already done, then this fieldset can be filled in however you want.
2. If you are doing crawling on only one machine, you can put `http://localhost/path_to_yioop/` or `http://127.0.0.1/path_to_yioop/`, where you appropriately modify "path\_to\_yioop".
3. Otherwise, if you are doing a crawl on multiple machines, use the url of Yioop on the machine that will act as the name server.

In communicating between the fetcher and the server, Yioop uses curl. Curl can be particular about redirects in the case where posted data is involved. i.e., if a redirect happens, it does not send posted data to the redirected site. For this reason, Yioop insists on a trailing slash on your queue server url. Beneath the Queue Server Url field, is a Memcached checkbox and a Filecache checkbox. Only one of these can be checked at a time. The Memcached check box only shows if you have [PHP Memcache](#) installed. Checking the Memcached checkbox allows you to specify

as index pages that have been accessed. Checking the Filecache box, tells Yioop to cache search query results in temporary files. Memcache probably gives a better performance boost than Filecaching, but not all hosting environments have Memcache available.

The **Database Set-up** fieldset is used to specify what database management system should be used, how it should be connected to, and what user name and password should be used for the connection. At present [PDO](#) (PHP's generic DBMS interface), sqlite3, and Mysql databases are supported. The database is used to store information about what users are allowed to use the admin panel and what activities and roles these users have. Unlike many database systems, if an sqlite3 database is being used then the connection is always a file on the current filesystem and there is no notion of login and password, so in this case only the name of the database is asked for. For sqlite, the database is stored in WORK\_DIRECTORY/data. For single user settings with a limited number of news feeds, sqlite is probably the most convenient database system to use with Yioop. If you think you are going to make use of Yioop's social functionality and have many users, feeds, and crawl mixes, using a system like Mysql or Postgres might be more appropriate.

If you would like to use a different DBMS than Sqlite or Mysql, then the easiest way is to select PDO as the Database System and for the Hostname given use the DSN with the appropriate DBMS driver. For example, for Postgres one might have something like:

```
pgsql:host=localhost;port=5432;dbname=test;user=bruce;password=mypass
```

You can put the username and password either in the DSN or in the Username and Password fields. The database name field must be filled in with the name of the database you want to connect to. It is also include needs to be included in the dsn, as in the above. PDO and Yioop has been tested to work with Postgres and sqlite, for other DBMS's it might take some tinkering to get things to work.

When switching database information, Yioop checks first if a usable database with the user supplied data exists. If it does, then it uses it; otherwise, it tries to create a new database. Yioop comes with a small sqlite demo database in the data directory and this is used to populate the installation database in this case. This database has one account root with no password which has privileges on all activities. Since different databases associated with a Yioop installation might have different user accounts set-up after changing database information you might have to sign in again.

The **Account Registration** fieldset is used to control how user's can obtain accounts on a Yioop installation. The dropdown at the start of this fieldset allows one to select one of four possibilities: Disable Registration, users cannot register themselves, only the root account can add users; No Activation, user accounts are immediately activated once a user signs up; Email Activation, after registering, users must click on a link which comes in a separate email to activate their accounts; and Admin Activation, after registering, an admin account must activate the user before the user is allowed to use their account. When Disable Registration is selected, the Suggest A Url form and link on the tool.php page is disabled as well, for all other registration type this link is enabled. If Email Activation is chosen, then the reset of this fieldset can be used to specify the email address that the email comes to the user. The Send Mail From Media Updater checkbox controls whether emails are sent immediately from the web app or if they are queued, and then sent out by the Media Updater process/processes. The checkbox Use PHP mail() function controls whether to use the mail function in PHP to send the mail, this only works if mail can be sent from the local machine. Alternatively, if this is not checked like in the image above, one can configure an outgoing SMTP server to send the email through.

The **Proxy Server** fieldset is used to control which proxies to use while crawling. By default Yioop does not use any proxies while crawling. A Tor Proxy can serve as a gateway to the Tor Network. Yioop can use this proxy to download .onion URLs on the [Tor network](#). The configuration given in the example above works with the Tor Proxy that comes with the [Tor Browser](#). Obviously, this proxy needs to be running though for Yioop to make use of it. Beneath the Tor Proxy input field is a checkbox labelled Crawl via Proxies. Checking this box, will reveal a textarea labelled Proxy Servers. You can enter the address:port or address:port:proxytype of proxy servers you would like to crawl through. If proxy servers are used, Yioop will make any requests to download pages to a randomly chosen server on the list which will proxy the request to the site which has the page to

different ip (and potentially location) than it actually is. In practice, servers can often use HTTP headers to guess that a proxy is being used.

The **Ad Server Configuration** fieldset can be used to specify whether advertising will be served with Yioop search results, and if it is, what kind. The Advertising Source dropdown controls this, allowing the administrator to choose between **No Advertisements**, **Keyword Advertisements**, and **External Ad Server**. Yioop's built-in mechanism for keyword advertising is described in the [Keyword Advertising](#) section. External Ad Server refers to advertising scripts (such as Google Ad Words, Bidvertiser, Adspeed, etc) which are to be added on search result pages or on discussion thread pages. There are four possible placements of ads: None -- don't display advertising at all, Top -- display banner ads beneath the search bar but above search results, Side -- display skyscraper Ads in a column beside the search results, and Both -- display both banner and skyscraper ads. Choosing any option other than None reveals text areas where one can insert the Javascript one would get from the Ad network. The **Global Ad Script** text area is used for any Javascript or HTML the Ad provider wants you to include in the HTML head tag for the web page (many advertisers don't need this).

The Security activity looks like:

### Authentication and Captcha Types

Authentication Type

Captcha Type

### Captcha and Recovery Questions

[\[Edit Account Recovery Questions\]](#)

[\[Edit Text Captcha Questions\]](#)

The **Authentication Type** fieldset is used to control the protocol used to log people into Yioop. This can either be Normal Authentication, passwords are checked against stored as salted hashes of the password; or ZKP (zero knowledge protocol) authentication, the server picks challenges at random and send these to the browser the person is logging in from, the browser computes based on the password an appropriate response according to the [Fiat Shamir](#) protocol. The password is never sent over the internet and is not stored on the server. These are the main advantages of ZKP, its drawback is that it is slower than Normal Authentication as to prove who you are with a low probability of error requires several browser-server exchanges. You should choose which authentication scheme you want before you create many users as if you switch everyone will need to get a new password.

The **Captcha Type** fieldset controls what kind of [captcha](#) will be used during account registration, password recovery, and if a user wants to suggest a url. The captcha type only has an effect if under the Server Settings activity, Account Registration is not set to Disable Registration. The choices for captcha are: Text Captcha, the user has to select from a series of dropdown answers to questions of the form: Which in the following list is the most/largest/etc? or Which is the following list is the least/smallest/etc?; Graphic Captcha, the user needs to enter a sequence of characters from a



to extend a random string with additional characters to get a string whose hash begins with a certain lead set of characters. Of these, Hash Captcha is probably the least intrusive but requires Javascript and might run slowly on older browsers. A text captcha might be used to test domain expertise of the people who are registering for an account. Finally, the graphic captcha is probably the one people are most familiar with.

The Captcha and Recovery Questions section of the Security activity provides links to edit the Text Captcha and Recovery Questions for the current locale (you can change the current locale in Settings). In both cases, there are a fixed list of tests you can localize. A single test consists of a more question, a less question, and a comma separate list of possibilities. For example,

```
Which lives or lasts the longest?  
Which lives or lasts the shortest?  
lightning,bacteria,ant,dog,horse,person,oak tree,planet,star,galaxy
```

When challenging a user, Yioop picks a subset of tests. For each test, it randomly chooses between more or less question. It then picks a subset of the ordered list of choices, randomly permutes them, and presents them to the user in a dropdown.

Yioop's captcha-ing system tries to prevent attacks where a machine quickly tries several possible answers to a captcha. Yioop has a IP address based timeout system (implemented in models/visitor\_model.php). Initially a timeout of one second between requests involving a captcha is in place. An error screen shows up if multiple requests from the same IP address for a captcha page are made within the time out period. Every mistaken entry of a captcha doubles this timeout period. The timeout period for an IP address is reset on a daily basis back to one second.

[Return to table of contents](#) .

## Upgrading Yioop

If you have an older version of Yioop that you would like to upgrade, make sure to back up your data. Be advised that the upgrade from a version 2 to version 3 or higher of Yioop is a reasonably major upgrade, your indexes should work after the upgrade, but version 2 classifiers do not work in Yioop version 3. Yioop stores data in two places: a database, the details of which can be found by looking at the Server Settings activity; and in a WORK\_DIRECTORY folder, the path to this folder can be found in the Configure activity, Search Engine Work Directory form. Once your back up is done, how you should upgrade depends on how you installed Yioop in the first place. We give three techniques below, the first of which will always work but might be slightly more complicated than the other two.

If you originally installed Yioop from the download page, download the latest version of Yioop. Extract the src subfolder of the zip file you get and replace the current src folder of the installation you have. If you had a file src/configs/LocalConfig.php, in your old installation make sure to copy this file to the upgrade. Make sure Yioop has write permissions on src/configs/Configs.php. Yioop should be then able to complete the upgrade process. If it doesn't look like your old work directory data is showing up, please check the Configure activity, Search Engine Work Directory form and make sure it has the same value as before the upgrade. In the case of upgrading a version 2 to a version 3 instance of Yioop, you should extract the whole Yioop folder you download and replace the old Yioop folder with this new one. Then to complete the upgrade, make sure src/configs/Configs.php is writable by your web server, go to the Configure activity, Search Engine Work Directory form and enter the value you had from before the upgrade.

If you have been getting Yioop directly from the git repository, then to upgrade Yioop you can just issue the command:

```
git pull
```

from a command prompt after you've switched into the Yioop directory.

If you are using Yioop as part of a Composer project, then issue the command:

```
composer update
```

[Return to table of contents](#) .

## Summary of Files and Folders

The Yioop search engine consists of three main scripts:

### **src/executables/Fetcher.php**

Used to download batches of urls provided the queue server.

### **src/executables/QueueServer.php**

Maintains a queue of urls that are going to be scheduled to be seen. It also keeps track of what has been seen and robots.txt info. Its last responsibility is to create the `index_archive` that is used by the search front end.

### **index.php**

Acts as the search engine web page. It is also used to handle message passing between the fetchers (multiple machines can act as fetchers) and the queue server.

The file `index.php` is used when you browse to an installation of a Yioop website. The description of how to use a Yioop web site is given in the sections starting from the The Yioop User Interface section. The files `Fetcher.php` and `QueueServer.php` are only connected with crawling the web. If one already has a stored crawl of the web, then you no longer need to run or use these programs. For instance, you might obtain a crawl of the web on your home machine and upload the crawl to a an instance of Yioop on the ISP hosting your website. This website could serve search results without making use of either `Fetcher.php` or `QueueServer.php`. To perform a web crawl you need to use both of these programs as well as the Yioop web site. This is explained in detail in the section on [Performing and Managing Crawls](#) .

The Yioop folder itself consists of several files and sub-folders. The file `index.php` as mentioned above is the main entry point into the Yioop web application. There is also an `.htaccess` file that is used to route most requests to the Yioop folder to go through this `index.php` file. If you look at the top folder structure of Yioop it has three main subfolders: **src** , **work\_directory** , and **tests** . If you are developing a [Composer project](#) with Yioop there might also be a **vendor** folder which contains a class autoloader for the PHP classes used in your project. The **src** folder contains the source code for Yioop. The **work\_directory** is the default location where Yioop stores indexes as well as where you can put any site specific customizations. As the location of where Yioop stores stuff is customizable, we will write `WORK_DIRECTORY` in caps to refer to location of this folder in the current instance of Yioop. Finally, **tests** is a folder of unit tests for various Yioop classes. We now describe the major files and folders in each of these folders in a little more detail.

The **src** contains several useful files as well as additional sub-folders. `yioopbar.xml` is the xml file specifying how to access Yioop as an Open Search Plugin. `favicon.ico` is used to display the little icon in the url bar of a browser when someone browses to the Yioop site. A URL to the file `bot.php` is given by the Yioop robot as it crawls websites so that website owners can find out information about who is crawling their sites. Here is a rough guide to what the `src` folder's various sub-folders contain:

### **configs**

This folder contains configuration files. You will probably not need to edit any of these files directly as you can set the most common configuration settings from with the admin panel of Yioop. The file `Config.php` controls a number of parameters about how data is stored, how, and how often, the queue server and fetchers communicate, and which file types are supported by Yioop. `ConfigureTool.php` is a command-line tool which can perform some of the configurations needed to get a Yioop installation running. It is only necessary in some virtual private server settings -- the preferred way to configure Yioop is through the web interface. `Createdb.php` can be used to create a bare instance of the Yioop database with a root admin user having no password. This script is not strictly necessary as the database should be creatable via the admin panel; however, it can be useful if the database isn't working for some reason. `createdb.php` includes the file `PublicHelpPages.php` from `WORK_DIRECTORY/app/configs` if present or from `BASE_DIR/configs` if not. This file contains the initial rows for the Public and Help group wikis.

WORK\_DIRECTORY/app/configs/PublicHelpPages.php. This can be done by running the file *ExportPublicHelpDb.php* which is in the configs folder. Also, in the configs folder is the file *default\_crawl.ini*. This file is copied to WORK\_DIRECTORY after you set this folder in the admin/configure panel. There it is renamed as *crawl.ini* and serves as the initial set of sites to crawl until you decide to change these. The file *TokenTool.php* is a tool which can be used to help in term extraction during crawls and for making trie's which can be used for word suggestions for a locale. To help word extraction this tool can generate in a locale folder (see below) a word bloom filter. This filter can be used to segment strings into words for languages such as Chinese that don't use spaces to separate words in sentences. For trie and segmenter filter construction, this tool can use a file that lists words one on a line.

### **controllers**

The controllers folder contains all the controller classes used by the web component of the Yioop search engine. Most requests coming into Yioop go through the top level *index.php* file. The query string (the component of the url after the *?*) then says who is responsible for handling the request. In this query string there is a part which reads *c= ...* This says which controller should be used. The controller uses the rest of the query string such as the *a=* variable for activity function to call and the *arg=* variable to determine which data must be retrieved from which models, and finally which view with what elements on it should be displayed back to the user. Within the controller folder is a sub-folder *components*, a component is a collection of activities which may be added to a controller so that it can handle a request.

### **css**

This folder contains the stylesheets used to control how web page tags should look for the Yioop site when rendered in a browser.

### **data**

This folder contains a default sqlite database for a new Yioop installation. Whenever the WORK\_DIRECTORY is changed it is this database which is initially copied into the WORK\_DIRECTORY to serve as the database of allowed users for the Yioop system.

### **examples**

This folder contains a file *SearchApi.php* whose code gives an example of how to use the Yioop search function api.

### **executables**

This folder is intended to hold command-line scripts and daemons which are used in conjunction with Yioop. In addition to the *Fetcher.php* and *QueueServer.php* script already mentioned, it contains: *ArcTool.php* , *ClassifierTool.php* , *ClassifierTrainer.php* , *CodeTool.php* , *Mirror.php* , *MediaUpdater.php* , and *QueryTool.php* . *ArcTool.php* can be used to examine the contents of *WebArchiveBundle*'s and *IndexArchiveBundle*'s from the command line. *ClassifierTool.php* is a command line tool for creating a classifier it can be used to perform some of the tasks that can also be done through the [Web Classifier Interface](#) . *ClassifierTrainer.php* is a daemon used in the finalization stage of building a classifier. *CodeTool.php* is for use by developers to maintain the Yioop code-base in various ways. *Mirror.php* can be used if you would like to create a mirror/copy of a Yioop installation. *MediaUpdater.php* can be used to do hourly updates of news feed search sources in Yioop. It also does video conversions of video files into web formats. Finally, *QueryTool.php* can be used to run queries from the command-line.

### **library**

This folder is short for library. It contains all the common classes for things like indexing, storing data to files, parsing urls, etc. *lib* contains eight subfolders: *archive\_bundle\_iterators* , *classifiers* , *compressors* , *index\_bundle\_iterators* , *indexing\_plugins* , *media\_jobs* , *processors* , *summarizers* . The *archive\_bundle\_iterators* folder has iterators for iterating over the objects of various kinds of web archive file formats, such as *arc*, *wiki-media*, etc. These iterators are used to iterate over such archives during a recrawl. The classifier folder contains code for training classifiers used by Yioop. The *compressors* folder contains classes that might be used to compress objects in a *web\_archive*. The *index\_bundle\_iterators* folder contains a variety of iterators useful for iterating over lists of documents which might be returned during a query to the search engine. The *media\_jobs* folder contains subclasses of *MediaJob*. These are jobs which are run on a periodic basis by the *MediaUpdater* class to do things such as update news

*processors* folder contains processors to extract page summaries for a variety of different mimetypes. The *summarizers* folder contains summarizers which could be used when processing a page to be indexed to produce a summary of what should be indexed out of that page.

### **locale**

This folder contains the default locale data which comes with the Yioop system. A locale encapsulates data associated with a language and region. A locale is specified by an [IETF language tag](#). So, for instance, within the locale folder there is a folder en-US for the locale consisting of English in the United States. Within a given locale tag folder there is a file `configure.ini` which contains translations of string ids to string in the language of the locale. This approach is the same idea as used in [Gettext](#) .po files. Yioop's approach does not require a compilation step nor a restart of the webserver for translations to appear. On the other hand, it is slower than the Gettext approach, but this could be easily mitigated using a memory cache such as [memcached](#) or [apc](#) . Besides the file `configure.ini`, there is a `statistics.txt` file which has info about what percentage of the id's have been translated. In addition to `configure.ini` and `statistics.txt`, the locale folder for a language contains a *resources* subfolder. The resources folder contains the files: *locale.js* , which contains locale specific Javascript code such as the variable `alpha` which is used to list out the letters in the alphabet for the language in question for spell check purposes, and `roman_array` for mapping between roman alphabet and the character system of the locale in question; *suggest-trie.txt.gz* , a Trie data structure used for search bar word suggestions; and *Tokenizer.php* , which can specify the number of characters for this language to constitute a char gram, might contain `segmenter` to split strings into words for this language, a stemmer class used to stem terms for this language, a stopword remover for the centroid summarizer, a part of speech tagger, or thesaurus lookup procedure for the locale.

### **models**

This folder contains the subclasses of Model used by Yioop Models are used to encapsulate access to secondary storage. i.e., Accesses to databases or the filesystem. They are responsible for marshalling/de-marshalling objects that might be stored in more than one table or across several files. The models folder has within it a *datasources* folder. A datasource is an abstraction layer for the particular filesystem and database system that is being used by a Yioop installation. At present, datasources have been defined for PDO (PHP's generic DBMS interface), `sqlite3`, and `mysql` databases.

### **resources**

Used to store binary resources such as graphics, video, or audio. For now, just stores the Yioop logo.

### **scripts**

This folder contains the Javascript files used by Yioop.

### **views**

This folder contains View subclasses as well as folders for elements, helpers, and layouts. A View is responsible for taking data given to it by a controller and formatting it in a suitable way. Most views output a web page; however, some of the views responsible for communication between the fetchers and the queue server output serialized objects. The elements folder contains Element classes which are typically used to output portions of web pages. For example, the `html` that allows one to choose an Activity in the Admin portion of the website is rendered by an `ActivityElement`. The helpers folder contains Helper subclasses. A Helper is used to automate the task of outputting certain kinds of web tags. For instance, the `OptionsHelper` when given an array can be used to output select tags and option tags using data from the array. The layout folder contains Layout subclasses. A Layout encapsulates the header and footer information for the kind of a document a View lives on. For example, web pages on the Yioop site all use the `WebLayout` class as their Layout. The `WebLayout` class has a `render` method for outputting the doctype, open html tag, head of the document including links for style sheets, etc. This method then calls the render methods of the current View, and finally outputs scripts and the necessary closing document tags.

The second top level subfolder of a Yioop instance is the **tests** folder. This folder contains UnitTest's

UnitTest and JavascriptUnitTest classes which defined in the src/library/Unit\_Test.php and src/library/JavascriptUnitTest.php. It also contains a few files used for experiments. For example, StringCatExperiment.php was used to test which was the faster way to do string concatenation in PHP. ManyUserExperiment.php can be used to create a test Yioop installation with many users, roles, and groups. Some unit testing of the wiki Help system makes use of [PhantomJS](#). If PhantomJS is not configured, these tests will be skipped. To configure PhantomJS you simply add a define for your path to PhantomJS to your src/configs/LocalConfig.php file. For example, one might have add the define: `define("PHANTOM_JS", "/usr/local/bin/phantomjs");`

The last top level folder that Yioop makes use of is the WORK DIRECTORY. The location of this directory is set during the configuration of a Yioop installation, but defaults to your yioop folder's work\_directory sub-folder. Yioop stores crawls, and other data local to a particular Yioop installation in files and folders in this directory. In the event that you upgrade your Yioop installation you should only need to replace the Yioop src folder and in the configuration process of Yioop tell it where your WORK DIRECTORY is. Of course, it is always recommended to back up one's data before performing an upgrade. Within the WORK DIRECTORY, Yioop stores four main files: Profile.php, crawl.ini, bot.txt, and robot\_table.txt. Here is a rough guide to what the WORK DIRECTORY's sub-folder contain:

#### **app**

This folder is used to contain your overrides to the views, controllers, models, resources, locale etc. For example, if you wanted to change how the search results were rendered, you could add a views/SearchView.php file to the app folder and Yioop would use it rather than the one in the Yioop base directory's views folder. Using the app dir makes it easier to have customizations that won't get messed up when you upgrade Yioop.

#### **cache**

The directory is used to store folders of the form ArchiveUNIX\_TIMESTAMP, IndexDataUNIX\_TIMESTAMP, and QueueBundleUNIX\_TIMESTAMP. ArchiveUNIX\_TIMESTAMP folders hold complete caches of web pages that have been crawled. These folders will appear on machines which are running fetcher.php. IndexDataUNIX\_TIMESTAMP folders hold a word document index as well as summaries of pages crawled. A folder of this type is needed by the web app portion of Yioop to serve search results. These folders can be moved from machine to whichever machine you want to server results from. QueueBundleUNIX\_TIMESTAMP folders are used to maintain the priority queue during the crawling process. The QueueServer.php program is responsible for creating both IndexDataUNIX\_TIMESTAMP and QueueBundleUNIX\_TIMESTAMP folders.

#### **data**

If an sqlite or sqlite3 (rather than say MySQL) database is being used then a default.db file is stored in the data folder. In Yioop, the database is used to manage users, roles, locales, and crawls. Data for crawls themselves are NOT stored in the database. Suggest a url data is stored in data in the file suggest\_url.txt, certain cron information about machines is saved in cron\_time.txt, and plugin configuration information can also be stored in this folder.

#### **locale**

This is generally a copy of the locale folder mentioned earlier. In fact, it is the version that Yioop will try to use first. It contains any customizations that have been done to locale for this instance of Yioop. If you using a version of Yioop after Yioop 2.0, this folder have been moved to app/locale.

#### **log**

When the fetcher and queue server are run as daemon processes log messages are written to log files in this folder. Log rotation is also done. These log files can be opened in a text editor or console app.

#### **query**

This folder is used to stored caches of already performed queries when file caching is being used.

#### **schedules**

This folder has four kinds of subfolders: media\_convert, IndexDataUNIX\_TIMESTAMP,

the `media_convert` folder. It is used by `MediaUpdater.php` to store job information about video files that need to be converted. For the other folder, when a fetcher communicates with the web app to say what it has just crawled, the web app writes data into these folders to be processed later by the queue server. The `UNIX_TIMESTAMP` is used to keep track of which crawl the data is destined for. `IndexData` folders contain mini-inverted indexes (word document records) which are to be added to the global inverted index (called the dictionary) for that crawl. `RobotData` folders contain information that came from `robots.txt` files. Finally, `ScheduleData` folders have data about found urls that could eventually be scheduled to crawl. Within each of these three kinds of folders there are typically many sub-folders, one for each day data arrived, and within these subfolders there are files containing the respective kinds of data.

### search\_filters

This folder is used to store text files containing global after crawl search filter and summary data. The global search filter allows a user to specify after a crawl is done that certain urls be removed from the search results. The global summary data can be used to edit the summaries for a small number of web pages whose summaries seem inaccurate or inappropriate. For example, some sites like Facebook only allow big search engines like Google to crawl them. Still there are many links to Facebook, so Facebook on an open web crawl will appear, but with a somewhat confused summary based only on link text; the results editor allows one to give a meaningful summary for Facebook.

### temp

This is used for storing temporary files that Yioop creates during the crawl process. For example, temporary files used while making thumbnails. Each fetcher has its own temp folder, so you might also see folders `0-temp`, `1-temp`, etc.

[Return to table of contents](#) .

## Search and User Interface

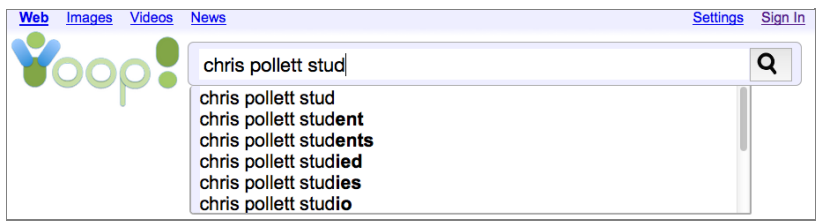
At this point one hopefully has installed Yioop. If you used one of the [install guides](#) , you may also have performed a simple crawl. We are now going to describe some of the basic search features of Yioop as well as the Yioop administration interface. We will describe how to perform crawls with Yioop in more detail in the [Crawling and Customizing Results](#) chapter. If you do not have a crawl available, you can test some of these features on the [Yioop Demo Site](#).

### Search Basics

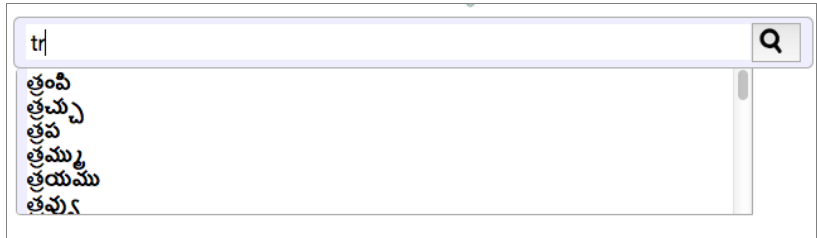
The main search form for Yioop looks like:



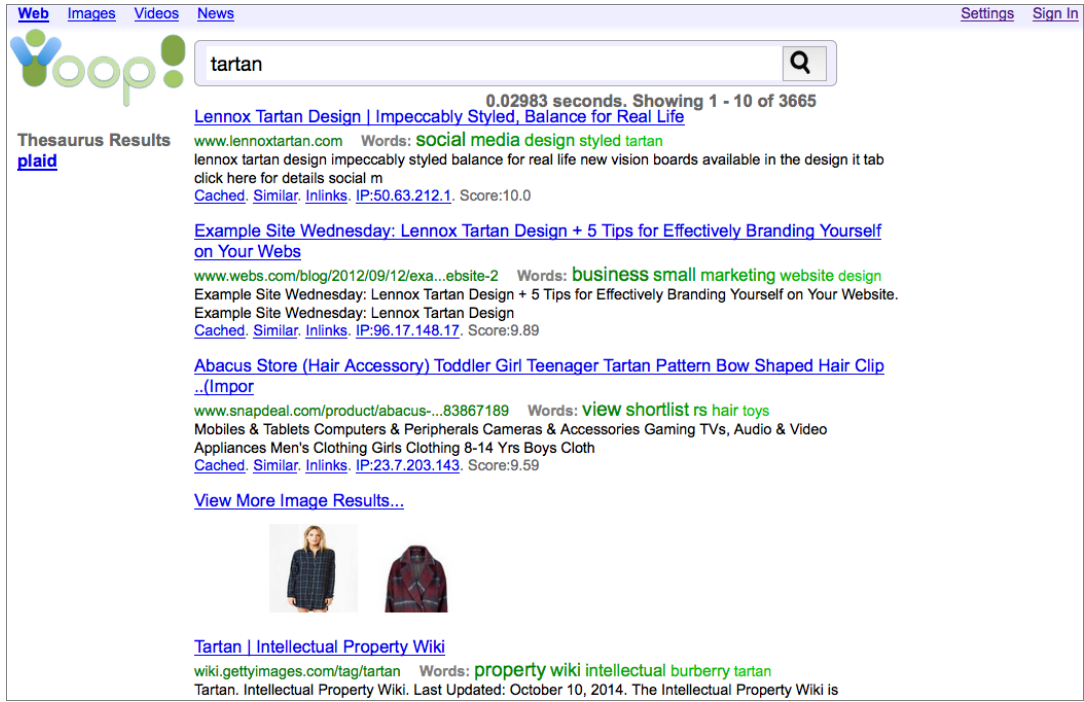
The HTML for this form is in `views/search_views.php` and the icon is stored in `resources/yioop.png`. You may want to modify these to incorporate Yioop search into your site. For more general ways to modify the look of this pages, consult the [Building a site using Yioop](#) documentation. The Yioop logo on any screen in the Yioop interface is clickable and returns the user to the main search screen. One performs a search by typing a query into the search form field and clicking the Search button. As one is typing, Yioop suggests possible queries, you can click, or use the up down arrows to select one of



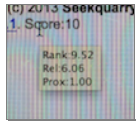
For some non-roman alphabet scripts such as Telugu you can enter words using how they sound using roman letters and get suggestions in the script in question:

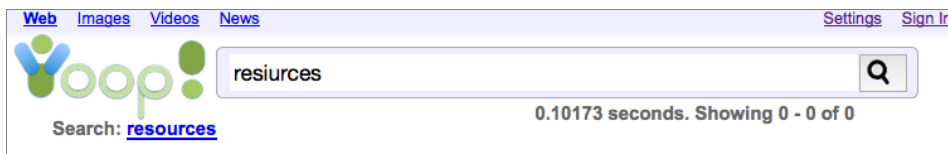


The [More Statistics] link only shows if under the Admin control panel you clicked on more statistics for the crawl. This link goes to a page showing many global statistics about the web crawl. Beneath this link are the Blog and Privacy links (as well as a link back to the SeekQuarry site). These two links are to static pages which can be customized through the Manage Locale activity. Typical search results might look like:

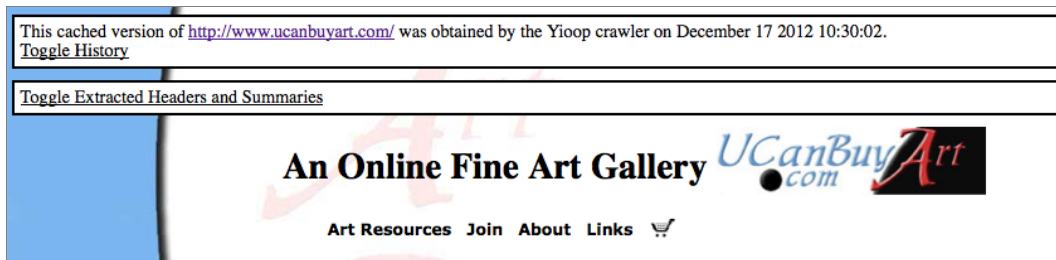


Thesaurus results might appear to one side and suggest alternative queries based on a thesaurus look up (for English, this is based on Wordnet). The terms next Words: are a word cloud of important terms in the document. These are created if the indexer user the centroid summarizer. Hovering over the Score of a search result reveals its component scores. These might include: Rank, Relevance, Proximity, as well as any Use to Rank Classifier scores and Word Net scores (if installed).

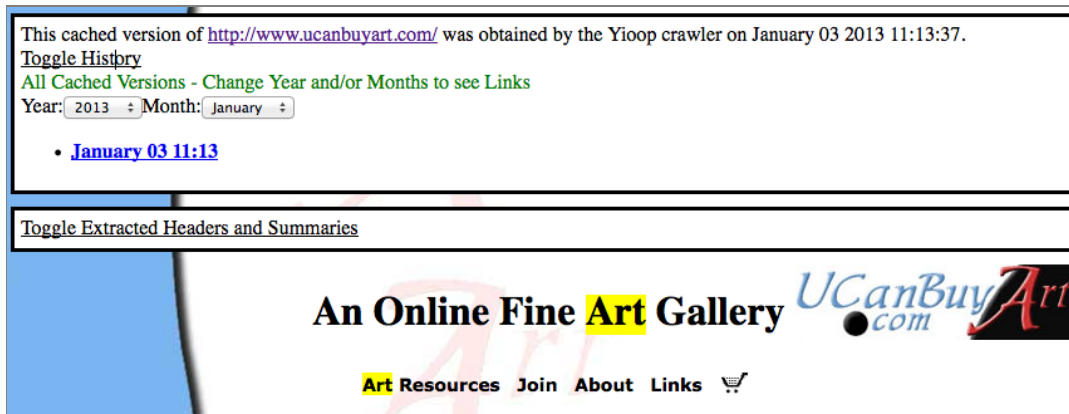




Each result back from the query consists of several parts: First comes a title, which is a link to the page that matches the query term. This is followed by a brief summary of that page with the query words in bold. Then the document rank, relevancy, proximity, and overall scores are listed. Each of these numbers is a grouped statistic -- several "micro index entry" are grouped together/summed to create each. So even though a given "micro index entry" might have a document rank between 1 and 10 there sum could be a larger value. Further, the overall score is a generalized inner product of the scores of the "micro index entries", so the separated scores will not typically sum to the overall score. After these scores there are three links: Cached, Similar, and Inlinks. Clicking on Cached will display Yioop's downloaded copy of the page in question. We will describe this in more detail in a moment. Clicking on Similar causes Yioop to locate the five words with the highest relevancy scores for that document and then to perform a search on those words. Clicking on Inlinks will take you to a page consisting of all the links that Yioop found to the document in question. Finally, clicking on an IP address link returns all documents that were crawled from that IP address.



As the above illustrates, on a cache link click, Yioop will display a cached version of the page. The cached version has a link to the original version and download time at the top. Next there is a link to display all caches of this page that Yioop has in any index. This is followed by a link for extracted summaries, then in the body of the cached document the query terms are highlighted. Links within the body of a cache document first target a cached version of the page that is linked to which is as near into the future of the current cached page as possible. If Yioop doesn't have a cache for a link target then it goes to location pointed to by that target. Clicking on the history toggle, produces the following interface:



This lets you select different caches of the page in question.

Clicking the "Toggle extracted summary" link will show the title, summary, and links that were extracted from the full page and indexed. No other terms on the page are used to locate the page via a search query. This can be viewed as an "SEO" view of the page.



Toggle Extracted Headers and Summaries

```

* About to connect() to 174.120.10.130 port 80 (#0)
*   Trying 174.120.10.130...
* Connected to 174.120.10.130 (174.120.10.130) port 80 (#0)
* Connected to 174.120.10.130 (174.120.10.130) port 80 (#0)
> GET / HTTP/1.1
Range: bytes=0-10000
User-Agent: Mozilla/5.0 (compatible; YioopBot; http://localhost/git/yioop/bot.php)
Accept: */*
Accept-Encoding: deflate, gzip
Host: www.ucanbuyart.com

HTTP/1.1 204 Partial Content
Date: Thu, 03 Jan 2013 19:11:37 GMT
Server: Apache
Cache-Control: max-age=7200, must-revalidate
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Range: bytes 0-6012/6013
Content-Length: 6013
Content-Type: text/html
X-Pad: avoid browser bug
Extracted Title

An Online Fine Art Gallery U Can Buy Art - Buy Fine Art Online

Extracted Description

... UCanBuyArt.com is an online art gallery and dealer designed for sales of high quality and original art from renowned artists. Art categories include oil paintings, water colors, ceramics, artist bears, fantasy art, metal art, native art, home decor, and garden art. ... An Online Fine Art Gallery U Can Buy Art is a fine art dealer & gallery for famous artists original oil paintings, fine art prints, watercolors, sculptures, photography, drawings, and metal art. UCanBuyArt.com is designed with the philosophy that the fine arts and artists should be supported on a global medium and made accessible to all. With this in mind we have created an online community for artists to exhibit and sell their work directly to the public. Prices are in US Dollars and art purchases may be made directly from this secure online art gallery using VISA or MasterCard. Visit us often for the latest updates. Items range in price from $5 to $245,000. See our fine art services for fine art appraisals, professional art restoration, upcoming art shows, and much more! For a feeling of Christmas year round see our Christmas art that perfect pieces of fine art. Search our fine art gallery by one of the convenient options below. - Art Category -Artist Name -Gallery Advanced Search -Art Classifieds What's New! Site Index Contact Us Link to Us Receive our newsletter! Online art gallery .. Buy that perfect piece of fine art. Search our fine art gallery by one of the convenient options below. - Art Category -Artist Name -Gallery Advanced Search -Art Classifieds What's New! Site Index Contact Us Link to Us Receive our newsletter! Online art gallery .. Buy that perfect piece of fine art. Search our fine art gallery by one of the convenient options below.

Extracted Links

Array
(
    [http://www.ucanbuyart.com/default.html] =>
    [http://www.ucanbuyart.com/artresources.html] => Art Resources
    [http://www.ucanbuyart.com/join.html] => Join
    [http://www.ucanbuyart.com/about.html] => About
    [http://www.ucanbuyart.com/links.html] => Links
    [http://www.ucanbuyart.com/advancedsearch.html] => Advanced Search
    [http://www.ucanbuyart.com/classifieds.html] => Art Classifieds
    [http://www.ucanbuyart.com/new/] => What's New!
    [http://www.ucanbuyart.com/index.html] => Site Index
    [http://www.ucanbuyart.com/contact.html] => Contact Us
    [http://www.ucanbuyart.com/links.html] => Link to Us
    [http://www.ucanbuyart.com/newsletter.html] => Receive our newsletter!
    [http://www.ucanbuyart.com/all_paintings.html] => oil paintings
    [http://www.ucanbuyart.com/art_prints.html] => fine art prints
    [http://www.ucanbuyart.com/water_colors.html] => watercolors
    [http://www.ucanbuyart.com/sculptures.html] => sculptures
    [http://www.ucanbuyart.com/art_photography.html] => photography
    [http://www.ucanbuyart.com/art_drawings.html] => drawings
    [http://www.ucanbuyart.com/fine_art_services.html] => Fine Art Services
    [http://www.ucanbuyart.com/art_search.html] => fine art appraisals
    [http://www.ucanbuyart.com/restoration.html] => professional art restoration
    [http://www.ucanbuyart.com/events.html] => upcoming art shows
    [http://www.ucanbuyart.com/christmas-boutique.html] => Christmas Art Boutique
    [ ] => sales@UCanBuyArt.com
    [http://www.ucanbuyart.com/Custom_Art.html] => View artist's who offer custom art
    [http://www.ucanbuyart.com/fine_art.html] => Complete art gallery tour (alphabetical)
    [http://www.ucanbuyart.com/fine_art.html] => Complete art gallery tour (by price)
    [http://www.ucanbuyart.com/fine_art/logo2.jpg] => online fine art gallery u can buy art
    [http://www.ucanbuyart.com/images/visa_6x8x3.gif] => Visa
    [http://www.ucanbuyart.com/images/mc_6x8x3.gif] => MasterCard
)

```

An Online Fine Art Gallery **UCanBuyArt.com**

Art Resources Join About Links


It should be noted that cached copies of web pages are stored on the fetcher which originally downloaded the page. The IndexArchive associated with a crawl is stored on the queue server and can be moved around to any location by simply moving the folder. However, if an archive is moved off the network on which fetcher lives, then the look up of a cached page might fail.

In addition, to a straightforward web search, one can also do image, video, news searches by clicking on the Images, Video, or News links in the top bar of Yioop search pages. Below are some examples of what these look like for a search on "Obama":

Web Images Videos News Settings Sign In

**Yioop** obama


0.03609 seconds. Showing 1 - 50 of 40846




0.03427 seconds. Showing 1 - 10 of 3787

[Obama Claims He's Visited 57 States - YouTube](#)  
 [www.youtube.com/watch?v=EpGH02Dtlws](http://www.youtube.com/watch?v=EpGH02Dtlws) Words: **play views obama loading sign**  
 Obama Claims He's Visited 57 States - YouTube. . Upload. Sign in. Search. . Loading...  
 This video is unavailable. Watch Queue. TV Queue. Watch Que

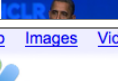
[Cached. Similar. Inlinks. IP:74.125.239.130. Score:8.74](#)

[The Speeches of President Obama - President Obama at White House Correspondents Dinner - Video](#)  
 [www.metacafe.com/watch/hl-501450...s\\_dinner](http://www.metacafe.com/watch/hl-501450...s_dinner) Words: **president obama speeches hulu views**  
 The Speeches of President Obama - President Obama at White House Correspondents Dinner - Video. The Speeches of President Obama - President Obama at W

[Cached. Similar. Inlinks. IP:54.164.159.109. Score:8.27](#)


[The Speeches of President Obama - President Obama: Memorial in Arizona - Video](#)  
 [www.metacafe.com/watch/hl-501138...\\_arizona](http://www.metacafe.com/watch/hl-501138..._arizona) Words: **hl itempageactions president itemid sphinx**  
 Metacafe Popular Movies Games Web Originals Tech Viral Music More ... Safe Search  
 OFF Go Sign In The Speeches of President Obama - President Obama: Me

[Cached. Similar. Inlinks. IP:54.164.159.109. Score:8.21](#)

[Obama: Neither Party "blameless" in Debt Crisis - Video](#)  
 [www.metacafe.com/watch/cb-LJee4y...t\\_crisis](http://www.metacafe.com/watch/cb-LJee4y...t_crisis) Words: **views cbs obama debt ceiling**


---

[Web](#) [Images](#) [Videos](#) [News](#) [Settings](#) [Sign In](#)



0.03339 seconds. Showing 1 - 20 of 52

[Trump: Deport children of immigrants living illegally in US.](#) Associated Press Top Headlines -  
 1 hour ago  
[hosted2.ap.org/APDEFAULT/3d281c11a96b4ad082fe88aa0db04305/Article\\_2015-08-16-US-...4aec9a48e54cb61e26a0](http://hosted2.ap.org/APDEFAULT/3d281c11a96b4ad082fe88aa0db04305/Article_2015-08-16-US-...4aec9a48e54cb61e26a0)  
 WASHINGTON (AP) — Republican presidential candidate Donald Trump wants more than a wall to keep out immigrants living in the country illegally. He als

 [How Obama can do Iran nuclear deal even if Congress disapproves.](#) Yahoo  
 News - 13 hours ago  
[news.yahoo.com/obama-iran-nuclear-deal-even-congress-disapproves-115227371.html](http://news.yahoo.com/obama-iran-nuclear-deal-even-congress-disapproves-115227371.html)  
 WASHINGTON (AP) — The September vote on the Iran nuclear deal is billed as a titanic standoff between President Barack Obama and Congress. Yet even if

When Yioop crawls a page it adds one of the following meta words to the page media:text, media:image, or media:video. RSS (or Atom) feed sources that have been added to Media Sources under the [Search Sources](#) activity are downloaded from each hour. Each RSS item on such a downloaded pages has the meta word media:news added to it. A usual web search just takes the search terms provided to perform a search. An Images, Video, News search tacks on to the search terms, media:image or media:video, or media:news. Detection of images is done via mimetype at initial page download time. At this time a thumbnail is generated. When search results are presented it is this cached thumbnail that is shown. So image search does not leak information to third party sites. On any search results page with images, Yioop tries to group the images into a thumbnail strip. This is true of both normal and images search result pages. In the case of image search result pages, except for not-yet-downloaded pages, this results in almost all of the results being the thumbnail strip. Video page detection is not done through mimetype as popular sites like YouTube, Vimeo, and others vary in how they use Flash or video tags to embed video on a web page. Yioop uses the Video Media sources that have been added in the Search Sources activity to detect whether a link is in the format of a video page. To get a thumbnail for the video it again uses the method for rewriting the video url to an image link specified for the particular site in question in Search Sources. i.e., the thumbnail will be downloaded from the original site. **This could leak information to third party sites about your search.**

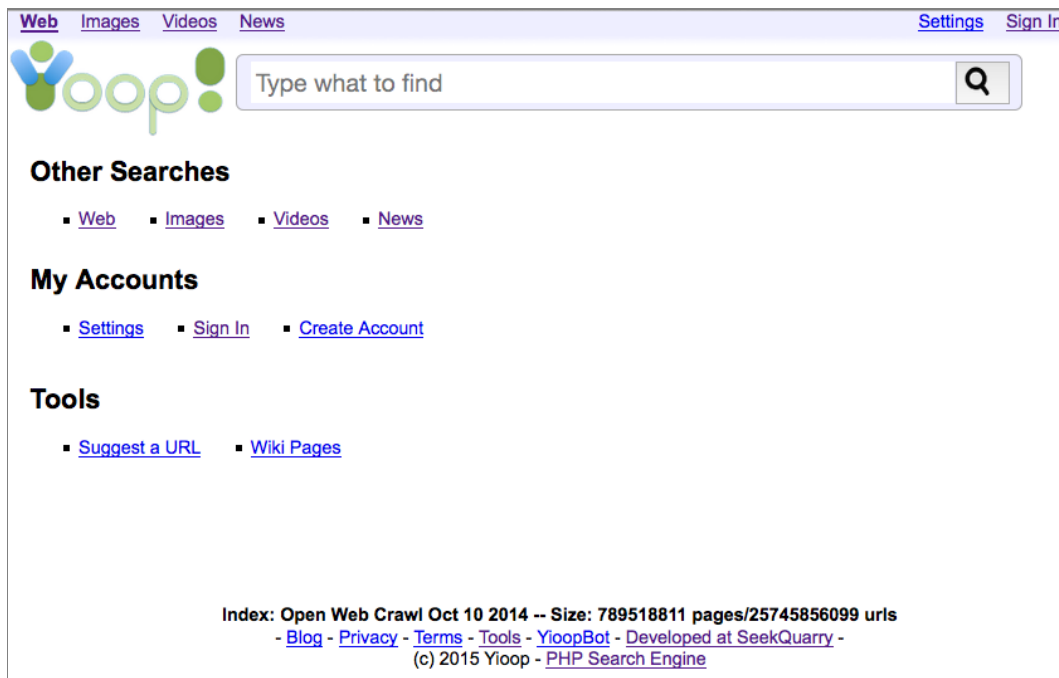
The format of News search results is somewhat different from usual search results. News search results can appear during a normal web search, in which case they will appear clustered together, with a leading link "News results for ...". No snippets will be shown for these links, but the original media source for the link will be displayed and the time at which the item first appeared will be

description of the new item is displayed. In both settings, it is possible to click on the media source name next to the news item link. This will take one to a page of search results listing all articles from that media source. For instance, if one were to click on the Yahoo News text above one would go to results for all Yahoo News articles. This is equivalent to doing a search on: `media:news:Yahoo+News` . If one clicks on the News subsearch, not having specified a query yet, then all stored news items in the current language will be displayed, roughly ranked by recentness. If one has RSS media sources which are set to be from different locales, then this will be taken into account on this blank query News page.

[Return to table of contents](#) .

## Search Tools Page

As one can see from the image of the main search form shown previously, the footer of each search and search result page has several links. Blog takes one to the group feed of the built in PUBLIC group which is editable from the root account, Privacy takes one to the Yioop installations privacy policy, and Terms takes one to the Yioop installations terms of service. The YioopBot link takes one to a page describing the installation's web crawler. These static pages are all Wiki pages of the PUBLIC group and can be edited by the root account. The Tools link takes one to the following page:



[Web](#) [Images](#) [Videos](#) [News](#) [Settings](#) [Sign In](#)

**Yioop!** Type what to find

**Other Searches**

- [Web](#)
- [Images](#)
- [Videos](#)
- [News](#)

**My Accounts**

- [Settings](#)
- [Sign In](#)
- [Create Account](#)

**Tools**

- [Suggest a URL](#)
- [Wiki Pages](#)

Index: Open Web Crawl Oct 10 2014 -- Size: 789518811 pages/25745856099 uris  
- [Blog](#) - [Privacy](#) - [Terms](#) - [Tools](#) - [YioopBot](#) - [Developed at SeekQuarry](#) -  
(c) 2015 Yioop - [PHP Search Engine](#)

Beneath the Other Search Sources section is a complete listing of all the search sources that were created using [Search Sources](#) . This might be more than just the Images, Video, and News that come by default with Yioop. The My Account section of this page gives another set of links for signing into, modifying the settings of, and creating account. The Other Tools section has a link to the form below where user's can suggest links for the current or future crawls.



Suggest a site for the next web crawl. Up to ten sites per day can be accepted.

URL:

Human Check:

Human Check:

[Return to Yoop](#)

This link only appears if under Server Settings, Account Registration is not set to Disable registration. The Wiki Pages link under Other Tools takes one to a searchable list of all Wiki pages of the default PUBLIC group.

[Return to table of contents](#) .

## Search Operators

Turning now to the topic of how to enter a query in Yoop: A basic query to the Yoop search form is typically a sequence of words separated by whitespace. This will cause Yoop to compute a "conjunctive query", it will look up only those documents which contain all of the terms listed. Yoop also supports a variety of other search box commands and query types:

- **#num#** in a query are treated as query presentation markers. When a query is first parsed, it is split into columns based with **#num#** as the column boundary. For example, bob #2# bob sally #3# sally #1#. A given column is used to present **#num#** results, where **#num#** is what is between the hash marks immediately after it. So in the query above, the subquery *bob* is used for the first two search results, then the subquery *bob sally* is used for the next three results, finally the last column is always used for any remaining results. In this case, the subquery *sally* would be used for all remaining results even though its **#num#** is 1. If a query does not have any **#num#**'s it is assumed that it has only one column.
- Separating query terms with a vertical bar | results in a disjunctive query. These are parsed for after the presentation markers above. So a search on: *Chris | Pollett* would return pages that have either the word *Chris* or the word *Pollett* or both.
- Putting the query in quotes, for example "Chris Pollett", will cause Yoop to perform an exact match search. Yoop in this case would only return documents that have the string "Chris Pollett" rather than just the words Chris and Pollett possibly not next to each other in the document. Also, using the quote syntax, you can perform searches such as "Chris \* Homepage" which would return documents which have the word Chris followed by some text followed by the word Homepage.
- If the query has at least one word not prefixed by -, then adding a '-' in front of a word in a query means search for results not containing that term. So a search on: of *-the* would return results containing the word "of" but not containing the word "the".
- Searches of the forms: **related:url** , **cache:url** , **link:url** , **ip:ip\_address** are equivalent to having clicked on the Similar, Cached, InLinks, IP address links, respectively, on a summary with that url and ip address.

The remaining query types we list in alphabetical order:

### **code:http\_error\_code**

returns the summaries of all documents downloaded with that HTTP response code. For example, code:04 would return all summaries where the response was a Page Not Found error.

returns summaries of all documents crawled on the given date. For example, *date:2011-01* returns all document crawled in January, 2011. As one can see detail goes down to the second level, so one can have an idea about how frequently the crawler is hitting a given site at a given time.

**dns:num\_seconds**

returns summaries of all documents whose DNS lookup time was between num\_seconds and num\_seconds + 0.5 seconds. For example, *dns:0.5*.

**filetype:extension**

returns summaries of all documents found with the given extension. So a search: *Chris Pollett filetype:pdf* would return all documents containing the words Chris and Pollett and with extension pdf.

**host:all**

returns summaries of all domain level pages (pages where the path was /).

**index:timestamp or i:timestamp**

causes the search to make use of the IndexArchive with the given timestamp. So a search like: *Chris Pollett i:1283121141 | Chris Pollett* take results from the index with timestamp 1283121141 for Chris Pollett and unions them with results for Chris Pollett in the default index

**if:keyword!add\_keywords\_on\_true!add\_keywords\_on\_false**

checks the current conjunctive query clause for "keyword"; if present, it adds "add\_keywords\_on\_true" to the clause, else it adds the keywords "add\_keywords\_on\_true". This meta word is typically used as part of a crawl mix. The else condition does not need to be present. As an example, *if:oracle!info:http://oracle.com!/site:none* might be added to a crawl mix so that if a query had the keyword oracle then the site *http://oracle.com/* would be returned by the given query clause. As part of a larger crawl mix this could be used to make oracle's homepage appear at the top of the query results. If you would like to inject multiple keywords then separate the keywords using plus rather than white space. For example, *if:corvette!fast+car*.

**info:url**

returns the summary in the Yioop index for the given url only. For example, one could type *info:http://www.yahoo.com/* or *info:www.yahoo.com* to get the summary for just the main Yahoo! page. This is useful for checking if a particular page is in the index.

**lang:IETF\_language\_tag**

returns summaries of all documents whose language can be determined to match the given language tag. For example, *lang:en-US* .

**media:kind**

returns summaries of all documents found of the given media kind. Currently, the text, image, news, and video are the four supported media kinds. So one can add to the search terms *media:image* to get only image results matching the query keywords.

**mix:name or m:name** : tells Yioop to use the crawl mix "name" when computing the results of the query. The section on mixing crawl indexes has more details about crawl mixes. If the name of the original mix had spaces, for example, cool mix then to use the mix you would need to replace the spaces with plusses, *m:cool+mix* .

**modified:Y, modified:Y-M, modified:Y-M-D**

returns summaries of all documents which were last modified on the given date. For example, *modified:2010-02* returns all document which were last modified in February, 2010.

**no:some\_command** is used to tell Yioop not to perform some default transformation of the search terms. For example, *no*

*guess* tells Yioop not to try to guess the semantics of the search before doing the search. This would mean for instance, that Yioop would not rewrite the query *yahoo.com* into *site:yahoo.com*. *no:network* tells Yioop to only return search results from the current machine and not to send the query to all machines in the Yioop instance. *no:cache* says to recompute the query and not to make use of memcache or file cache.

**numlinks:some\_number**

returns summaries of all documents which had some\_number of outgoing links. For example,

**os:operating\_system**

returns summaries of all documents served on servers using the given operating system. For example, *os:centos* , make sure to use lowercase.

**path:path\_component\_of\_url**

returns summaries of all documents whose path component begins with *path\_component\_of\_url*. For example, *path:/phpBB* would return all documents whose path started with phpBB, *path:/robots.txt* would return summaries for all robots.txt files.

**raw:number**

control whether or not Yioop tries to do deduplication on results and whether links and pages for the same url should be grouped. Any number greater than zero says don't do deduplication.

**robot:user\_agent\_name**

returns robots.txt pages that contained that *user\_agent\_name* (after lower casing). For example, *robot:yioopbot* would return all robots.txt pages explicitly having a rule for YioopBot.

**safe:boolean\_value**

is used to provide "safe" or "unsafe" search results. Yioop has a crude, "hand-tuned", linear classifier for whether a site contains pornographic content. If one adds *safe:true* to a search, only those pages found which were deemed non-pornographic will be returned. Adding *safe:false* has the opposite effect.

**server:web\_server\_name**

returns summaries of all documents served on that kind of web server. For example, *server:apache* .

**site:url, site:host, or site:domain**

returns all of the summaries of pages found at that url, host, or domain. As an example, *site:http://prints.ucanbuyart.com/lithograph\_art.html* , *site:http://prints.ucanbuyart.com/* , *site:prints.ucanbuyart.com* , *site:.ucanbuyart.com* , *site:ucanbuyart.com*, *site:com*, will all returns with decreasing specificity. To return all pages and links to pages in the Yioop index, you can do *site:any* . To return all pages (as opposed to pages and links to pages) listed in a Yioop index you can do *site:all* . *site:all* doesn't return any links, so you can't group links to urls and pages of that url together. If you want all sites where one has a page in the index as well as links to that site, than you can do *site:doc* .

**size:num\_bytes**

returns summaries of all documents whose download size was between *num\_bytes* and *num\_bytes* + 5000. *num\_bytes* must be a multiple of 5000. For example, *size:15000* .

**time:num\_seconds**

returns summaries of all documents whose download time excluding DNS lookup time was between *num\_seconds* and *num\_seconds* + 0.5 seconds. For example, *time:1.5* .

**version:version\_number**

returns summaries of all documents served on web servers with the given version number. For example, one might have a query *server:apache version:2.2.9* .

**weight:some\_number or w:some\_number**

has the effect of multiplying all score for this portion of a query by *some\_number*. For example, *Chris Pollett | Chris Pollett site:wikipedia.org w:5* would multiply scores satisfying *Chris Pollett* and on *wikipedia.org* by 5 and union these with those satisfying *Chris Pollett* .

Although we didn't say it next to each query form above, if it makes sense, there is usually an *all* variant to a form. For example, *os:all* returns all documents from servers for which os information appeared in the headers.

**Result Formats**

In addition to using the search form interface to query Yioop, it is also possible to query Yioop and get results in Open Search RSS format. To do that you can either directly type a URL into your browser of the form:

```
http://my-yioop-instance-host/?f=rss&q=query+terms
```

Or you can write AJAX code that makes requests of URLs in this format. Although, there is no official Open Search JSON format, one can get a JSON object with the same structure as the RSS search

`http://my-yioop-instance-host/?f=json&q=query+terms`

Sometimes it is advantageous to get the json data back as an argument to a Javascript function -- so called, [JSONP Format](#). This can be accomplished by add a callback parameter to your json query:

`http://my-yioop-instance-host/?f=json&callback=myJavascriptFunction&q=query+terms`

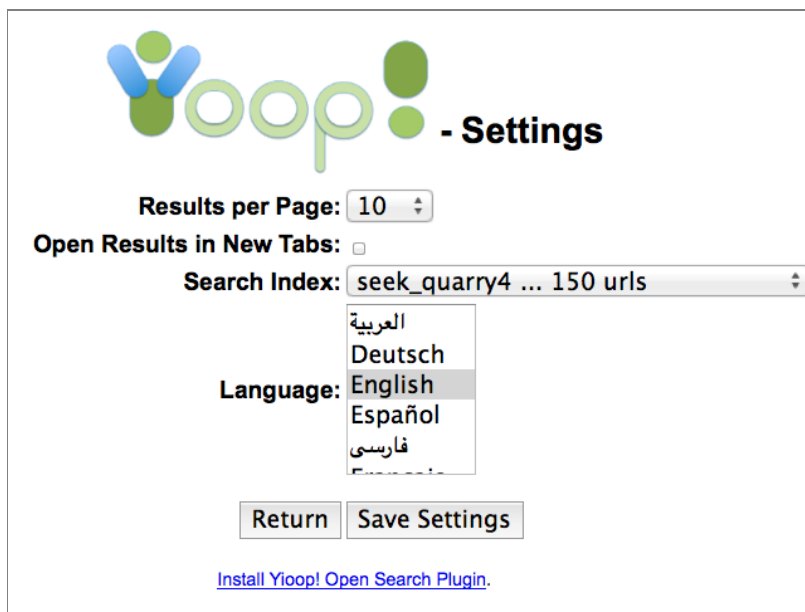
[Return to table of contents](#) .

## Settings

In the corner of the page with the main search form is a Settings-Signin element:

[Settings](#) [Sign In](#)

This element provides access for a user to change their search settings by clicking Settings. The Sign In link provides access to the Admin and User Accounts panels for the website. Clicking the Sign In link also takes one to a page where one can register for an account if Yioop is set up to allow user registration.



On the Settings page, there are currently three items which can be adjusted: The number of results per page when doing a search, the language Yioop should use, and the particular search index Yioop should use. When a user clicks save, the data is stored by Yioop. The user can then click "Return to Yioop" to go back the search page. Thereafter, interaction with Yioop will make use of any settings' changes. Data is stored in Yioop and associated with a given user via a cookies mechanism. In order for this to work, the user's browser must allow cookies to be set. This is usually the default for most browsers; however, it can sometimes be disabled in which case the browser option must be changed back to the default for Settings to work correctly. It is possible to control some of these settings by tacking on stuff to the URL. For instance, adding `&l=fr-FR` to the URL query string (the portion of the URL after the question mark) would tell Yioop to use the French from France for outputting text. You can also add `&its=` the Unix timestamp of the search index you want.

[Return to table of contents](#) .

## Mobile Interface

Yioop's user interface is designed to display reasonably well on tablet devices such as the iPad. For smart phones, such as iPhone, Android, Blackberry, or Windows Phone, Yioop has a separate user interface. For search, settings, and login, this looks fairly similar to the non-mobile user interface:

The screenshot displays three parts of the Yioop interface. On the left is the main page with the Yioop logo, a search bar, and site statistics. In the center is the 'Settings' page, which includes a 'Results/Page' dropdown set to 10, an 'Open in Tabs' checkbox, a 'Search Index' dropdown, and a 'Language' dropdown menu with options like العربية, Deutsch, English, Español, and فارسی. On the right is the 'Signin' page, featuring 'Username:' and 'Password:' input fields, a 'Login' button, and links for 'Forgot Password?', 'Create Account', and 'Return to Yioop'.

For Admin pages, each activity is controlled in an analogous fashion to the non-mobile setting, but the Activity element has been replaced with a dropdown:

The screenshot shows the 'Admin' page with a 'Manage Account' dropdown menu. Below the menu, it says 'Welcome, root!' and provides instructions on how to use the page. There is a section for 'Account Details' with an 'Edit' link, and input fields for 'Username: root' and 'First Name: admin'.

We now resume our discussion of how to use each of the Yioop admin activities for the default, non-mobile, setting, simply noting that except for the above minor changes, these instructions will also apply to the mobile setting.

[Return to table of contents](#) .

## User Accounts and Social Features Registration and Signin

Clicking on the Sign In link on the corner of the Yioop web site will bring up the following form:

The screenshot shows the 'Signin' form with the Yioop logo and the text '- Signin'. It includes 'Username:' and 'Password:' input fields, a 'Login' button, and links for 'Forgot Password?', 'Create Account', and 'Return to Yioop'.

Correctly, entering a username and password will then bring the user to the User Account portion of



the current activity is displayed. The Activity element allows the user to choose what is the current activity for the session. The choices available on the Activity element depend on the roles the user has. A default installation of Yioop comes with two predefined roles Admin and User. If someone has the Admin role then the Activity element looks like:

<b>Account Access</b>
Manage Account
Manage Users
Manage Roles
<b>Crawls</b>
Manage Crawl
Manage Classifiers
Page Options
Results Editor
Search Sources
<b>Social</b>
Manage Groups
Feeds and Wikis
Mix Crawls
<b>System Settings</b>
Manage Machines
Manage Locales
Server Settings
Security
Configure

On the other hand, if someone just has the User role, then their Activity element looks like:

<b>Account Access</b>
Manage Account
<b>Social</b>
Manage Groups
Feeds and Wikis
Mix Crawls

In addition to the Admin and User roles, there is also a Business role that can be added to either. If under [Server Settings](#), the Advertisement Source is set to Keywords Advertisements, and a user has the Business role, then the Activity element will also contain:

<b>Advertisements</b>
Manage Credits
Manage Advertisements

Over the next several sections we will discuss each of the Yioop account activities in turn. Before we do that we make a couple remarks about using Yioop from a mobile device.

[Return to table of contents](#) .

## Managing Accounts

By default, when a user first signs in to the Yioop admin panel the current activity is the Manage Account activity. This activity just lets user's change their account information using the form pictured below. It also has summary information about Crawls and Indexes (Admin account only), Groups and Feeds, and Crawl mixes. There are also helpful links from each of these sections to a related activity for managing them.

## Welcome, root!

From this page you can access and control aspects of your account.

### Account Details



Username:   
First Name:   
Last Name:   
Email:

[\[Language and Search Settings\]](#)

### Crawls and Indexes

Crawl and index the web or an existing archive and create a searchable index. You have 0 active crawls, 25 previous crawl indexes.

[\[Manage Crawls and Indexes\]](#)

### Groups and Feeds

Create or subscribe to groups to communicate with friends. You belong to 3 groups.

<a href="#">Help [wiki]</a> (42 posts, 42 threads) Last Post: <a href="#">Account Registration Wiki Page Created!</a>
<a href="#">Public [wiki]</a> (11 posts, 11 threads) Last Post: <a href="#">Main Wiki Page Created!</a>
<a href="#">ted [wiki]</a> (1 posts, 1 threads) Last Post: <a href="#">Welcome</a>

[\[Join/Manage All Groups\]](#) [\[See Combined Group Feeds\]](#)

### Crawl Mixes

Present search results from multiple indexes in the order you prefer. Share your search creations with groups of friends. You have 103 crawl mixes.


[\[Create/Manage Crawl Mixes\]](#)

Initially, the Account Details fields are grayed out. To edit them, to toggle whether a user can do keyword advertising, or to edit the user icon next to them, click the lock icon next to account details. This will allow a user to change information using their account password. A new user icon can either be selected by clicking the choose link underneath it, or by dragging and dropping an icon into the image area. The user's password must be entered correctly into the password field for changes to take effect when Save is clicked. Clicking the now opened Lock icon will cause the lock to close, these details to be grayed out, and not editable.

### Account Details

Username:   
First Name:   
Last Name:   
Email:

**Advertise With Us**

 [Drag or ... choose](#) **Password:**

It should be noted that the **Advertise With Us** check box only appears if a Yioop site administrator has selected the Advertisement Source as Keywords Advertisements in the Server Settings Activity. When this box is checked and save, a Manage Advertisements like will appear on the Activity Element on the side of the screen. If a user wants to change their password they can click the Password link label for the password field. This reveals the following additional form fields where the password can be changed:

**Account Details**

**Username:** root

**First Name:** admin

**Last Name:** admin

**Email:** root@dev.null

**Advertise With Us**

**Password:**

**New Password:**

**Retype:**

[Return to table of contents](#) .

## Managing Users, Roles, and Groups

The Manage Users, Manage Groups, and Manage Roles activities have similar looking forms as well as related functions. All three of these activities are available to accounts with the Admin role, but only Manage Groups is available to those with a standard User role. To describe these activities, let's start at the beginning... Users are people who have accounts to connect with a Yioop installation. Users, once logged in may engage in various Yioop activities such as Manage Crawls, Mix Crawls, and so on. A user is not directly assigned which activities they have permissions on. Instead, they derive their permissions from which roles they have been directly assigned and by which groups they belong to. When first launched, Manage User activity looks like:

**Add User**

**Username:**

**First Name:**

**Last Name:**

**Email:**

**Status:** Active

**Password:**

**Retype Password:**

**User List** Row 0 to 50 of 502 >> Show

Username	First Name	Last Name	Email	Groups	Status	Actions
User0	First0	Last0	user0@email.ne t	2	Active <input type="button" value="v"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
User1	First1	Last1	user1@email.ne t	3	Active <input type="button" value="v"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
User10	First10	Last10	user10@email.n et	3	Active <input type="button" value="v"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

The purpose of this activity is to allow an administrator to add, monitor and modify the accounts of users of a Yioop installation. At the top of the activity is the "Add User" form. This would allow an administrator to add a new user to the Yioop system. Most of the fields on this form are self-explanatory except the Status field which we will describe in a moment. Beneath this is a User List table. At the top of this table is a dropdown used to control how many users to display at one time. If there are more than that many users, there will be arrow links to page through the user list. There is also a search link which can be used to bring up the following Search User form:

### Search

Username: Equals  No sort

First Name: Equals  No sort

Last Name: Equals  No sort

Email: Equals  No sort

Status: Equals  Any  No sort

Search

### User List

Row 0 to 50 of 502 >> Show 50

Username	First Name	Last Name	Email	Groups	Status	Actions
root	admin	admin	root@dev.null	3	Active	<a href="#">Edit</a> <a href="#">Delete</a>
cpollett	Chris	Pollett		2	Active	<a href="#">Edit</a> <a href="#">Delete</a>
User0	First0	Last0	user0@email.ne t	2	Active	<a href="#">Edit</a> <a href="#">Delete</a>

This form can be used to find and sort various users out of the complete User List. If we look at the User List, the first four columns, Username, First Name, Last Name, and Email Address are pretty self-explanatory. The Groups column says how many groups the user belongs to. The Status column has a dropdown for each user row, this dropdown also appear in the Add User form. It represents the current status of the User and can be either Inactive, Active, or Banned. An Inactive user is typically a user that has used the Yioop registration form to sign up for an account, but who hasn't had the account activated by the administrator, nor had the account activated by using an email link. Such a user can't create or post to groups or log in. On the other hand, such a user has reserved that username so that other people can't use it. A Banned user is a user who has been banned from logging, but might have groups or posts that the administrator wants to retain. Selecting a different dropdown value changes that user's status. Next to the Status column are two action columns which can be used to edit a user or to delete a user. Deleting a user, deletes their account, any groups that the user owns, and deletes any posts the user made. The Edit User form looks like:

[Add User Form](#)

### User Information

Username:

First Name:

Last Name:

Email:

Status:

Roles: [\[1 roles\]](#)

Groups: [\[3 groups\]](#)

Password:

Retype Password:

Save

### User List

Row 0 to 50 of 502 >> Show 50  [\[Search\]](#)

Username	First Name	Last Name	Email	Groups	Status	Actions
User0	First0	Last0	user0@email.ne t	2	Active	<a href="#">Edit</a> <a href="#">Delete</a>
User1	First1	Last1	user1@email.ne t	3	Active	<a href="#">Edit</a> <a href="#">Delete</a>

This form let's you modify some of the attributes of a users. There are also two links on it: one with the number of roles that a user has, the other with the number of groups that a user has. Here the word "role" means a set of activities. Clicking on one of these links brings up a paged listing of the particular roles/groups the user has/belongs to. It will also let you add or delete roles/groups. Adding a role to a user means that the user can do the set of activities that the role contains, adding a group to the user means the user can read that group, and if the privileges for non-owners allow posting then can also post or comment to that group's feed and edit the group's wiki. This completes the description of the Manage User Activity.

Roles are managed through the Manage Role activity, which looks like:

Row 0 to 3 of 3 Show 50 [Search]

Name	Actions
Admin	<a href="#">Edit</a> <a href="#">Delete</a>
Business User	<a href="#">Edit</a> <a href="#">Delete</a>
User	<a href="#">Edit</a> <a href="#">Delete</a>

Similar to the Manage User form, at the top of this activity, there is an Add Role form, and beneath this a Role List. The controls of the Role List operate in much the same fashion as those of the User List described earlier. Clicking on the Edit link of a role brings up a form which looks like:

### Role Information

Role Name:

Activities:

- Manage Account [Delete](#)
- Manage Groups [Delete](#)
- Mix Crawls [Delete](#)
- Feeds and Wikis [Delete](#)

In the above, we have a Localizer role. We might have created this role, then used the Select Activity dropdown to add all the activities of the User role. A localizer is a person who can localize Yioop to a new language. So we might then want to use the Select dropdown to add Manage Locales to the list of activities. Once we have created a role that we like, we can then assign user's that role and they will be able to perform all of the activities listed on it. If a user has more than one role, than they can perform an activity as long as it is listed in at least one role.

Groups are collections of users that have access to a group feed and a set of wiki pages. Groups are managed through the Manage Groups activity which looks like:

### Create/Join Group


Name:  [\[Browse\]](#) [?](#)

### Subscribed Groups

Row 0 to 3 of 3 Show 50 [Search]

Name	Owner	Register	Access	Voting	Post Lifetime	Actions
<a href="#">Help</a> <a href="#">[wiki]</a>	root [2 users]	Public Request	Read Write Wiki	+/- Voting	Never Expires	<a href="#">Edit</a> <a href="#">Delete</a>
<a href="#">Public</a> <a href="#">[wiki]</a>	root [503 users]	Anyone	Read	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Delete</a>
<a href="#">ted</a> <a href="#">[wiki]</a>	root [2 users]	Public Request	No Read	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Delete</a>

Unlike Manage Users and Manage Roles, the Manage Group activity belongs to the standard User role, allowing any user to create and manage groups. As one can see from the image above The Create/Join Group form takes the name of a group. If you enter a name that currently does not exist the following form will appear:

**Create Group** 

**Name:**

**Register:**

**Access:**

**Voting:**

**Post Lifetime:**

The user who creates a group is set as the initial group owner.

The **Register dropdown** says how other users are allowed to join the group: **No One** means no other user can join the group (you can still invite other users), **By Request** means that other users can request the group owner to join the group, but that the group is not publicly visible in the browsable group directory; **Public Request** is the same as By Request, but the group is publicly visible in the browsable group directory; and **Anyone** means all users are allowed to join the group and the group appears in the browseable directory of groups. It should be noted that the root account can always join and browse for any group. The root account can also always take over ownership of any group.

The **Access dropdown** controls how users who belong/subscribe to a group other than the owner can access that group. The possibilities are **No Read** means that non-members of the group cannot read the group feed or wiki, a non-owner member of the group can read but not write the group news feed and wiki; **Read** means that a non-member of the group can read the group news feed and the groups wiki page, but non-owners cannot write the feed or wiki; **Read Comment** means that a non-owner member of the group can read the group feed and wikis and can comment on any existing threads, but cannot start new ones, **Read Write** means that a non-owner member of the group can start new threads and comment on existing ones in the group feed, but cannot edit the group's wiki; finally, **Read Write Wiki** is wiki Read Write except a non-owner member can edit the group's wiki. The access to a group can be changed by the owner after a group is created. No Read and Read are often suitable if a group's owner wants to perform some kind of moderation. Read and Read Comment groups are often suitable if someone wants to use a Yioop Group as a blog. Read Write makes sense for a more traditional bulletin board.

The **Voting dropdown** controls to what degree users can vote on posts. **No Voting** means group feed posts cannot be voted on; **+ Voting** means that a post can be voted up but not down; and **+/- Voting** means a post can be voted up or down. Yioop restricts a user to at most one vote/post.

The **Post Lifetime dropdown** controls how long a group feed post is retained by the Yioop system before it is automatically deleted. The possible values are **Never Expires** , **One Hour** , **One Day** , or **One Month** .

A default installation of Yioop has two built-in groups: **Public** and **Help** owned by root. Public has Read access and all users automatically subscribed to it and cannot unsubscribe it. It is useful for general announcements and its wiki can be used as part of building a site for Yioop. The Help group's wiki is used to maintain all the wiki pages related to Yioop's integrated help system. When a user clicks on the help icon [?], the page that is presented in blue comes from this wiki. This group's registration is by default by Public Request and its access is Read Write Wiki.

If on the Create/Join Group form, the name of a group entered already exists, but is not joinable, then an error message that the group's name is in use is displayed. If either anyone can join the group or the group can be joined by request, then that group will be added to the list of subscribed to groups. If membership is by request, then initially in the list of groups it will show up with access Request Join.

Beneath the Create/Join Group form is the Groups List table. This lists all the groups that a user is currently subscribed to:

Row 0 to 3 of 3 Show 50 [Search]

Name	Owner	Register	Access	Voting	Post Lifetime	Actions
<a href="#">Help</a> <small>(wiki)</small>	<a href="#">root</a> <small>(2 users)</small>	Public Request	Read Write Wiki	+/- Voting	Never Expires	<a href="#">Edit</a> <a href="#">Delete</a>
<a href="#">Public</a> <small>(wiki)</small>	<a href="#">root</a> <small>(503 users)</small>	Anyone	Read	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Delete</a>
<a href="#">ted</a> <small>(wiki)</small>	<a href="#">root</a> <small>(2 users)</small>	Public Request	No Read	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Delete</a>

The controls at the top of this table are similar in functionality to the controls we have already discussed for the User Lists table of Manage Users and the Roles List table of Manage Roles. This table let's a user manage their existing groups, but does not let a user to see what groups already exist. If one looks back at the Create/Join Groups form though, one can see next to it there is a link "Browse". Clicking this link takes one to the Discover Groups form and the Not Subscribed to Groups table:

**Discover Groups** [Create/Join Group](#)

Name:   No sort

Owner:   No sort

Register:   No sort

Access:   No sort

Post Lifetime:   No sort

**Not Subscribed to Groups**

Row 0 to 50 of 100 >> Show 50

Name	Owner	Register	Access	Voting	Post Lifetime	Actions
<a href="#">Group0</a> <small>(wiki)</small>	<a href="#">User0</a> <small>(2 users)</small>	Anyone	Read Write	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Join</a>
<a href="#">Group1</a> <small>(wiki)</small>	<a href="#">User1</a> <small>(101 users)</small>	Anyone	Read Write	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Join</a>
<a href="#">Group2</a> <small>(wiki)</small>	<a href="#">User2</a> <small>(1 users)</small>	Anyone	Read Write	No Voting	Never Expires	<a href="#">Edit</a> <a href="#">Join</a>

If a group is subscribable then the Join link in the Actions column of Not Subscribed to Groups table should be clickable. Let's briefly now consider the other columns of either the Groups List or not Subscribed to Groups table. The Name column gives the name of the group. Group name are unique identifiers for a group on a Yioop system. In the Groups List table the name is clickable and takes you to the group feed for that group. The Owner column gives the username of the owner of the group and beneath this the number of members of the group. If you are the root account or if you are the owner of the group, then this field should be a clickable link that take you to the following form:

**Transfer Group Owner**

Name:

New Owner:

that can be used to transfer the ownership of a group. The next two column give the register and access information for the group. If you are the owner of the group these will be dropdowns allow you to change these settings. We have already explained what the Join link does in the actions column. Other links which can appear in the actions column are Unsubscribe, which let's you leave a group which you have joined but are not the owner of; Delete, which, if you are the owner of a group, let's you delete the group, its feed, and all its wiki pages; and Edit, which displays the following form:

## Edit Group

Name:	<input type="text" value="CS175 Fall 2014"/>
Register:	<input type="button" value="Public Request"/>
Access:	<input type="button" value="Read Write"/>
Voting:	<input type="button" value="+ Voting"/>
Post Lifetime:	<input type="button" value="Never Expires"/>
Members:	<input type="text" value="[22 users]"/>
<input type="button" value="Save"/>	

The Register, Access, Voting, Post Lifetime dropdowns lets one modify the registration, group access, voting, and post lifetime properties for the group which we have already described before. Next to the Members table header is a link with the number of current members of the group. Clicking this link expands this area into a listing of users in the group as seen above. This allows one to change access of different members to the group, for example, approving a join request or banning a user. It also allows one to delete a member from a group. Beneath the user listing is a link which can take one to a form to invite more users.

## Feeds and Wikis

The initial screen of the Feeds and Wikis page has an integrated list of all the recent posts to any groups to which a user subscribes:

### Group Activity



[I just learned an interesting thing!](#) (+0/0). - 7 m 27 s ago Pollett [\[Edit\]](#) [\[X\]](#)

**Genealogy** 

Isn't that cool?

**cpollett**

Last Edited: 3 m 55 s ago

Post Vote:

[Comment.](#)



[Welcome to the Test Group!](#) (+1). - 21/01/2015 Test

This **group** 's purpose is to talk about tests. These include:

- software tests
- math tests

Post Vote:

[Comment.](#)

The arrow icon next to the feed allow one to collapse the Activities element to free up screen real estate for the feed. Once collapsed, an arrow icon pointing the opposite direction will appear to let you show the Activities element again. Next to the Group Activity header at the top of the page are two icons:



These control whether the Feed View above has posts in order of time or if posts are arranged by group as below:



## Group Activity

[CS175 Fall 2014 \[wiki\]](#) (99 posts, 54 threads)

Last Post: [-- Practice Final - Problem 5](#)

[CS 156 Fall 2014 \[wiki\]](#) (142 posts, 75 threads)

Last Post: [Hw5 Scores Back!](#)

[Chris Blog \[wiki\]](#) (5 posts, 3 threads)

Last Post: [-- My First Blog Post](#)

[Eswara CS298 \[wiki\]](#) (4 posts, 4 threads)

Last Post: [User Assisted Testing Part Two Wiki Page Created!](#)

[Yloop Install & Config Help \[wiki\]](#) (6 posts, 2 threads)

Last Post: [-- test](#)

Going back to the original feed view above, notice posts are displayed with the most recent post at the top. If there has been very recent activity (within the last five minutes), this page will refresh every 15 seconds for up to twenty minutes, checking for new posts. Each post has a title which links to a thread for that post. This is followed by the time when the post first appeared and the group title. This title, although gray, can be clicked to go to that particular group feed. If the user has the ability to start new threads in a group and one is in single feed mode, an icon with a plus-sign and a pencil appears next to the group name, which when clicked allows a user to start a new thread in that group. Beneath the title of the post, is the username of the person who posted. Again, this is clickable and will take you to a page of all recent posts of that person. Beneath the username, is the content of the post. On the opposite side of the post box may appear links to Edit or X (delete) the post, as well as a link to comment on a post. The Edit and X delete links only appear if you are the poster or the owner of the group the post was made in. The Comment link lets you make a follow up post to that particular thread in that group. For example, for the "I just learned an interesting thing!" post above, the current user could start a new thread by clicking the plus-pencil icon or comment on this post by clicking the Comment link. If you are not the owner of a group then the Comment and Start a New Thread links only appear if you have the necessary privileges on that group.

The image below shows what happens when one clicks on a group link, in this case, the Chris Blog link.

## Chris Blog [Feed|Wiki]

[my Group Feeds](#)

Start New Thread



[Test Page 2 Wiki Page Created!](#) (1 posts, 14 views) . - 26/06/2014 **Chris Blog**  
Last Post: 26/06/2014 - [cpollett](#)

cpollett



[My First Blog Post](#) (2 posts, 35 views) . - 26/06/2014 **Chris Blog**  
Last Post: 23/06/2014 - [cpollett](#)

cpollett



[Test Page Wiki Page Created!](#) (1 posts, 27 views) . - 26/06/2014 **Chris Blog**  
Last Post: 26/06/2014 - [cpollett](#)

cpollett

On the opposite side of the screen there is a link My Group Feeds, which let's one go back to the previous screen. At the top of this screen is clickable title of the group, in this case, Chris Blog, this takes one to the Manage Groups activity where properties of this group could be examined. Next we see a toggle between Feed and Wiki. Currently, on group feed page, clicking Wiki would take one to the Main page of the wiki. Posts in the single group view are grouped by thread with the thread containing the most recent activity at the top. Notice next to each thread link there is a count of the number of posts to that thread. The content of the thread post is the content of the starting post to the thread, to see latter comments one has to click the thread link. There is now a Start New Thread button at the top of the single group feed as it is clear which group the thread will be started in. Clicking this button would reveal the following form:

Start New Thread

**Subject**

**Post**

**B** **I** **U** **\$** **W** **↔** **☰** **☷** **☰** **☷** **☰** **☷** **☰** **☷** **☰** **☷** **☰** **☷** **☰** **☷**

Drag items into the textarea to add them...[or click to select them.](#)

Save

Adding a Subject and Post to this form and clicking save would start a new thread. Posts can make use of [Yioop's Wiki Syntax](#) to achieve effects like bolding text, etc. The icons above the text area can be used to quickly add this mark-up to selected text. Although the icons are relatively standard, hovering over an icon will display a tooltip which should aid in figuring out what it does. Beneath the text area is a dark gray area with instructions on how to add resources to a page such as images, videos, or other documents. For images and videos these will appear embedded in the text of the post when it is saved, for other media a link to the resource will appear when the source is saved. The size allowed for uploaded media is determined by your php instances php.ini configuration file's values for post\_max\_size and upload\_max\_filesize. Yioop uses the value of the constant MAX\_VIDEO\_CONVERT\_SIZE set in a configs/local\_config.php or from configs/config.php to determine if a video should be automatically converted to the two web friendly formats mp4 and

has been given as a constant FFMPEG in either a configs/local\_config.php or in configs/config.php.

Clicking the comment link of any existing thread reveals the following form to add a comment to that thread:

### [Chris Blog](#) :My First Blog Post [\[RSS\]](#) [My Group Feeds](#)

Comment



[My First Blog Post](#) - 23/06/2014 [Chris Blog](#) [\[Edit\]](#) [\[X\]](#)  
This is a test post in my new blog!

cpollett



-- [My First Blog Post](#) - 26/06/2014 [Chris Blog](#) [\[Edit\]](#) [\[X\]](#)  
Cool! I have a new blog

cpollett

Comment

**Add a Comment**

**B** *I* U ~~S~~           

Drag items into the textarea to add them...[or click to select them.](#)

Save

Below we see an example of the feed page we get after clicking on the My First Blog Post thread in the Chris Blog group:

## Chris Blog :My First Blog Post [RSS]

Comment



[My First Blog Post.](#) - 23/06/2014 Chris Blog

This is a test post in my new blog!

[\[Edit\]](#) [\[X\]](#)



-- [My First Blog Post.](#) - 26/06/2014 Chris Blog

Cool! I have a new blog

[\[Edit\]](#) [\[X\]](#)



-- [My First Blog Post.](#) - 0 m 0 s ago Chris Blog

Great!

[\[Edit\]](#) [\[X\]](#)

Comment

1

Since we are now within a single thread, there is no Start New Thread button at the top. Instead, we have a Comment button at the top and bottom of the page. The starting post of the thread is listed first and ending most recent post is listed last (paging buttons both on the group and single thread page, let one jump to the last post). The next image below is an example of the feed page one gets when one clicks on a username link, in this case, cpollett:

## cpollett User Feed

[My Group Feeds](#)



[Hw4 grades back!](#). - 24/11/2014 CS 156 Fall 2014

Hey Everyone,  
Hw4 grades are up!  
Chris

[Comment.](#)



[Hw5 is up!](#). - 23/11/2014 CS 156 Fall 2014

Hey Everyone,  
Hw5 is up!  
Chris

[Comment.](#)



[Hw5 up! \(+0\)](#). - 19/11/2014 CS175 Fall 2014

Hey Everyone,  
Hw5 is up. This is a more flexible project than the previous ones. So you will need to scope it carefully. Feel free to run your project ideas to see if I think they would be feasible in the given time frame.  
Best, Chris  
Post Vote:

[Comment.](#)

When a user starts a new thread or comments to an existing thread, several people will be notified via email provided they have their email address configured properly. For a new thread, the groups owner will be notified if they themselves weren't the ones starting the thread. If it was in fact the group's owner that starts a thread, then everyone who belongs to the group will be notified. For a comment to a thread, excluding the commenter, the group's owner, the person who started the

comment by the group's owner, it is still only the people participating in the thread who are notified.

Single Group, Threads, and User feeds of groups which anyone can join (i.e., public groups) all have RSS feeds which could be used in a news aggregator or crawled by Yioop. To see what the link would be for the item you are interested in, first collapse the activity element if its not collapsed (i.e., click the [<<] link at the top of the page). Take the URL in the browser in the url bar, and add ? or & followed by f=rss to it. It is okay to remove the YIOOP\_TOKEN= variable from this URL. Doing this for the cpollett user feed, one gets the url:

<http://www.yioop.com/user/4?f=rss>

whose RSS feed looks like:



Adding f=json and f=json&callback=myFunction also, work if JSON or JSONP formats are preferred. As we mentioned above when we described the single group feed page, if we click on the Wiki link at the top we go to the Main wiki page of the group, where we could read that page. If the Main Wiki page (or for that if matter if we go to any wiki page that) does not exist, then we would get like the following:



This page might be slightly different depending on whether the user has write access to the given group. The [Wiki Syntax Guide](#) link in the above takes one to a page that describes how to write wiki pages. The Edit link referred to in the above looks slightly different and is in a slightly different location depending on whether we are viewing the page with the Activity element collapsed or not. If the Activity element is not collapsed then it appears one of three links within the current activity as:



On the other hand, if the Activity element is collapsed, then it appear on the navigation bar at the top of the screen as:



Notice besides editing a page there is a link to read the page and a link Pages. The Pages link takes us to a screen where we can see all the pages that have been created for a group:



having to remember it. Also, as mentioned above the dark gray area below the textarea describes how to associate images, video, and other media to the document. Unlike with posts, a complete list of currently associated media can be found at the bottom of the document under the **Page Resources** heading. Links to Rename, Add a resource to the page, and Delete each resource can also be found here. Clicking on the icon next to a resource let's you look at the resource on a page by itself. This icon will be a thumbnail of the resource for images and videos. In the case of videos, the thumbnail is only generated if the FFmpeg software mentioned earlier is installed and the FFmpeg constant is defined. In this case, as with posts, if the video is less than MAX\_VIDEO\_CONVERT\_SIZE, Yioop will automatically try to convert it to mp4 and webm so that it can be streamed by Yioop using HTTP pseudo-streaming.

Clicking the **Settings Link** next to the wiki page name reveals the following additional form elements:

**Locale:** en-US [\[History\]](#) [\[Discuss\]](#)

**Page:** Test Page [\[Settings\]](#)

**Page Type:** Standard

**Page Border:** Solid

**Table of Contents:**

**Title:**










**Author:**

**Meta Robots:**

**Meta Description:**

**Header Page Name:**

**Footer Page Name:**

**B** **I** **U** **S**         **Heading**  

=Test Page=

The meaning of these various settings is described in [Page Settings](#), [Page Type](#) section of the Yioop Wiki Syntax Guide.

The **Discuss link** takes you to a thread in the current group where the contents of the wiki page should be discussed. Underneath the textarea above is a Save button. Every time one clicks the save button a new version of the page is saved, but the old one is not discarded. We can use the Edit Reason field to provide a reason for the changes between versions. When we read a page it is the most recent version that is displayed. However, by clicking the History link above we can see a history of prior version. For example:

**Test Page Page** - [\[Read\]](#) [\[Edit\]](#) [\[Pages\]](#)

**Difference:**  -   [Back](#)

[\(First\)](#) [\(Second\)](#) [2015-01-24T12:03:56-08:00](#), 1422129836. Edited by cpollett. (260 bytes). [\[Revert\]](#).

[\(First\)](#) [\(Second\)](#) [2015-01-24T12:03:23-08:00](#), 1422129803. Edited by cpollett. (195 bytes). [\[Revert\]](#).

1

The time link for each version can be clicked to view that prior version without reverting. The First and Second links next to a version can be used to set either the first field or second field at the top of the history page which is labelled Difference: . Clicking the Go button for the Difference form computes the change set between two selected versions of a wiki document. This might look like:

**Chris Blog**[\[Feed\]](#)[\[Wiki\]](#)

Test Page Page - [\[Read\]](#)[\[Edit\]](#)[\[Pages\]](#)

Test\_Page line differences between 2015-01-24T16:19:34-08:00 and 2015-01-24T12:03:56-08:00. [Back](#)

+++ Test\_Page "1422145174"  
 --- Test\_Page "1422129836"  
 -Test Page for Chris' Blog  
 +Page for Chris' Blog

This completes the description of group feeds and wiki pages.

[Return to table of contents](#) .

## Keyword Advertising

Keyword advertising allows a user with the Business Role to pay to associate a query term or phrase with a text-based advertisement for a fixed number of days. By default, Yioop does not come with an advertisement platform turned on. To enable keyword advertising, one can select **Keyword Advertisements** from the dropdown in the **Ad Server Configuration** fieldset of the [Server Settings](#) activity. This turns on keyword advertising. A user can then choose to upgrade their account to do advertising as described in the [Manage Account](#) section. If this is done, the following new activities will appear in the side activity panel:

**Advertisements**

Manage Credits

Manage Advertisements

The **Manage Credits** activity allows a user to purchase Ad Credits with a credit card. Ad Credits in turn can be used to pay for advertisements created in **Manage Advertisements** .

The Manage Credits activity looks like

**Purchase Ad Credits** [?](#)

Quantity:

Credit Card Number:

CVC:

Expiration:   /

Using the Purchase button charges the above card the Quantity field's amount in US dollars and agrees to the [Ad Program Terms](#).

**Balance: 1940 credits**

**Ad Credit Transactions** Row 0 to 3 of 3 Show

Type	Amount	Date	Total
Buy Ad	-60	Mon, 31 Aug 2015 08:09:22 -0700	1940
Credit Purchase	2000	Sun, 30 Aug 2015 21:17:15 -0700	2000
Starting Balance	0	Sun, 30 Aug 2015 20:53:57 -0700	0

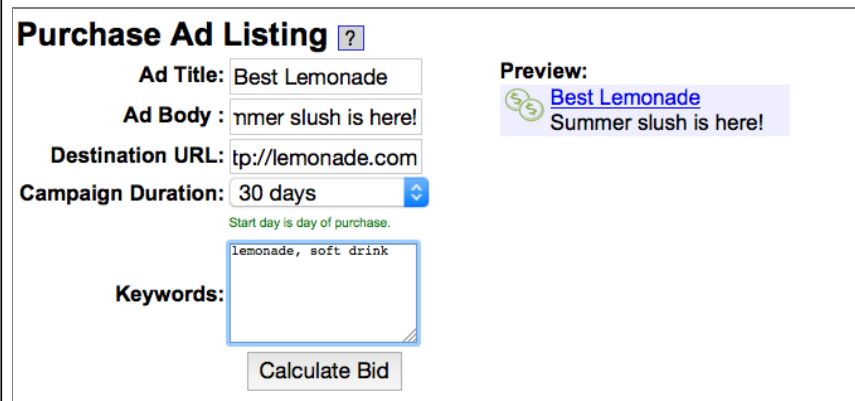
The **Purchase Ad Credits** form is used to actually purchase ad credits. On it, the **Quantity** dropdown specifies the number of credits one wants to purchase at what price. The **Card Number**



card verification number for your card, finally, the **Expiration** dropdown is used to set your cards expiration date.

Beneath the Purchase form is the list of **Ad Credit Transactions** that have been made with your account.

Clicking on the **Manage Advertisements** activity link will display a form like:



**Purchase Ad Listing** [?](#)

**Ad Title:** Best Lemonade

**Ad Body :** mmer slush is here!

**Destination URL:** tp://lemonade.com

**Campaign Duration:** 30 days

Start day is day of purchase.

**Keywords:** lemonade, soft drink

**Preview:**

Best Lemonade  
Summer slush is here!

We split the remainder of this section into three subsections: the Manage Advertisements activity, the mechanics of how ad bids are calculated, and how payment processing is done.

### The Manage Advertisements Activity

Returning to the screenshot at the end of the last section, the **Ad Title** , **Ad Body** , **Destination URL** fields can be used to create a text-based advertisement. What this ad will look like appears in the **Preview** area.

The **Duration** dropdown controls how many days the ad campaign will run for. The campaign starts on the date of purchase. The first day of a campaign starts from the time of purchase and last until midnight in the Yioop instance's timezone. Subsequent days start at midnight and last till the following midnight.

**Keywords** should consist of a comma separated list of words or phrases. Each word or phrase has a minimum bid for each day based on demand for that keyword. If no one so far has purchased an ad for any of the keywords, then this minimum is \$1/day/(word or phrase). Otherwise, it is calculated using the total of the bids so far. Keywords can include Yioop meta words such as media:news.

The **Calculate Bid** button computes the minimum cost for the campaign you have chosen, and then presents a form to receive your credit card information.

**Purchase Ad Listing** [?](#)

**Ad Title:** Best Lemonade

**Ad Body :** Summer slush is here!

**Destination URL:** http://lemonade.co

**Campaign Duration:** 30 days   
Start day is day of purchase.

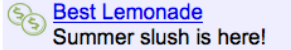
**Keywords:**

**Minimum Bid Required:** 60

**Expensive word** SOFT DRINK

**Budget:**

Using the Purchase button deducts the Budget field amount from your 2000 available ad credits. [Buy more ad credits.](#)

**Preview:**  
 Best Lemonade  
 Summer slush is here!

On this form the static field **Minimum Bid Required** displays the minimum amount required to pay for the advertisement campaign in question. The **Expensive word** static field says for your campaign which term contributes the most to this minimum bid cost. The Budget fields allows you to enter an amount greater than or equal to the minimum bid that you are willing to pay your ad campaign. If there have been no other bids on your keywords then the minimum bid will show your ad 100% of the time any of your keywords are searched for on the Yioop instance. If, however, there have been other bids, your bid amount as a fraction of the total bid amount for that day for the search keyword is used to select a frequency with which your ad is displayed. Thus, it can make sense to bid more than the minimum required amount.

If you need to edit the keywords or other details of your ad before purchasing it, you can click the **Edit Ad** button; otherwise, clicking the **Purchase** button completes the purchase of your Ad campaign. After purchase, your ad will appear an advertisement list beneath the Purchase Ad listing form.

**Purchase Ad Listing** [?](#)


**Ad Title:**

**Ad Body :**

**Destination URL:**

**Campaign Duration:** Number of Days   
Start day is day of purchase.

**Keywords:**

**Preview:**  


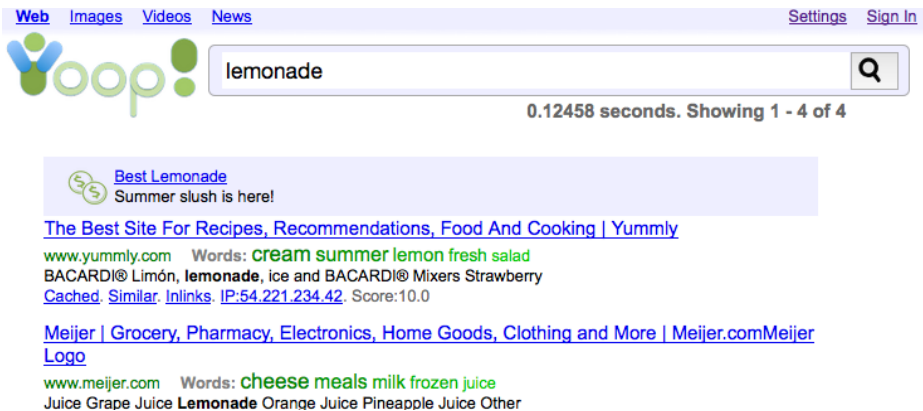
**Advertisement List** Row 0 to 1 of 1 Show

Ad Name	Username	Keywords	Budget	Date	Views/Clicks	Status	Actions
<a href="#">Best Lemonade</a>	root	LEMONADE, SOFT DRINK	60	2015-08-16 / 2015-09-14	0 / 0	Active	<a href="#">Edit</a> <a href="#">Deactivate</a>

The **Advertisement List** is a list of all current and previous ads in descending order by when the campaign was created. This form appears slightly differently for an administrative user than a normal user. For the former, as in the above image, there is a *Username* column, and ads not belonging to the current user will be shown. For these ads, the administrator gets a link to suspend the ad. In the normal user mode, the table does not have a *Username* column and only the current user's ads are displayed. A normal user can deactivate and reactivate their own ads, but can't reactivate an ad that

or links associated with an active ad. One can determine how many times the ad was served and how many times an ad was clicked so far during a campaign by looking at the **Views/Clicks** column.

Once an ad is purchased, for the duration of the campaign, if someone searches on the ad keywords (in the case of the ad above, either the word *lemonade* or the phrase *soft drink*), it has an opportunity to be displayed at the top of the search results. This would typically look something like:



## Bid Mechanics

In this subsection, we describe how the minimum purchase price for an ad is determined, how ads are chosen for display, and try to give some intuitions on the ad bidding process.

The minimum purchase of an ad campaign is determined by adding together the minimum costs of the different keywords involved in that campaign. Let CR denote the value of an ad credit. By default in Yioop, one ad credit is about 1 cent. As an example, if one had three keywords and the cost of each separately was 7CR, 10CR, and 5CR, then the minimum purchase price would be 22CR. In turn, the minimum purchase price of a keyword for an ad campaign, is the sum of the current costs of the ad for each day during the campaign. For example, suppose we were running an ad on the single keyword *lemonade* for one week, the minimum cost for the current day was 10CR, and for subsequent days was 9CR, 7CR, 9CR, 5CR, 2CR, 1CR. Then the minimum purchase price for that keyword for the seven days would be  $(10 + 9 + 7 + 9 + 5 + 2 + 1)CR = 43CR$ . The minimum cost of a keyword on a particular day is half the sum of the purchase for that keyword for that day so far rounded up. For example, suppose for a particular day, ads on the word *lemonade* had been purchased for 1CR, 1CR, 1CR, 2CR, 3CR, and 4CR. The total credit amount spent on *lemonade* for that day would then be 12CR. So the minimum next bid for the word *lemonade* for that day would be half that amount round up, in this case, 6CR.

The Yioop keyword advertising system allows one to bid higher than the minimum purchase price for an ad campaign. As we will see below there might be strategic reasons for doing this. Let

$\alpha = \frac{\text{bid amount}}{\text{minimum bid amount}}$ . Then the bid amount this corresponds to for a keyword for a

particular day, is  $\alpha$  times the minimum bid for that keyword for that day. For example, suppose the minimum purchase price for a seven day campaign on the keywords *lemonade* and *soft drink* was 24CR, but the campaign was purchased for 36CR. Then  $\alpha$  would be 1.5. If for *lemonade* the daily minimum bids were 4CR, 2CR, 2CR, 2CR, 2CR, 2CR, 2CR and those for *soft drink* were 2CR, 1CR, 1CR, 1CR, 1CR, 1CR, 1CR, then this ad purchase would correspond to bids for these days of 1.5 times this minimum bids for these days. For *lemonade*, this is relatively straightforward and results in the bids of 6CR, 3CR, 3CR, 3CR, 3CR, 3CR, 3CR. For *soft drink*, we might end up with fractional values. To handle this, Yioop rounds down the fractional value, but also keeps track of the sum of the left over fractions so far, if this sum ever exceeds 1, then 1CR is added to that days bid, and left over is deducted by one. So on the *soft drink* bids we get,  $1.5 \times 2CR = 3CR$ ,  $\lfloor 1.5 \times 1 \rfloor = 1CR$  with the left over at 0.5,  $\lfloor 1.5 \times 1 \rfloor = 1CR$  with the left over now at  $0.5 + 0.5 = 1$ , so the bid value is upped to 2CR and the left over is deducted by 1 back to 0. Continuing in this fashion for the

the last keyword on the last day, then an additional bid of 1 is made for this last keyword on the last day.

Once an ad is purchased, it can be displayed when a user performs a search. Ads are only displayed for simple queries, that is, queries not involving the operators | or #. Given such a query, Yioop performs a look up to see if there are any ads for that day which exactly match the query phrase. For example, if the query was "soft drink" then Yioop would do a look up to see if there are any active ad campaigns with "soft drink" as one of their keywords. If there are, Yioop moves on to an ad selection phase, if there are no such ads, then Yioop looks to see if there are ads involving single terms in the query. In this case, there are two possible single terms "soft" and "drink". On the query "lemon soft drink", there would be three: "lemon", "soft", and "drink". If there are ad campaigns involving these single terms, then Yioop selects the term for which the most money has been spent and proceeds to the ad selection phase using that term.

In the ad selection phase, Yioop arranges the ad campaigns for the chosen keyword in the order in which the purchases were made. It uses a pseudo-random number generator (PRNG) to pick an integer less than the total bid on that keyword for the current day. It then computes sums of ad prices from initial sublists of campaigns until it finds the first partial sum greater than or equal to the integer. Finally, it chooses the ad campaign that corresponded to the last summand in this sublist. For example, suppose the query was "soft drink" and there were four purchased ads with the "soft drink" keyword for that day. Bid 1 was 1CR, Bid 2 was 1CR, Bid 3 was 3CR, and Bid 4 was 4CR. The total spend on this keyword for the current day would be 9CR. If the PRNG picked the number 3 from among the choices 0, 1, 2, 3, 4, 5, 6, 7, 8, then Yioop would start computing initial sums. The first value greater than this is 4. The blank sum 0CR is less than 4CR, so it would add Bid 1 to get the partial sum 1CR, then it adds Bid 2 to get the partial sum 2CR, and finally, it would add Bid 3's bid to get the partial sum 5CR. As  $5CR \geq 4CR$ , and the last bid added was Bid 3's, it would be Bid 3's ad that would be displayed for this particular search. As the PRNG chooses a different number reasonably uniformly from 0, 1, 2, 3, 4, 5, 6, 7, 8 each time the query "soft drink" is made, the frequency with which a given ad for that term is displayed will be proportional to the amount that that ad was purchased for.

We now conclude this section with some observations about this bidding system. The first observation is that days further in the future will tend to have fewer bids for keywords on those days as there has been less time for people to bid on those days. Currently, the maximum campaign length is 180 days, but most campaign will probably be shorter. Hence, on a cost/day basis for a minimum bid for a keyword is like to be lower for a longer campaign. A second observation is that if your valuation of a keyword on a given day is more than twice anyone else's valuation, then if you bid your valuation first for that day, no one else will try to bid as half the total will be more than their valuation. So truthful bidding can be advantageous.

Many keyword advertisement systems use some kind of auction system, so it is interesting to compare Yioop's system with an auction. One could imagine have an open, ascending price auction for a keyword to purchase all impressions/clicks that that ad generates from Yioop searches for a day. Suppose for the keyword lemonade, searches generate on average 100 impressions in a day. If two bidders both value an impression at 1CR, then both would value the lemonade keyword for one day at 100CR. The first person would bid 100CR, and the second person would not bid as then he would be paying more than a 1CR/impression. The first person wouldn't underbid as then the second would have the opportunity to take all 100 impressions. Now consider what would happen with Yioop's bidding system. If the first person bids 100CR for lemonade for the day, then the second person would have a minimum bid of 50CR. If the second person bids 50CR, then they will receive about  $\frac{50}{100 + 50} \cdot 100 \approx 33$  impressions, so the cost/impression will be over a credit. Higher bids would only make the cost/impression worse, so the second bidder would not bid. Hence, in this situation Yioop's bidding system has the same effect as an auction.

Now consider a situation in which one has two bidders, both of whom value an impression at 1CR, but both of whom have a limited budget of 50CR. In the auction system, the first bidder would bid

do so. So the first bidder gets all the clicks for 0.5CR each, the auction house has lost potential revenue on the item, and the second bidder doesn't get there ad displayed at all. In the Yioop system, the second bidder could bid 50CR as well, each bidder would get half of the 100 impressions, that is, 50 each, and they would each be paying 1CR/impression. So in this situation the Yioop system is advantageous to the seller and gives more bidders an opportunity to have their ads displayed at a fair price.

## Payment Processing

Payment processing entails actually charging the credit card based on the data collected **Purchase Ad Credits** form after Purchase is clicked. The default Yioop download comes with a stub class in src/configs/CreditConfig.php which does not actually charge any credit card, but then gives the user that many credit. I.e., effectively all ad credits are free. This allows you to experiment with how the advertising platform works. The link below take you to a version of this file which uses [stripe.com](https://stripe.com) for payment processing:

[Yioop Keyword Ad Script.](#)

One advantage to using stripe.com for handling payments is that your never store the credit data on your Yioop instances servers -- the data is only sent to the stripe.com servers. You should still make sure the page that collect the purchase form is on uses https rather than http.

[Return to table of contents](#) .

## Crawling and Customizing Results Performing and Managing Crawls

The Manage Crawl activity in Yioop looks like:

### Create Crawl

Name:  Start New Crawl [Options](#) ?

### Currently Processing

Description: Open Web Crawl Oct 10 2014 Stop Crawl

[Search Index] [Change Crawl Options](#)

Timestamp: 1412956124

Time started: Fri, 10 Oct 2014 08:48:44 -0700

Indexer Peak Memory: 1559499752

Scheduler Peak Memory: 629217200

Fetcher Peak Memory: 509321536

Web App Peak Memory: 74608520

Visited Urls/Hour: 169065.78

Visited Urls Count: 312771348

Total Urls Seen: 13390581822

Most Recent Fetcher: 1-SearchFG2 @ Sat, 24 Jan 2015 16:40:53 -0800

### Most Recent Urls

<http://www.fondation-seligmann.org/rechercheArticles.php?PHPSESSID=5fe04566e732dac71e9252d1812eb429>

<http://www.fondation-seligmann.org/le-conseil-d-administrati>

<http://www.fondation-seligmann.org/prix-seligmann-contre-le-racisme-2011/138/26?PHPSESSID=d93f8e8e3fb5a097877b9b295cf4526e>

<http://www.fondation-seligmann.org/evenements/14/27?PHPSESSID=d93f8e8e3fb5a097877b9b295cf4526e>

<http://www.fondation-seligmann.org/le-conseil-d-animation/128/17?PHPSESSID=fe31ce4db3d092499f8aa816c33bd72>

<http://www.streanate.com/cam/AlonaFlirt/?langchoice=nl>

[http://www.statchukd.com/images/avatars/low-res/avatar98744\\_20.jpg](http://www.statchukd.com/images/avatars/low-res/avatar98744_20.jpg)

[http://www.statchukd.com/images/avatars/low-res/avatar31022\\_2\\_10.jpg](http://www.statchukd.com/images/avatars/low-res/avatar31022_2_10.jpg)

<http://www.lancerregister.com/member.php?s=6ebb96db0cc36a4ae9b6e5ef9348c24u=152466>

[http://www.statchukd.com/images/avatars/low-res/avatar11368\\_3\\_2.jpg](http://www.statchukd.com/images/avatars/low-res/avatar11368_3_2.jpg)

### Previous Crawls

Row 0 to 6 of 6 Show  ⌵

Description:	Timestamp:	Visited/Extracted Urls:	Actions
RECRWL::English Wikipedia 2013 <a href="#">[Statistics]</a>	1374028420 Tue, 16 Jul 2013 19:33:40 -0700	12522400/134166883	<a href="#">Resume</a> <a href="#">Set as Index</a> <a href="#">Delete</a>
RECRWL::Open	1374442375		<a href="#">Set</a>

This activity will actually list slightly different kinds of peak memory usages depending on whether

done when a single queue server was being run from the terminal. The first form in this activity allows you to name and start a new web crawl. Next to the Start New Crawl button is an Options link, which allows one to set the parameters under which the crawl will execute. We will return to what the Options page looks like in a moment. When a crawl is executing, under the start crawl form appears statistics about the crawl as well as a Stop Crawl button. Crawling continues until this Stop Crawl button is pressed or until no new sites can be found. As a crawl occurs, a sequence of IndexShard's are written. These keep track of which words appear in which documents for groups of 50,000 or so documents. In addition an IndexDictionary of which words appear in which shard is written to a separate folder and subfolders. When the Stop button is clicked the "tiers" of data in this dictionary need to be logarithmically merged, this process can take a couple of minutes, so after clicking stop do not kill the queue server (if you were going to) until after it says waiting for messages again. Beneath this stop button line, is a link which allows you to change the crawl options of the currently active crawl. Changing the options on an active crawl may take some time to fully take effect as the currently processing queue of urls needs to flush. At the bottom of the page is a table listing previously run crawls. Next to each previously run crawl are three links. The first link lets you resume this crawl, if this is possible, and say Closed otherwise. Resume will cause Yioop to look for unprocessed fetcher data regarding that crawl, and try to load that into a fresh priority queue of to crawl urls. If it can do this, crawling would continue. The second link let's you set this crawl's result as the default index. In the above picture there were only two saved crawls, the second of which was set as the default index. When someone comes to your Yioop installation and does not adjust their settings, the default index is used to compute search results. The final link allows one to Delete the crawl. For both resuming a crawl and deleting a crawl, it might take a little while before you see the process being reflected in the display. This is because communication might need to be done with the various fetchers, and because the on screen display refreshes only every 20 seconds or so.

### Prerequisites for Crawling

Before you can start a new crawl, you need to run at least one QueueServer.php script and you need to run at least one fetcher.php script. These can be run either from the same Yioop installation or from separate machines or folder with Yioop installed. Each installation of Yioop that is going to participate in a crawl should be configured with the same name server and server key. Running these scripts can be done either via the command line or through a web interface. As described in the Requirements section you might need to do some additional initial set up if you want to take the web interface approach. On the other hand, the command-line approach only works if you are using only one queue server. You can still have more than one fetcher, but the crawl speed in this case probably won't go faster after ten to twelve fetchers. Also, in the command-line approach the queue server and name server should be the same instance of Yioop. In the remainder of this section we describe how to start the QueueServer.php and fetcher.php scripts via the command line; the GUI for Managing Machines and Servers section describes how to do it via a web interface. To begin open a command shell and cd into the executables subfolder of the Yioop folder. To start a queue server type:

```
php QueueServer.php terminal
```

To start a fetcher type:

```
php Fetcher.php terminal
```

The above lines are under the assumption that the path to php has been properly set in your PATH environment variable. If this is not the case, you would need to type the path to php followed by php then the rest of the line. If you want to stop these programs after starting them simply type CTRL-C. Assuming you have done the additional configuration mentioned above that are needed for the GUI approach managing these programs, it is also possible to run the queue server and fetcher programs as daemons. To do this one could type respectively:

```
php QueueServer.php start
```

Or

```
php Fetcher.php start
```

of the WORK\_DIRECTORY folder. To stop these daemons one types:

```
php QueueServer.php stop
```

or

```
php Fetcher.php stop
```

Once the queue server is running and at least one fetcher is running, the Start New Crawl button should work to commence a crawl. Again, it will up to a minute or so for information about a running crawl to show up in the Currently Processing fieldset. During a crawl, it is possible for a fetcher or the queue server to crash. This usually occurs due to lack of memory for one of these programs. It also can sometimes happen for a fetcher due to flakiness in multi-curl. If this occurs simply restart the fetcher in question and the crawl can continue. A queue server crash should be much rarer. If it occurs, all of the urls to crawl that reside in memory will be lost. To continue crawling, you would need to resume the crawl through the web interface. If there are no unprocessed schedules for the given crawl (which usually means you haven't been crawling very long), it is not possible to resume the crawl. We have now described what is necessary to perform a crawl we now return to how to set the options for how the crawl is conducted.

### Common Crawl and Search Configurations

When testing Yioop, it is quite common just to have one instance of the fetcher and one instance of the queue server running, both on the same machine and same installation of Yioop. In this subsection we wish to briefly describe some other configurations which are possible and also some src/configs/Config.php configurations that can affect the crawl and search speed. The most obvious config.php setting which can affect the crawl speed is NUM\_MULTI\_CURL\_PAGES. A fetcher when performing downloads, opens this many simultaneous connections, gets the pages corresponding to them, processes them, then proceeds to download the next batch of NUM\_MULTI\_CURL\_PAGES pages. Yioop uses the fact that there are gaps in this loop where no downloading is being done to ensure robots.txt Crawl-delay directives are being honored (a Crawl-delayed host will only be scheduled to at most one fetcher at a time). The downside of this is that your internet connection might not be used to its fullest ability to download pages. Thus, it can make sense rather than increasing NUM\_MULTI\_CURL\_PAGES, to run multiple copies of the Yioop fetcher on a machine. To do this one can either install the Yioop software multiple times or give an instance number when one starts a fetcher. For example:

```
php Fetcher.php start 5
```

would start instance 5 of the fetcher program.

Once a crawl is complete, one can see its contents in the folder WORK\_DIRECTORY/cache/IndexDataUNIX\_TIMESTAMP. In the multi-queue server setting each queue server machine would have such a folder containing the data for the hosts that queue server crawled. Putting the WORK\_DIRECTORY on a solid-state drive can, as you might expect, greatly speed-up how fast search results will be served. Unfortunately, if a given queue server is storing ten million or so pages, the corresponding IndexDataUNIX\_TIMESTAMP folder might be around 200 GB. Two main sub-folders of IndexDataUNIX\_TIMESTAMP largely determine the search performance of Yioop handling queries from a crawl. These are the dictionary subfolder and the posting\_doc\_shards subfolder, where the former has the greater influence. For the ten million page situation these might be 5GB and 30GB respectively. It is completely possible to copy these subfolders to a SSD and use symlinks to them under the original crawl directory to enhance Yioop's search performance.

### Specifying Crawl Options and Modifying Options of the Active Crawl

As we pointed out above, next to the Start Crawl button is an Options link. Clicking on this link, let's you set various aspect of how the next crawl should be conducted. If there is a currently processing crawl, there will be an options link under its stop button. Both of these links lead to similar pages, however, for an active crawl fewer parameters can be changed. So we will only describe the first link. We do mention here though that under the active crawl options page it is possible to inject new seed urls into the crawl as it is progressing. In the case of clicking the Option link next to the start

## Edit Crawl Options

[Back](#)

Web Crawl
Archive Crawl

**Get Crawl Options From:** Use options below ⌵

**Crawl Order:** Page Importance ⌵ ?

**Restrict Sites By Url:**

**Allowed To Crawl Sites** ?

domain:allrecipes.com  
 domain:food.com  
 domain:betterrecipes.com  
 domain:foodnetwork.com  
 domain:bettycrocker.com

**Disallowed Sites/Sites with Quotas** ?

domain:arxiv.org  
 domain:ask.com  
 domain:yelp.com  
 domain:clixsense.com

**Seed Sites** [Add User Suggest data](#) ?

http://restaurant.betterrecipes.com/  
 http://www.food.com

The Back link in the corner returns one to the previous activity.

There are two kinds of crawls that can be performed by Yioop either a crawl of sites on the web or a crawl of data that has been previously stored in a supported archive format such as data that was crawled by Versions 0.66 and above of Yioop, data coming from a database or text archive via Yioop's importing methods described below, [Internet Archive ARC file](#), [ISO WARC Files](#), [MediaWiki xml dump](#), [Open Directory Project RDF file](#). In the next subsection, we describe new web crawls and then return to archive crawls subsection after that. Finally, we have a short section on some advanced crawl options which can only be set in config.php or local\_config.php. You will probably not need these features but we mention them for completeness

### Web Crawl Options

On the web crawl tab, the first form field, "Get Crawl Options From", allows one to read in crawl options either from the default\_crawl.ini file or from the crawl options used in a previous crawl. The rest of the form allows the user to change the existing crawl options. The second form field is labeled Crawl Order. This can be set to either Bread First or Page Importance. It specifies the order in which pages will be crawled. In breadth first crawling, roughly all the seeds sites are visited first, followed by sites linked directly from seed sites, followed by sites linked directly from sites linked directly from seed sites, etc. Page Importance is our modification of [ [APC2003](#) ]. In this order, each seed sites starts with a certain quantity of money. When a site is crawled it distributes its money equally amongst sites it links to. When picking sites to crawl next, one chooses those that currently have the most money. Additional rules are added to handle things like the fact that some sites might have no outgoing links. Also, in our set-up we don't revisit already seen sites. To handle these situation we take a different tack from the original paper. This crawl order roughly approximates crawling according to page rank.



To Crawl Sites appears. If one checks Restricts Sites by Url then only pages on those sites and domains listed in the Allowed To Crawl Sites textarea can be crawled. We will say how to specify domains and sites in a moment, first let's discuss the last two textareas on the Options form. The Disallowed sites textarea allows you to specify sites that you do not want the crawler to crawl under any circumstance. There are many reasons you might not want a crawler to crawl a site. For instance, some sites might not have a good robots.txt file, but will ban you from interacting with their site if they get too much traffic from you.

Just above the Seed Sites textarea are two links "Add User Suggest Data". If on the Server Settings activity Account Registration is set to anything other than Disable Registration, it is possible for a search site user to suggest urls to crawl. This can be done by going to the [Search Tools Page](#) and clicking on the Suggest a Url link. Suggested links are stored in WORK\_DIRECTORY/data/suggest\_url.txt. Clicking Add User Suggest Data adds any suggested urls in this file into the Seed Site textarea, then deletes the contents of this file. The suggested urls which are not already in the seed site list are added after comment lines (lines starting with #) which give the time at which the urls were added. Adding Suggest data can be done either for new crawls or to inject urls into currently running crawls. The Seed sites textarea allows you to specify a list of urls that the crawl should start from. The crawl will begin using these urls. This list can include ".onion" urls if you want to crawl [TOR networks](#).

The format for sites, domains, and urls are the same for each of these textareas, except that the Seed site area can only take urls (or urls and title/descriptions) and in the Disallowed Sites/Sites with Quotas one can give a url followed by #. Otherwise, in this common format, there should be one site, url, or domain per line. You should not separate sites and domains with commas or other punctuation. White space is ignored. A domain can be specified as:

```
domain:.sjsu.edu
```

Urls like:

```
http://www.sjsu.edu/  
https://www.sjsu.edu/gape/  
http://bob.cs.sjsu.edu/index.html
```

would all fall under this domain. The word domain above is a slight misnomer as domain:sjsu.edu, without the leading period, also matches a site like http://mysjsu.edu/. A site can be specified as scheme://domain/path. Currently, Yioop recognizes the three schemas: http, https, and gopher (an older web protocol). For example, https://www.somewhere.com/foo/. Such a site includes https://www.somewhere.com/foo/anything\_more . Yioop also recognizes \* and \$ within urls. So http://my.site.com/\*/\*/ would match http://my.site.com/subdir1/subdir2/rest and http://my.site.com/\*/\*/\$ would require the last symbol in the url to be '/'. This kind of pattern matching can be useful in the to restrict the depth of a crawl to within a url to a certain fixed depth -- you can allow crawling a site, but disallow the downloading of pages with more than a certain number of '/' in them.

In the Disallowed Sites/Sites with Quotas, a number after a # sign indicates that at most that many pages should be downloaded from that site in any given hour. For example,

```
http://www.ucanbuyart.com/#100
```

indicates that at most 100 pages are to be downloaded from http://www.ucanbuyart.com/ per hour.

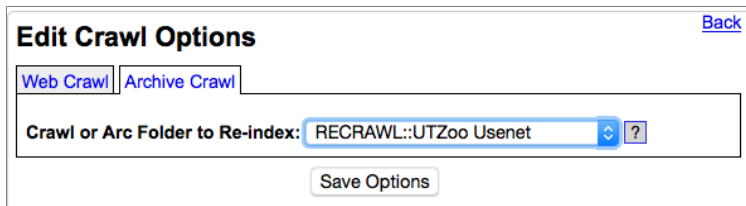
In the seed site area one can specify title and page descriptions for pages that Yioop would otherwise be forbidden to crawl by the robots.txt file. For example,

```
http://www.facebook.com/###!Facebook###!A%20famous%20social%20media%20site
```

tells Yioop to generate a placeholder page for http://www.facebook.com/ with title "Facebook" and description "A famous social media site" rather than to attempt to download the page. The [Results Editor](#) activity can only be used to affect pages which are in a Yioop index. This technique allows one to add arbitrary pages to the index.

When configuring a new instance of Yioop the file default\_crawl.ini is copied to WORK\_DIRECTORY/crawl.ini and contains the initial settings for the Options form.

We now consider how to do crawls of previously obtained archives. From the initial crawl options screen, clicking on the Archive Crawl tab gives one the following form:



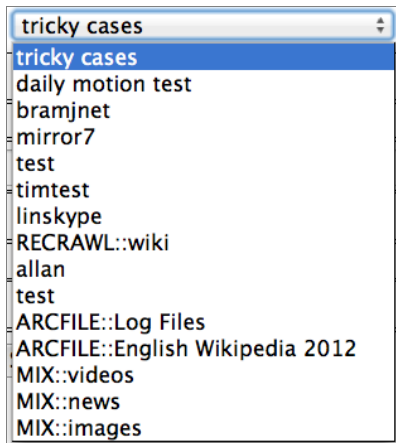
**Edit Crawl Options** [Back](#)

[Web Crawl](#) [Archive Crawl](#)

Crawl or Arc Folder to Re-index: RECRAWL::UTZoo Usenet ?

Save Options

The dropdown lists all previously done crawls that are available for recrawl.



tricky cases

tricky cases

daily motion test

bramjnet

mirror7

test

timtest

linskype

RECRAWL::wiki

allan

test

ARCFILE::Log Files

ARCFILE::English Wikipedia 2012

MIX::videos

MIX::news

MIX::images

These include both previously done Yioop crawls, previously done recrawls (prefixed with RECRAWL::), Yioop Crawl Mixes (prefixed with MIX::), and crawls of other file formats such as: arc, warc, database data, MediaWiki XML, and ODP RDF, which have been appropriately prepared in the PROFILE\_DIR/cache folder (prefixed with ARCFILE::). In addition, Yioop also has a generic text file archive importer (also, prefixed with ARCFILE::).

You might want to re-crawl an existing Yioop crawl if you want to add new meta-words, new cache page links, extract fields in a different manner, or if you are migrating a crawl from an older version of Yioop for which the index isn't readable by your newer version of Yioop. For similar reasons, you might want to recrawl a previously re-crawled crawl. When you archive crawl a crawl mix, Yioop does a search on the keyword site: any using the crawl mix in question. The results are then indexed into a new archive. This new archive might have considerably better query performance (in terms of speed) as compared to queries performed on the original crawl mix. How to make a crawl mix is described in the [Crawl Mixes](#) section. You might want to do an archive crawl of other file formats if you want Yioop to be able to provide search results of their content. Once you have selected the archive you want to crawl, you can add meta words as discussed in the Crawl Time Tab Page Rule portion of the [Page Options](#) section. Afterwards, go back to the Create Crawl screen to start your crawl. As with a Web Crawl, for an archive crawl you need both the queue server running and a least one fetcher running to perform a crawl.

To re-crawl a previously created web archive that was made using several fetchers, each of the fetchers that was used in the creation process should be running. This is because the data used in the recrawl will come locally from the machine of that fetcher. For other kinds of archive crawls and mix crawls, which fetchers one uses, doesn't matter because archive crawl data comes through the name server. You might also notice that the number of pages in a web archive re-crawl is actually larger than the initial crawl. This can happen because during the initial crawl data was stored in the fetcher's archive bundle and a partial index of this data sent to appropriate queue servers but was not yet processed by these queue servers. So it was waiting in a schedules folder to be processed in the event the crawl was resumed.

need to create an `PROFILE_DIR/cache/archives` folder on the name server machine. Yioop checks subfolders of this for files with the name `arc_description.ini`. For example, to do a Wikimedia archive crawl, one could make a subfolder `PROFILE_DIR/cache/archives/my_wiki_media_files` and put in it a file `arc_description.ini` in the format to be discussed in a moment. In addition to the `arc_description.ini`, you would also put in this folder all the archive files (or links to them) that you would like to index. When indexing, Yioop will process each archive file in turn. Returning to the `arc_description.ini` file, `arc_description.ini`'s contents are used to give a description of the archive crawl that will be displayed in the archive dropdown as well as to specify the kind of archives the folder contains and how to extract it. An example `arc_description.ini` for MediaWiki dumps might look like:

```
arc_type = 'MediaWikiArchiveBundle';
description = 'English Wikipedia 2012';
```

In the Archive Crawl dropdown the description will appear with the prefix `ARCFILE::` and you can then select it as the source to crawl. Currently, the supported `arc_types` are: `ArcArchiveBundle`, `DatabaseBundle`, `MediaWikiArchiveBundle`, `OdpRdfArchiveBundle`, `TextArchiveBundle`, and `WarcArchiveBundle`. For the `ArcArchiveBundle`, `OdpRdfArchiveBundle`, `MediaWikiArchiveBundle`, `WarcArchiveBundle` `arc_types`, generally a two line `arc_description.ini` file like above suffices. We now describe how to import from the other kind of formats in a little more detail. In general, the `arc_description.ini` will tell Yioop how to get string items (in a associative array with a minimal amount of additional information) from the archive in question. Processing on these string items can then be controlled using Page Rules, described in the [Page Options](#) section.

An example `arc_description.ini` where the `arc_type` is `DatabaseBundle` might be:

```
arc_type = 'DatabaseBundle';
description = 'DB Records';
dbms = "mysql";
db_host = "localhost";
db_name = "MYGREATDB";
db_user = "someone";
db_password = "secret";
encoding = "UTF-8";
sql = "SELECT MYCOL1, MYCOL2 FROM MYTABLE1 M1, MYTABLE2 M2 WHERE M1.FOO=M2.BAR";
field_value_separator = '|';
column_separator = '##';
```

Here is a specific example that gets the rows out of the `TRANSLATION` table of Yioop where the database was stored in a Postgres DBMS. In the comments I indicate how to alter it for other DBMS's.

```
arc_type = 'DatabaseBundle';
description = 'DB Records';
;sqlite3 specific
;dbms="sqlite3";
;mysql specific
;dbms = "mysql";
;db_host = "localhost";
;db_user = "root";
;db_password = "";
dbms = "pdo";
;below is for postgres; similar if want db2 or oracle
db_host = "pgsql:host=localhost;port=5432;dbname=seek_quarry"
db_name = "seek_quarry";
db_user = "cpollett";
db_password = "";
encoding = "UTF-8";
sql = "SELECT * from TRANSLATION";
field_value_separator = '|';
column_separator = '##';
```

Possible values for `dbms` are `pdo`, `mysql`, `sqlite3`. If `pdo` is chosen, then `db_host` should be a PHP DSN specifying which DBMS driver to use. `db_name` is the name of the database you would like to connect to, `db_user` is the database username, `db_password` is the password for that user, and `encoding` is the character set of rows that the database query will return.

The `sql` variable is used to give a query whose result rows will be the items indexed by Yioop. Yioop indexes string "pages", so to make these rows into a string each column result will be made into a string: `field field_value_separator value`. Here `field` is the name of the column, `value` is the value for that column in the given result row. Columns are concatenated together separated by the value of `column_separator`. The resulting string is then sent to Yioop's `TextProcessor` page processor.

First, suppose we wanted to index access log file records that look like:

```
127.0.0.1 - - [21/Dec/2012:09:03:01 -0800] "POST /git/yioop2/ HTTP/1.1" 200 - \
 "-" "Mozilla/5.0 (compatible; YioopBot; \
 +http://localhost/git/yioop/bot.php)"
```

Here each record is delimited by a newline and the character encoding is UTF-8. The records are stored in files with the extension .log and these files are uncompressed. We then might use the following arc\_description.ini file:

```
arc_type = 'TextArchiveBundle';
description = 'Log Files';
compression = 'plain';
file_extension = 'log';
end_delimiter = "\n";
encoding = "UTF-8";
```

In addition to compression = 'plain', Yioop supports gzip and bzip2. The end\_delimiter is a regular expression indicating how to know when a record ends. To process a TextArchiveBundle Yioop needs either an end\_delimiter or a start\_delimiter (or both) to be specified. As another example, for a mail.log file with entries of the form:

```
From pollett@mathcs.sjsu.edu Wed Aug 7 10:59:04 2002 -0700
Date: Wed, 7 Aug 2002 10:59:04 -0700 (PDT)
From: Chris Pollett <pollett@mathcs.sjsu.edu>
X-Sender: pollett@eniac.cs.sjsu.edu
To: John Doe <johndoe@mail.com>
Subject: Re: a message
In-Reply-To: <5.1.0.14.0.20020723093456.00ac9c00@mail.com>
Message-ID: <Pine.GSO.4.05.10208071057420.9463-100000@eniac.cs.sjsu.edu>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
Status: 0
X-Status:
X-Keywords:
X-UID: 17
```

Hi John,

I got your mail.

Chris

The following might be used:

```
arc_type = 'TextArchiveBundle';
description = 'Mail Logs';
compression = 'plain';
file_extension = 'log';
start_delimiter = "\n\nFrom\s";
encoding = "ASCII";
```

Notice here we are splitting records using a start delimiter. Also, we have chosen ASCII as the character encoding. As a final example, we show how to import tar gzip files of Usenet records as found, in the [UTzoo Usenet Archive 1981-1991](#). Further discussion on how to process this collection is given in the [Page Options](#) section.

```
arc_type = 'TextArchiveBundle';
description = 'Utzoo Usenet Archive';
compression = 'gzip';
file_extension = 'tgz';
start_delimiter = "\0\0\0\0Path:";
end_delimiter = "\n\0\0\0\0";
encoding = "ASCII";
```

Notice in the above we set the compression to be gzip. Then we have Yioop act on the raw tar file. In tar files, content objects are separated by long paddings of null's. Usenet posts begin with Path, so to keep things simple we grab records which begin with a sequence of null's the Path and end with another sequence of null's.

As a final reminder for this section, remember that, in addition, to the arc\_description.ini file, the subfolder should also contain instances of the files in question that you would like to archive crawl. So for arc files, these would be files of extension .arc.gz; for MediaWiki, files of extension .xml.bz2; and for ODP-RDF, files of extension .rdf.u8.gz . Crawl Options of config.php or local\_config.php

There are a couple of flags which can be set in the config.php or in a local\_config.php file that affect web crawling which we now mention for completeness. As was mentioned before, when Yioop is crawling it makes use of Etag: and Expires: HTTP headers received during web page download to determine when a page can be recrawled. This assumes one has not completely turned off

off, one can add to a local\_config.php file the line:

```
define("USE_ETAG_EXPIRES", false);
```

Yioop can be run using the [Hip Hop Virtual Machine from FaceBook](#). This will tend to make Yioop run faster and use less memory than running it under the standard PHP interpreter. Hip Hop can be used on various Linux flavors and to some degree runs under OSX (the queue server and fetcher will run, but the web app doesn't). If you want to use the Hip Hop on Mac OSX, and if you install it via Homebrew, then you will need to set a force variable and set the path for Hip Hop in your local\_config.php file with lines like:

```
define('FORCE_HHVM', true);  
define('HHVM_PATH', '/usr/local/bin');
```

The above lines are only needed on OSX to run Hip Hop.

[Return to table of contents](#)

## Mixing Crawl Indexes

Once you have performed a few crawls with Yioop, you can use the Mix Crawls activity to create mixture of your crawls. This activity is available to users who have either Admin role or just the standard User role. This section describes how to create crawl mixes which are processed when a query comes in to Yioop. Once one has created such a crawl mix, an admin user can make a new index which consists of results of the crawl mix ("materialize it") by doing an archive crawl of the crawl mix. The [Archive Crawl Options](#) subsection has more details on how to do this latter operation. The main Mix Crawls activity looks like:

### Make a Crawl Mix ?

Mix Name:

#### Available Mixes

Row 0 to 4 of 4 Show

Name	Definition	Actions			
<b>CanMix</b> 1422154469 24 Jan 2015 18:54:29	#1[1.0 * (Default Crawl + K:canada) + 1.0 * (Can Crawl Test + K:)]	<a href="#">Share</a>	<a href="#">Edit</a>	<a href="#">Set as Index</a>	<a href="#">Delete</a>
<b>images</b> 2 31 Dec 1969 16:00:02	#1[1.0 * (Default Crawl + K:media:image site:doc)]	<a href="#">Share</a>	<a href="#">Edit</a>	<a href="#">Set as Index</a>	<a href="#">Delete</a>
<b>news</b> 4 31 Dec 1969 16:00:04	#1[1.0 * (Default Crawl + K:media:news)]	<a href="#">Share</a>	<a href="#">Edit</a>	<a href="#">Set as Index</a>	<a href="#">Delete</a>
<b>videos</b> 3	#1[1.0 * (Default Crawl + K:media:video site:doc)]	<a href="#">Share</a>	<a href="#">Edit</a>	<a href="#">Set as Index</a>	<a href="#">Delete</a>

The first form allows you to name and create a new crawl mixture. Clicking "Create" sends you to a second page where you can provide information about how the mixture should be built. Beneath the Create mix form is a table listing all the previously created crawl mixes. Above this listing, but below the Create form is a standard set of nav elements for selecting which mixes will be displayed in this table. A Crawl mix is "owned" by the user who creates that mix. The table only lists crawl mixes "owned" by the user. The first column has the name of the mix, the second column says how the mix is built out of component crawls, and the actions columns allows you to edit the mix, set it as the default index for Yioop search results, or delete the mix. You can also append "m:name+of+mix" or "mix:name+of+mix" to a query to use that quiz without having to set it as the index. When you create a new mix, and are logged in so Yioop knows the mix belongs to you, your mix will also show up on the Settings page. The "Share" column pops a link where you can share a crawl mix with a Yioop Group. This will post a message with a link to that group so that others can import your mix into their lists of mixes. Creating a new mix or editing an existing mix sends you to a second page:

## EDIT CRAWL MIX

Mix Name

### Mix Components [?](#)

[\[Add Search Result Fragment\]](#)

Add Crawls		Select Crawl	Results Shown	1	<a href="#">Remove</a>
Weight	Name	Keywords	Actions		
2	Default Crawl	media:text canada	<a href="#">Delete</a>		
1	Can Crawl Test		<a href="#">Delete</a>		

Add Crawls		Select Crawl	Results Shown	3	<a href="#">Remove</a>
Weight	Name	Keywords	Actions		
1	onion test		<a href="#">Delete</a>		

Add Crawls		Select Crawl	Results Shown	1	<a href="#">Remove</a>
Weight	Name	Keywords	Actions		
1	Ucanbuyart	ottawa	<a href="#">Delete</a>		

Using the "Back" link on this page will take you to the prior screen. The first text field on the edit page lets you rename your mix if you so desire. Beneath this is an "Add Groups" button. A group is a weighted list of crawls. If only one group were present, then search results would come from any crawl listed for this group. A given result's score would be the weighted sum of the scores of the crawls in the group it appears in. Search results are displayed in descending order according to this total score. If more than one group is present then the number of results field for that group determines how many of the displayed results should come from that group. For the Crawl Mix displayed above, there are three groups: The first group is used to display the first result, the second group is used to display the second result, the last group is used to display any remaining search results.

The UI for groups works as follows: The top row has three columns. To add new components to a group use the dropdown in the first column. The second column controls for how many results the particular crawl group should be used. Different groups results are presented in the order they appear in the crawl mix. The last group is always used to display any remaining results for a search. The delete group link in the third column can be used to delete a group. Beneath the first row of a group, there is one row for each crawl that belongs to the group. The first link for a crawl says how its scores should be weighted in the search results for that group. The second column is the name of the crawl. The third column is a space separated list of words to add to the query when obtaining results for that crawl. So for example, in the first group above, there are two indexes which will be unioned: Default Crawl with a weight of 1, and CanCrawl Test with a weight of 2. For the Default Crawl we inject two keywords media:text and Canada to the query we get from the user. media:text means we will get whatever results from this crawl that consisted of text rather than image pages. Keywords can be used to make a particular component of a crawl mix behave in a conditional many by using the "if:" meta word described in the search and user interface section. The last link in a crawl row allows you to delete a crawl from a crawl group. For changes on this page to take effect, the "Save" button beneath this dropdown must be clicked.

[Return to table of contents](#) .

## Classifying Web Pages

Sometimes searching for text that occurs within a page isn't enough to find what one is looking for. For example, the relevant set of documents may have many terms in common, with only a small subset showing up on any particular page, so that one would have to search for many disjoint terms

formulate an appropriate query. Or the relevant documents may share many key terms with irrelevant documents, making it difficult to formulate a query that fetches one but not the other. Under these circumstances (among others), it would be useful to have meta words already associated with the relevant documents, so that one could just search for the meta word. The Classifiers activity provides a way to train classifiers that recognize classes of documents; these classifiers can then be used during a crawl to add appropriate meta words to pages determined to belong to one or more classes.

Clicking on the Classifiers activity displays a text field where you can create a new classifier, and a table of existing classifiers, where each row corresponds to a classifier and provides some statistics and action links. A classifier is identified by its class label, which is also used to form the meta word that will be attached to documents. Each classifier can only be trained to recognize instances of a single target class, so the class label should be a short description of that class, containing only alphanumeric characters and underscores (e.g., "spam", "homepage", or "menu"). Typing a new class label into the text box and hitting the Create button initializes a new classifier, which will then show up in the table.

### Manage Classifiers

Name:

### Available Classifiers

Row 0 to 2 of 2 Show  [\[Search\]](#)

Label	Positive	Negative	Actions		
<b>notspam</b> 06 Aug 2013 16:48:42	28	4	<a href="#">Edit</a>	<a href="#">Finalize</a>	<a href="#">Delete</a>
<b>homepage</b> 15 Jun 2014 17:40:15	0	0	<a href="#">Edit</a>	<a href="#">Finalize</a>	<a href="#">Delete</a>

Once you have a fresh classifier, the natural thing to do is edit it by clicking on the Edit action link. If you made a mistake, however, or no longer want a classifier for some reason, then you can click on the Delete action link to delete it; this cannot be undone. The Finalize action link is used to prepare a classifier to classify new web pages, which cannot be done until you've added some training examples. We'll discuss how to add new examples next, then return to the Finalize link.

### Editing a Classifier

Clicking on the Edit action link takes you to a new page where you can change a classifier's class label, view some statistics, and provide examples of positive and negative instances of the target class. The first two options should be self-explanatory, but the last is somewhat involved. A classifier needs labeled training examples in order to learn to recognize instances of a particular class, and you help provide these by picking out example pages from previous crawls and telling the classification system whether they belong to the class or do not belong to the class. The Add Examples section of the Edit Classifier page lets you select an existing crawl to draw potential examples from, and optionally narrow down the examples to those that satisfy a query. Once you've done this, clicking the Load button will send a request to the server to load some pages from the crawl and choose the next one to receive a label. You'll be presented with a record representing the selected document, similar to a search result, with several action links along the side that let you mark this document as either a positive or negative example of the target class, or skip this document and move on to the next one:

## Edit Classifier

Classifier Label:

### Statistics

Positive Examples: 8  
Negative Examples: 18  
Accuracy: 75.0% [\[Update\]](#)

### Add Examples

Source:

Keywords:

22 documents

<a href="#">[In Class]</a>	<a href="#">Chris Pollett &gt; Students &gt; Reddy Nithin</a>
<a href="#">[Not In Class]</a>	<a href="http://mirror.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring06/nithin/indexd399.shtml?CS298Proposal.html">http://mirror.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring06/nithin/indexd399.shtml?CS298Proposal.html</a>
<a href="#">[Skip]</a>	<b>Prediction: not 1homepage</b> (100.0% confidence, 61.1% disagreement) Chris Pollett Committee Members: Professor Chris <b>Tseng</b>
<a href="#">[In Class]</a>	<a href="#">Chris Pollett &gt; Students &gt; Kukreti Akshat</a>
<a href="#">[Not In Class]</a>	<a href="http://mirror.cs.sjsu.edu/faculty/pollett/masters/Semesters/Fall12/akshat/index4979.shtml?CS298%20Proposal.html">http://mirror.cs.sjsu.edu/faculty/pollett/masters/Semesters/Fall12/akshat/index4979.shtml?CS298 Proposal.html</a>
<a href="#">[Skip]</a>	<b>Prediction: not 1homepage</b> (100.0% confidence, 61.1% disagreement) Advisor: Dr. Chris Pollett Committee Members: Dr. Chris <b>Tseng</b> .
<a href="#">[In Class]</a>	<a href="#">Dr. Tseng - index</a>
<a href="#">[Not In Class]</a>	<a href="http://mirror.cs.sjsu.edu/_tseng/index.html">http://mirror.cs.sjsu.edu/_tseng/index.html</a>
<a href="#">[Skip]</a>	<b>Prediction: not 1homepage</b> (38.5% confidence, 61.1% disagreement) <a href="http://mirror.cs.sjsu.edu/_tseng/index.html">http://mirror.cs.sjsu.edu/_tseng/index.html</a> => <a href="http://professortseng.sjsu-cs.org/">http://professortseng.sjsu-cs.org/</a>

When you select any of the action buttons, your choice is sent back to the server, and a new example to label is sent back (so long as there are more examples in the selected index). The old example record is shifted down the page and its background color updated to reflect your decision—green for a positive example, red for a negative one, and gray for a skip; the statistics at the top of the page are updated accordingly. The new example record replaces the old one, and the process repeats. Each time a new label is sent to the server, it is added to the training set that will ultimately be used to prepare the classifier to classify new web pages during a crawl. Each time you label a set number of new examples (10 by default), the classifier will also estimate its current accuracy by splitting the current training set into training and testing portions, training a simple classifier on the training portion, and testing on the remainder (checking the classifier output against the known labels). The new estimated accuracy, calculated as the proportion of the test pages classified correctly, is displayed under the Statistics section. You can also manually request an updated accuracy estimate by clicking the Update action link next to the Accuracy field. Doing this will send a request to the server that will initiate the same process described previously, and after a delay, display the new estimate.

All of this happens without reloading the page, so avoid using the web browser's Back button. If you do end up reloading the page somehow, then the current example record and the list of previously-labeled examples will be gone, but none of your progress toward building the training set will be lost.

### Finalizing a Classifier

Editing a classifier adds new labeled examples to the training set, providing the classifier with a more complete picture of the kinds of documents it can expect to see in the future. In order to take advantage of an expanded training set, though, you need to finalize the classifier. This is broken out into a separate step because it involves optimizing a function over the entire training set, which can be slow for even a few hundred example documents. It wouldn't be practical to wait for the classifier to re-train each time you add a new example, so you have to explicitly tell the classifier that you're done adding examples for now by clicking on the Finalize action link either next to the Load button on the edit classifier page or next to the given classifier's name on the classifier management page.

Clicking this link will kick off a separate process that trains the classifier in the background. When the page reloads, the Finalize link should have changed to text that reads "Finalizing..." (but if the training set is very small, training may complete almost immediately). After starting finalization, it's fine to walk away for a bit, reload the page, or carry out some unrelated task for the user account.



that makes use of the classifier. When the classifier finishes its training phase, the Finalizing message will be replaced by one that reads "Finalized" indicating that the classifier is ready for use.

### **Using a Classifier**

Using a classifier is as simple as checking the "Use to Classify" or "Use to Rank" checkboxes next to the classifier's label on the [Page Options](#) activity, under the "Classifiers and Rankers" heading. When the next crawl starts, the classifier (and any other selected classifiers) will be applied to each fetched page. If "Use to Rank" is checked then the classifier score for that page will be recorded. If "Use to Classify" is checked and if a page is determined to belong to a target class, it will have several meta words added. As an example, if the target class is "spam", and a page is determined to belong to the class with probability .79, then the page will have the following meta words added:

- class:spam
- class:spam:50plus
- class:spam:60plus
- class:spam:70plus
- class:spam:70

These meta words allow one to search for all pages classified as spam at any probability over the preset threshold of .50 (with class:spam), at any probability over a specific multiple of .1 (e.g., over .6 with class:spam:60plus), or within a specific range (e.g., .60–.69 with class:spam:60). Note that no meta words are added if the probability falls below the threshold, so no page will ever have the meta words class:spam:10plus, class:spam:20plus, class:spam:20, and so on.

[Return to table of contents](#) .

### **Page Indexing and Search Options**

Several properties about how web pages are indexed and how pages are looked up at search time can be controlled by clicking on Page Options. There are three tabs for this activity: Crawl Time, Search Time, and Test Options. We will discuss each of these in turn.

#### **Crawl Time Tab**

Clicking on Page Options leads to the default Crawl Time Tab:

Get Page Options From: Use options below

Byte Range to Download (0 - Value): 50000

Summarizer: Centroid

Max Page Summary Length in Bytes: 2000

Cache whole crawled pages:

Allow Page Recrawl After: Never

Page File Types to Crawl:

unknown	cfm	java
bmp	cfml	jpg
doc	do	jpeg
csv	htm	pdf
tab	html	png
tsv	jsp	ppt
txt	php	pptx
docx	pl	rss
epub	py	rtf
asp	shtml	svg
aspx	gif	xlsx
cgi	xml	

Classifiers and Rankers

Row 0 to 3 of 3 Show 50

	Use to Classify	Use to Rank
baba	<input type="checkbox"/>	<input type="checkbox"/>
lala	<input type="checkbox"/>	<input type="checkbox"/>
notspam	<input type="checkbox"/>	<input type="checkbox"/>

Indexing Plugins

Plugin	Use in Crawl
AddressesPlugin	<input type="checkbox"/>
RecipePlugin	<input type="checkbox"/>
WordfilterPlugin	<input checked="" type="checkbox"/> <a href="#">[Configure]</a>

Page Field Extraction Rules

Save

This tab controls some aspects about how a page is processed and indexed at crawl time. The form elements before Page Field Extraction Rules are relatively straightforward and we will discuss these briefly below. The Page Rules textarea allows you to specify additional commands for how you would like text to be extracted from a page document summary. The description of this language will take the remainder of this subsection.

The Get Options From dropdown allows one to load in crawl time options that were used in a previous crawl. Beneath this, The Byte Range to Download dropdown controls how many bytes out of any given web page should be downloaded. Smaller numbers reduce the requirements on disk space needed for a crawl; bigger numbers would tend to improve the search results. If whole pages are being cached, these downloaded bytes are stored in archives with the fetcher. The Summarizer dropdown control what summarizer is used on a page during page processing. Yioop uses a summarizer to control what portions of a page will be put into the index and are available at search time for snippets. The two available summarizers are Basic, which picks the pages meta title, meta description, h1 tags, etc in a fixed order until the summary size is reached; and Centroid, which computes an "average sentence" for the document and adds phrases from the actual document according to nearness to this average. If Centroid summarizer is used Yioop also generates a word cloud for each document. Centroid tends to produces slightly better results than Basic but is slower. How to tweak the Centroid summarizer for a particular locale, is described in the [Localizing Yioop](#) section. The Max Page Summary Length in Bytes controls how many of the total bytes can be used to make a page summary which is sent to the queue server. It is only words in this summary which can actually be looked up in search result. Care should be taken in making this value larger as it can increase the both the RAM memory requirements (you might have to change the memory\_limit variable at the start of QueueServer.php to prevent crashing) while crawling and it can slow the crawl

both the whole web page downloaded as well as the summary extracted from the web page (checked) or just to keep the page summary (unchecked). The next dropdown, Allow Page Recrawl After, controls how many days that Yioop keeps track of all the URLs that it has downloaded from. For instance, if one sets this dropdown to 7, then after seven days Yioop will clear its Bloom Filter files used to store which urls have been downloaded, and it would be allowed to recrawl these urls again if they happened in links. It should be noted that all of the information from before the seven days will still be in the index, just that now Yioop will be able to recrawl pages that it had previously crawled. Besides letting Yioop get a fresher version of page it already has, this also has the benefit of speeding up longer crawls as Yioop doesn't need to check as many Bloom filter files. In particular, it might just use one and keep it in memory.

The Page File Types to Crawl checkboxes allow you to decide which file extensions you want Yioop to download during a crawl. This check is done before any download is attempted, so Yioop at that point can only guess the [MIME Type](#), as it hasn't received this information from the server yet. An example of a url with a file extension is:

```
http://flickr.com/humans.txt
```

which has the extension txt. So if txt is unchecked, then Yioop won't try to download this page even though Yioop can process plain text files. A url like:

```
http://flickr.com/
```

has no file extension and will be assumed to be have a html extension. To crawl sites which have a file extension, but no one in the above list check the unknown checkbox in the upper left of this list.

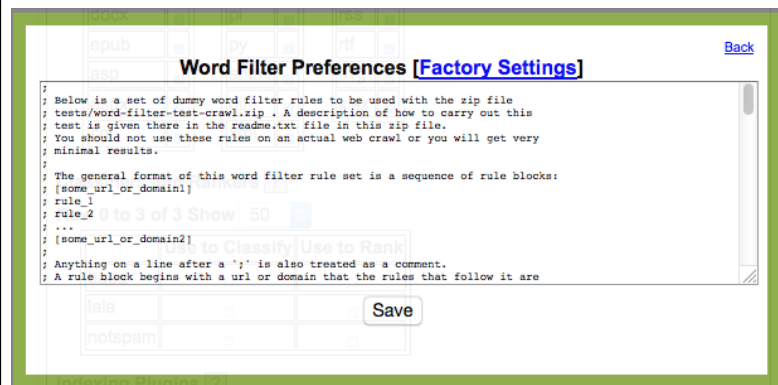
The Classifiers and Rankers checkboxes allow you to select the classifiers that will be used to classify or rank pages. Each classifier (see the [Classifiers](#) section for details) is represented in the list by its class label and two checkboxes. Checking the box under Use to classify indicates that the associated classifier should be used (made active) during the next crawl for classifying, checking the "Use to Rank" indicates that the classifier should be be used (made active) and its score for the document should be stored so that it can be used as part of the search time score. Each active classifier is run on each page downloaded during a crawl. If "Use to Crawl" was checked and the page is determined to belong to the class that the classifier has been trained to recognize, then a meta word like "class:label", where label is the class label, is added to the page summary. For faster access to pages that contain a single term and a label, for example, pages that contain "rich" and are labeled as "non-spam", Yioop actually uses the first character of the label "non-spam" and embeds it as part of the term ID of "rich" on "non-spam" pages with the word "rich". To ensure this speed-up can be used it is useful to make sure ones classifier labels begin with different first characters. If "Use to Rank" is checked then when a classifier is run on the page, the score from the classifier is recorded. When a search is done that might retrieve this page, this score is then used as one component of the overall score that this page receives for the query.

The Indexing Plugins checkboxes allow you to select which plugins to use during the crawl. Yioop comes with three built-in plugins: AddressesPlugin, RecipePlugin, and WordFilterPlugin. One can also write or downlaod additional plugins. If the plugin can be configured, next to the checkbox will be a link to a configuration screen. Let's briefly look at each of these plugins in turn...

Checking the AddressesPlugin enables Yioop during a crawl to try to calculate addresses for each page summary it creates. When Yioop processes a page it by default creates a summary of the page with a TITLE and a DESCRIPTION as well as a few other fields. With the addresses plugin activated, it will try to extract data to three additional fields: EMAILS, PHONE\_NUMBERS, and ADDRESSES. If you want to test out how these behave, pick some web page, view source on the web page, copy the source, and then paste into the Test Options Tab on the page options page (the Test Options Tab is described later in this section).

Clicking the RecipePlugin checkbox causes Yioop during a crawl to run the code in `indexing_plugins/recipe_plugin.php`. This code tries to detect pages which are food recipes and separately extracts these recipes and clusters them by ingredient. It then add search meta words `ingredient:` and `recipe:all` to allow one to search recipes by ingredient or only documents containing

Checking the WordFilterPlugin causes Yioop to run code in indexing\_plugins/wordfilter\_plugin.php on each downloaded page. The [Niche Crawling Video Tutorial](#) has information about how to use this plugin to create subject-specific crawls of the web. This code checks if the downloaded page has one of the words listed in the textarea one finds on the plugin's configure page. If it does, then the plugin follows the actions listed for pages that contain that term. Below is an example WordFilterPlugin configure page:



Lines in this configure file either specify a url or domain using a syntax like [url\_or\_domain] or specify a rule or a comment. Whitespace is ignored and everything after a semi-colon on a line is treated as a comment. The rules immediately following a url or domain line up till the next url or domain line are in effect if one crawling is crawling a page with that url or domain. Each **rule line** in the textarea consists of a comma separated list of literals followed by a colon followed by a comma separated list of what to do if the literal condition is satisfied. A single literal in the list of literals is an optional + or - followed by a sequence of non-space characters. After the + or -, up until a ; symbol is called the term in the literal. If the literal sign is + or if no sign is present, then the literal holds for a document if it contains the term, if the literal sign is - then the literal holds for a document if it does not contain the term, if there is a decimal number between 0 and 1, say x, after the # up to a comma or the first white-space character, then this is modified so the literal holds only if x'th fraction of the documents length comes from the literal's term. If rather than a decimal x were a positive natural number then the term would need to occur x times. If all the literal in the comma separated list hold, then the rule is said to hold, and the actions will apply. The line -term0:JUSTFOLLOW says that if the downloaded page does not contain the word "term0" then do not index the page, but do follow outgoing links from the page. The line term1:NOPROCESS says if the document has the word "term1" then do not index it or follow links from it. The last line +term2:NOFOLLOW,NOSNIPPET says if the page contains "term2" then do not follow any outgoing links. NOSNIPPET means that if the page is returned from search results, the link to the page should not have a snippet of text from that page beneath it. As an example of a more complicated rule, consider:

```
surfboard#2,bikini#0.02:NOINDEX, NOFOLLOW
```

Here for the rule to hold the condition surfboard#2 requires that the term surfboard occurred at least twice in the document and the condition bikini#0.02 requires that 0.02 percent of the documents total length also come from copies of the word bikini. In addition, to the commands just mentioned, WordFilterPlugin supports standard robots.txt directives such as: NOINDEX, NOCACHE, NOARCHIVE, NOODP, NOYDIR, and NONE. More details about how indexing plugins work and how to write your own indexing plugin can be found in the [Modifying Yioop](#) section.

### Page Field Extraction Language

We now return to the Page Field Extraction Rules textarea of the Page Options - Crawl Time tab. Commands in this area allow a user to control what data is extracted from a summary of a page. The textarea allows you to do things like modify the summary, title, and other fields extracted from a page summary; extract new meta words from a summary; and add links which will appear when a cache of a page is shown. Page Rules are especially useful for extracting data from generic text archives and database archives. How to import such archives is described in the Archive Crawls sub-section

that results from Yioop doing initial processing on a page. To see what this array looks like one can take a web page and paste it into the form on the Test Options tab. There are two types of page rule statements that a user can define: command statements and assignment statements. In addition, a semicolon ';' can be used to indicate the rest of a line is a comment. Although the initial textarea for rules might appear small. Most modern browsers allow one to resize this area by dragging on the lower right hand corner of the area. This makes it relatively easy to see large sets of rules.

A command statement takes a key field argument for the page associative array and does a function call to manipulate that page. Below is a list of currently supported commands followed by comments on what they do:

```
addMetaWords(field)      ;add the field and field value to the META_WORD
                          ;array for the page
addKeywordLink(field)    ;split the field on a comma, view this as a search
                          ;keywords => link text association, and add this to
                          ;the KEYWORD_LINKS array.
setStack(field)          ;set which field value should be used as a stack
pushStack(field)         ;add the field value for field to the top of stack
popStack(field)          ;pop the top of the stack into the field value for
                          ;field
setOutputFolder(dir)     ;if auxiliary output, rather than just to the
                          ;a yioop index, is being done, then set the folder
                          ;for this output to be dir
setOutputFormat(format)  ;set the format of auxiliary output.
                          ;Should be either CSV or SQL
                          ;SQL mean that writeOutput will write an insert
                          ;statement
setOutputTable(table)    ;if output is SQL then what table to use for the
                          ;insert statements
toArray(field)           ;splits field value for field on a comma and
                          ;assign field value to be the resulting array
toString(field)          ;if field value is an array then implode that
                          ;array using comma and store the result in field
                          ;value
unset(field)             ;unset that field value
writeOutput(field)       ;use the contents of field value viewed as an array
                          ;to fill in the columns of a SQL insert statement
                          ;or CSV row
```

Page rule assignments can either be straight assignments with '=' or concatenation assignments with '.='. Let \$page indicate the associative array that Yioop supplies the page rule processor. There are four kinds of values that one can assign:

```
field = some_other_field ; sets $page['field'] = $page['some_other_field']
field = "some_string"    ; sets $page['field'] to "some string"
field = /some_regex/replacement_where_dollar_vars_allowed/
                          ; computes the results of replacing matches to some_regex
                          ; in $page['field'] with replacement_where_dollar_vars_allowed
field = /some_regex/g ;sets $page['field'] to the array of all matches
                          ; of some regex in $page['field']
```

For each of the above assignments we could have used ".=" instead of "=". We next give a simple example and followed by a couple more complicated examples of page rules and the context in which they were used:

In the first example, we just want to extract meaningful titles for mail log records that were read in using a TextArchiveBundleIterator. Here after initial page processing a whole email would end up in the DESCRIPTION field of the \$page associative array given tot the page rule processor. So we use the following two rules:

```
TITLE = DESCRIPTION
TITLE = /(.\|\\n|\\Z)*?Subject:[\t ](.*?)\n(.\|\\n|\\Z)*/$2/
```

We initially set the TITLE to be the whole record, then use a regex to extract out the correct portion of the subject line. Between the first two slashes recognizes the whole record where the pattern inside the second pair of parentheses (.\*?) matches the subject text. The \$2 after the second parenthesis says replace the value of TITLE with just this portion.

The next example was used to do a quick first pass processing of record from the [UTzoo Archive of Usenet Posts from 1981-1991](#). What each block does is described in the comments below

```
;
; Set the UI_FLAGS variable. This variable in a summary controls
; which of the header elements should appear on cache pages.
; UI_FLAGS should be set to a string with a comma separated list
; of the options one wants. In this case, we use: yioop_nav, says that
; we do want to display header; version, says that we want to display
; when a cache item was crawled by Yioop; and summaries, says to display
```

```

; dropdown to other cached versions of item; highlight, whether search
; keywords should be highlighted in cached items
;
UI_FLAGS = "yioop_nav,version,summaries"
;
; Use Post Subject line for title
;
TITLE = DESCRIPTION
TITLE = /(.|\n)*?Subject:([\n+]\n(|\n)*)/$2/
;
; Add a link with a blank keyword search so cache pages have
; link back to yioop
;
link_yioop = ",Yioop"
addKeywordLink(link_yioop)
unset(link_yioop) ;using unset so don't have link_yioop in final summary
;
; Extract y-M and y-M-j dates as meta word u:date:y-M and u:date:y-M-j
;
date = DESCRIPTION
date = /(.|\n)*?Date:([\n+]\n(|\n)*)/$2/
date = /.*,\s*(\d*)-(\w*)-(\d*)\s*.*$/3-$2-$1/
addMetaWord(date)
date = /(\d*)-(\w*)-.*$/1-$2/
addMetaWord(date)
;
; Add a link to articles containing u:date:y-M meta word. The link text
; is Date:y-M
;
link_date = "u:date:"
link_date .= date
link_date .= ",Date:"
link_date .= date
addKeywordLink(link_date)
;
; Add u:date:y meta-word
;
date = /(\d*)-.*$/1/
addMetaWord(date)
;
; Get the first three words of subject ignoring re: separated by underscores
;
subject = TITLE
subject = /(\s*(RE:|re:|rE:|Re:)\s*)?(.*)/$3/
subject_word1 = subject
subject_word1 = /\s*([\s]*).*$/1/
subject_word2 = subject
subject_word2 = /\s*([\s]*)\s*([\s]*).*$/2/
subject_word3 = subject
subject_word3 = /\s*([\s]*)\s*([\s]*)\s*([\s]*).*$/3/
subject = subject_word1
unset(subject_word1)
subject .= " "
subject .= subject_word2
unset(subject_word2)
subject .= " "
subject .= subject_word3
unset(subject_word3)
;
; Get the first newsgroup listed in the Newsgroup: line, add a meta-word
; u:newsgroup:this-newgroup. Add a link to cache page for a search
; on this meta word
;
newsgroups = DESCRIPTION
newsgroups = /(.|\n)*?Newsgroups:([\n+]\n(|\n)*)/$2/
newsgroups = /\s*(\w|\.)+.*$/1/
addMetaWord(newsgroups)
link_news = "u:newsgroups:"
link_news .= newsgroups
link_news .= ",Newsgroup: "
link_news .= newsgroups
addKeywordLink(link_news)
unset(link_news)
;
; Makes a thread meta u:thread:newsgroup-three-words-from-subject.
; Adds a link to cache page to search on this meta word
;
thread = newsgroups
thread .= ":"
thread .= subject
addMetaWord(thread)
unset(newsgroups)
link_thread = "u:thread:"
link_thread .= thread
link_thread .= ",Current Thread"
addKeywordLink(link_thread)
unset(subject)
unset(thread)
unset(link_thread)

```

As a last example of page rules, suppose we wanted to crawl the web and whenever we detected a page had an address we wanted to write that address as a SQL insert statement to a series of text files. We can do this using page rules and the AddressesPlugin. First, we would check the AddressesPlugin and then we might use page rules like:

```

pushStack(DESCRIPTION)
pushStack(TITLE)
setOutputFolder(/Applications/MAMP/htdocs/crawls/data)
setOutputFormat(sql)
setOutputTable(SUMMARY);
writeOutput(summary)

```

The first line says copy the contents of the ADDRESSES field of the page into a new summary field. The next line says use the summary field as the current stack. At this point the stack would be an array with all the addresses found on the given page. So you could use the command like popStack(first\_address) to copy the first address in this array over to a new variable first\_address. In the above case what we do instead is push the contents of the DESCRIPTION field onto the top of the stack. Then we push the contents of the TITLE field. The line

```
setOutputFolder(/Applications/MAMP/htdocs/crawls/data)
```

sets /Applications/MAMP/htdocs/crawls/data as the folder that any auxiliary output from the page\_processor should go to. setOutputFormat(sql) says we want to output sql, the other possibility is csv. The line setOutputTable(SUMMARY); says the table name to use for INSERT statements should be called SUMMARY. Finally, the line writeOutput(summary) would use the contents of the array entries of the summary field as the column values for an INSERT statement into the SUMMARY table. This writes a line to the file data.txt in /Applications/MAMP/htdocs/crawls/data. If data.txt exceeds 10MB, it is compressed into a file data.txt.0.gz and a new data.txt file is started.

## Search Time Tab

The Page Options Search Time tab looks like:

The screenshot shows the 'Search Time' tab selected. It contains the following controls:

- Search Page Elements and Links**: A grid of checkboxes for 'Word Suggest', 'Subsearch', 'Signin', 'Cache', 'Similar', 'Inlinks', and 'IP Address', all of which are checked.
- Search Ranking Factors**: Three input fields for 'Title Weight' (value: 2), 'Description Weight' (value: 1), and 'Link Weight' (value: 1).
- Search Results Grouping**: Two input fields for 'Minimum Results to Group' (value: 200) and 'Server Alpha' (value: 1.6).

A 'Save' button is located at the bottom center of the form.

The Search Page Elements and Links control group is used to tell which element and links you would like to have presented on the search landing and search results pages. The Word Suggest checkbox controls whether a dropdown of word suggestions should be presented by Yioop when a user starts typing in the Search box. It also controls whether spelling correction and thesaurus suggestions will appear. The Subsearch checkbox controls whether the links for Image, Video, and News search appear in the top bar of Yioop. You can actually configure what these links are in the [Search Sources](#) activity. The checkbox here is a global setting for displaying them or not. In addition, if this is unchecked then the hourly activity of downloading any RSS media sources for the News subsearch will be turned off. The Signin checkbox controls whether to display the link to the page for users to sign in to Yioop. The Cache checkbox toggles whether a link to the cache of a search item should be displayed as part of each search result. The Similar checkbox toggles whether a link to similar search items should be displayed as part of each search result. The Inlinks checkbox toggles whether a link for inlinks to a search item should be displayed as part of each search result. Finally, the IP address checkbox toggles whether a link for pages with the same ip address should be displayed as part of each search result.

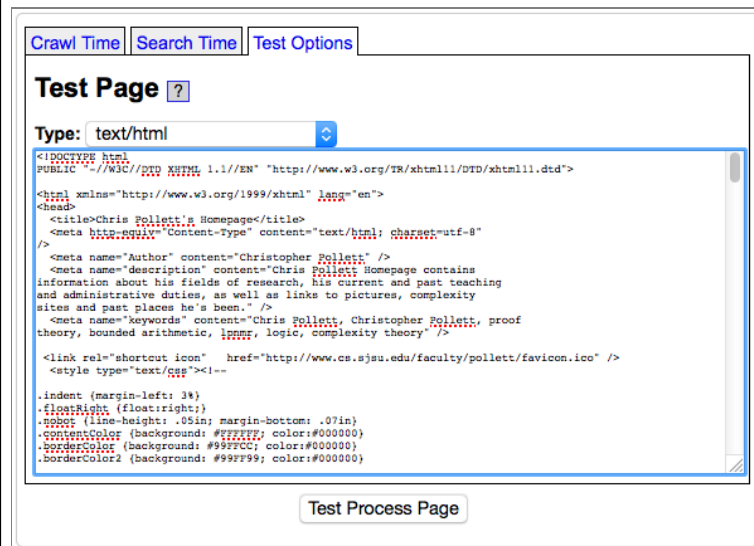
are used by Yioop to decide how to weigh each portion of a document when it returns query results to you.

When Yioop ranks search results it search out in its postings list until it finds a certain number of qualifying documents. It then sorts these by their score, returning usually the top 10 results. In a multi-queue-server setting the query is simultaneously asked by the name server machine of each of the queue server machines and the results are aggregated. The Search Results Grouping controls allow you to affect this behavior. Minimum Results to Group controls the number of results the name server want to have before sorting of results is done. When the name server request documents from each queue server, it requests for  $\alpha \times (\text{Minimum Results to Group}) / (\text{Number of Queue Servers})$  documents. Server Alpha controls the number alpha.

The Save button of course saves any changes you make on this form.

## Test Options Tab

The Page Options Test Options tab looks like:



The screenshot shows the 'Test Options' tab in the Yioop interface. It features a 'Test Page' section with a 'Type' dropdown menu set to 'text/html'. Below this is a large text area containing HTML code for a test page. The code includes a DOCTYPE declaration, a PUBLIC ID, an XML declaration, and various HTML elements such as <html>, <head>, <title>, <meta>, <link>, and <style>. At the bottom of the text area is a 'Test Process Page' button.

In the Type dropdown one can select a [MIME Type](#) used to select the page processor Yioop uses to extract text from the data you type or paste into the textarea on this page. Test Options let's you see how Yioop would process a web page and add summary data to its index. After filling in the textarea with a page, clicking Test Process Page will show the \$summary associative array Yioop would create from the page after the appropriate page processor is applied. Beneath it shows the \$summary array that would result after user-defined page rules from the crawl time tab are applied. Yioop stores a serialized form of this array in a IndexArchiveBundle for a crawl. Beneath this array is an array of terms (or character n-grams) that were extracted from the page together with their positions in the document. Finally, a list of meta words that the document has are listed. Either extracted terms or meta-word could be used to look up this document in a Yioop index.

## Results Editor

Sometimes after a large crawl one finds that there are some results that appear that one does not want in the crawl or that the summary for some result is lacking. The Result Editor activity allows one to fix these issues without having to do a completely new crawl. It has three main forms: An edited urls forms, a url editing form, and a filter websites form.

If one has already edited the summary for a url, then the dropdown in the edited urls form will list this url. One can select it and click load to get it to display in the url editing form. The purpose of the url editing form is to allow a user to change the title and description for a url that appears on a search results page. By filling out the three fields of the url editing form, or by loading values into them through the previous form and changing them, and then clicking save, updates the appearance of



the other two blank, and saves. This form does not affect whether the page is looked up for a given query, only its final appearance. It can only be used to edit the appearance of pages which appear in the index, not to add pages to the index. Also, the edit will affect the appearance of that page for all indexes managed by Yioop. If you know there is a page that won't be crawled by Yioop, but would like it to appear in an index, please look at the crawl options section of [Manage Crawls](#) documentation.

To understand the filter websites form, recall the disallowed sites crawl option allows a user to specify they don't want Yioop to crawl a given web site. After a crawl is done though one might be asked to removed a website from the crawl results, or one might want to remove a website from the crawl results because it has questionable content. A large crawl can take days to replace, to make the job of doing such filtering faster while one is waiting for a replacement crawl where the site has been disallowed, one can use a search filter.

The image shows two web forms. The top form is titled "Edit Result Page" and includes a dropdown menu for "Edited Urls" with a "Load" button, input fields for "URL:" and "Title:", and a large "Description" text area. Below the description are "Reset" and "Save Page" buttons. The bottom form is titled "Filter Websites from Results" and features a "Sites to Filter" section with a list of example URLs: `http://www.modularforde.com/`, `http://static.modularforde.com/`, and `http://www.searchresultsdirect.com/`. A "Save Filter" button is located at the bottom of this form.

Using the filter websites form one can specify a list of hosts which should be excluded from the search results. The sites listed in the Sites to Filter textarea are required to be hostnames. Using a filter, any web page with the same host name as one listed in the Sites to Filter will not appear in the search results. So for example, the filter settings in the example image above contain the line `http://www.cs.sjsu.edu/`, so given these settings, the web page `http://www.cs.sjsu.edu/faculty/pollett/` would not appear in search results.

[Return to table of contents](#) .

## Search Sources

The Search Sources activity is used to manage the media sources available to Yioop, and also to control the subsearch links displayed on the top navigation bar. The Search Sources activity looks like:

Type:

Name:

URL:

Language:

Provide xpaths to news feed components below:

Image XPath:

### Media Sources

Row 0 to 6 of 6 Show 50

Name	Type	Urls	Action
Break.com	Video	http://www.break.com/index/{} http://www.yioop.com/resources/blank.png?{}	<a href="#">Edit</a> <a href="#">Delete</a>
DailyMotion	Video	http://www.dailymotion.com/video/{} http://www.dailymotion.com/thumbnail/video/{}	<a href="#">Edit</a> <a href="#">Delete</a>
MetaCafe	Video	http://www.metacafe.com/watch/{} http://www.metacafe.com/thumb/{}.jpg	<a href="#">Edit</a> <a href="#">Delete</a>
Vimeo	Video	http://player.vimeo.com/video/{} http://www.yioop.com/resources/blank.png?{}	<a href="#">Edit</a> <a href="#">Delete</a>
Yahoo News	RSS	http://news.yahoo.com/rss/ ###/content/@uri	<a href="#">Edit</a> <a href="#">Delete</a>
YouTube	Video	http://www.youtube.com/watch?v={} http://i1.ytimg.com/vi/{}/default.jpg	<a href="#">Edit</a> <a href="#">Delete</a>

### Add a Subsearch

Folder Name:

Index Source:

Results Per Page:

### Current Subsearches

Row 0 to 3 of 3 Show 50

Folder Name	Index	Localization Identifier	Results/Page	Actions
images	images m:2	db_subsearch_images	50	<a href="#">Edit</a> <a href="#">Localize</a> <a href="#">Delete</a>
news	news m:4	db_subsearch_news	20	<a href="#">Edit</a> <a href="#">Localize</a> <a href="#">Delete</a>
videos	videos m:3	db_subsearch_videos	10	<a href="#">Edit</a> <a href="#">Localize</a> <a href="#">Delete</a>

The top form is used to add a media source to Yioop. Currently, the Media Kind can be either Video, RSS, or HTML. **Video Media** sources are used to help Yioop recognize links which are of videos on a web video site such as YouTube. This helps in both tagging such pages with the meta word `media:video` in a Yioop index and in being able to render a thumbnail of the video in the search results. When the media kind is set to video, this form has three fields: Name, which should be a short familiar name for the video site (for example, YouTube); URL, which should consist of a url pattern by which to recognize a video on that site; and Thumb, which consist of a url pattern to replace the original pattern by to find the thumbnail for that video. For example, the value of URL for YouTube is:

```
http://www.youtube.com/watch?v={}&
```

This will match any url which begins with `http://www.youtube.com/watch?v=` followed by some string followed by `&` followed by another string. The `{}` indicates that from `v=` to the `&` should be treated as the identifier for the video. The Thumb url in the case of YouTube is:

```
http://img.youtube.com/vi/{}/2.jpg
```

If the identifier in the first video link was `yv0zA9kN6L8`, then using the above, when displaying a thumb for the video, Yioop would use the image source:

```
http://img.youtube.com/vi/{yv0zA9kN6L8}/2.jpg
```

Some video sites have more complicated APIs for specifying thumbnails. In which case, you can still do `media:video` tagging but display a blank thumbnail rather than suggest a thumbnail link. To do this one uses the thumb url.

```
http://www.yioop.com/resources/blank.png?{}
```

If one selects the media kind to be **RSS** (really simple syndication, a kind of news feed, you can also use Atom feeds as sources), then the media sources form has four fields: **Name**, again a short familiar name for the RSS feed; **URL**, the url of the RSS feed, **Language**, what language the RSS feed is; and Image XPath, an optional field which allows you to specify and XPath relative to a RSS

item will display given the current language settings of Yioop. If under Manage Machines the Media Updater on the Name Server is turned on, then these RSS feeds will be downloaded hourly. If under the Search Time screen of the [Page Options](#) activity, the subsearch checkbox is checked, then there will be a link to News which appears on the top of the search page. Clicking on this link will display news items in order of recentness.

An **HTML Feed** is a web page that has news articles like an RSS page that you want the Media Updater to scrape on an hourly basis. To specify where in the HTML page the news items appear you specify different XPath information. For example,

```
Name: Cape Breton Post
URL: http://www.capebretonpost.com/News/Local-1968
Channel: //div[contains(@class, "channel")]
Item: //article
Title: //a
Description: //div[contains(@class, "dek")]
Link: //a
```

The Channel field is used to specify the tag that encloses all the news items. Relative to this as the root tag, //article says the path to an individual news item. Then relative to an individual news item, //a gets the title, etc. Link extracts the href attribute of that same //a .

Returning again to Image Xpath, which is a field of both the RSS form and the HTML Feed form. Not all RSS feeds use the same tag to specify the image associated with a news item. The Image XPath allows you to specify relative to a news item (either RSS or HTML) where an image thumbnail exists. If a site does not use such thumbnail one can prefix the path with ^ to give the path relative to the root of the whole file to where a thumb nail for the news source exists. Yioop automatically removes escaping from RSS containing escaped HTML when computing this. For example, the following works for the feed:

```
http://feeds.wired.com/wired/index
//description/div[contains(@class,
"rss_thumbnail")]img/@src
```

Beneath this media sources form is a table listing all the currently added media sources, their urls, and a links that allows one to edit or delete sources.

The second form on the page is the Add a Subsearch form. The form allows you to add a new specialized search link which may appear at the top the search page. If there are more that three of these subsearch are added or if one is seeing the page on a mobile platform, one instead gets a "More" link. This links to the tool.php page which then lists out all possible specialized search, some account links, and other useful Yioop tools. The Add a Subsearch form has three fields: Folder Name is a short familiar name for the subsearch, it will appear as part of the query string when the given subsearch is being performed. For example, if the folder names was news, then s=news will appear as part of the query string when a news subsearch is being done. Folder Name is also used to make the localization identifier used in translating the subsearch's name into different languages. This identifier will have the format db\_subsearch\_identifier. For example, db\_subsearch\_news. Index Source, the second form element, is used to specify a crawl or a crawl mix that the given subsearch should use in returning results. Results per Page, the last form element, controls the number of search results which should appear when using this kind of subsearch.

Beneath this form is a table listing all the currently added subsearches and their properties. The actions column at the end of this table let's one either edit, localize or delete a given subsearch. Clicking localize takes one to the Manage Locale's page for the default locale and that particular subsearch localization identifier, so that you can fill in a value for it. Remembering the name of this identifier, one can then in Manage Locales navigate to other locales, and fill in translations for them as well, if desired.

[Return to table of contents](#) .

## GUI for Managing Machines and Servers

Rather than use the command line as described in the [Prerequisites for Crawling](#) section, it is possible to start/stop and view the log files of queue servers and fetcher through the Manage

These processes are responsible for updating news feeds, and if run in a distributed mode, converting videos to mp4 format, and sending out bulk e-mail notifications about group activities. In order for Manage Machines to work, the additional requirements for this activity mentioned in the [Requirements](#) section must have been met. The Manage Machines activity looks like:

**Add Machine** ?

Machine Name:

Machine Url:

Is Mirror:

Has Queue Server:

Number of Fetchers: 0

**Machine Information** ?

Media Updater Mode: [Name Server] **Distributed**

**Name Server**

Media Updater [Log]  On  Off

Row 0 to 6 of 6 Show 50

**FG1** [http://10.1.10.12/search\_yioop/] [\[Delete\]](#)

<b>Queue Server</b>	#00 [Log] <input checked="" type="checkbox"/> On <input type="checkbox"/> Off
<b>Fetchers</b>	#00 [Log] <input checked="" type="checkbox"/> On <input type="checkbox"/> Off
	#01 [Log] <input checked="" type="checkbox"/> On <input type="checkbox"/> Off
	#02 [Log] <input checked="" type="checkbox"/> On <input type="checkbox"/> Off
	#03 [Log] <input type="checkbox"/> On <input checked="" type="checkbox"/> Off
	#04 [Log] <input type="checkbox"/> On <input checked="" type="checkbox"/> Off
	#05 [Log] <input type="checkbox"/> On <input checked="" type="checkbox"/> Off

**FG2** [http://10.1.10.10/search\_yioop/] [\[Delete\]](#)

<b>Queue Server</b>	#00 [Log] <input checked="" type="checkbox"/> On <input type="checkbox"/> Off
---------------------	---

The top of the page has a form for adding new machines, and beneath this is a list of existing machines. On Add machine form, the **Machine Name** field lets you give this machine an easy to remember name. The **Machine URL** field should be filled in with the URL to the installed Yioop instance. The **Mirror** checkbox says whether you want the given Yioop installation to act as a mirror for another Yioop installation. Checking it will reveal a Parent Name textfield that allows you to choose which installation amongst the previously entered machines names (not urls) you want to mirror. The **Has Queue Server** checkbox is used to say whether the given Yioop installation will be running a queue server or not. Finally, the **Number of Fetchers** dropdown allows you to say how many fetcher instances you want to be able to manage for that machine.

Beneath the Add machine form is the Machine Information listings. At the top of this is the **Media Updater Mode**. The MediaUpdater process is used to run the periodic MediaJob's that have been defined in the library/media\_jobs folders. These jobs will typically perform tasks like update news feeds, send bulk emails, and convert uploaded videos to mp4 format. In the above, this is set to Name Server. This means that there will be at most one running Media Updater process, and it will be on the name server. If you choose Distributed, there will be a Media Updater which can be turned on and off for each machine you add. Only if all the Media Updater's you have are on do updates to news, videos, etc happen properly, so the Name Server only set-up can be more convenient. On the other hand, the distributed set-up can handle updates for more news feeds, and the video upload conversion to mp4 only works in the distributed mode.

After the Media Updater Mode, the remainder of the page shows the currently configured machines. This list always begins with the Name Server itself and a toggle to control whether or not the Media Updater process is running on the Name Server. There is also a link to the log file of the Media

number of current machine statuses that are displayed for all other machines that have been added. It also might have next and previous arrow links to go through the currently available machines. To modify a machine that you have already added,

Beneath this dropdown is a set of boxes for each machine you have added to Yioop. In the far corner of this box is a link to Delete that machine from the list of known machines, if desired. Besides this, each box lists the queue server, if any, and each of the fetchers you requested to be able to manage on that machine. Next to these there is a link to the log file for that server/fetcher and below this there is an On/Off switch for starting and stopping the server/fetcher. This switch is green if the server/fetcher is running and red otherwise. A similar On/Off switch is present to turn on and off mirroring on a machine that is acting as a mirror. It is possible for a switch to be yellow if the machine is crashed but where it is possible that the machine might be automatically restarted by Yioop without your intervention.

## Building Sites with Yioop

### Building a Site using Yioop's Wiki System

As was mentioned in the [Appearance Activity](#) section of the documentation, background color, icons, title, and SEO meta information for a Yioop instance call all be configured from the Configure Activity. Customizing string which appear on Yioop pages can be done by editing the strings in [Manage Locales](#) . Adding advertisements such as banner and skyscraper ads or turning on user submitted keyword advertisements can be done using the form on the [Server Settings](#) activity. If you would like a site with a more custom landing page, then one can check **Use Wiki Public Main Page as Landing Page** under Toggle Advance Settings : Site Customizations. The Public Main page will then be the page you see when you first go to your site. You can then build out your site using the wiki system for the public group. Common headers and footers can be specified for pages on your site using each wiki page's Settings attributes. More advanced styling of pages can be done by specifying the auxiliary css data under Toggle Advance Settings. As Wiki pages can be set to be galleries, or slide presentations, and as Yioop supports including images, video, and embedding search bars and math equations on pages using its [Yioop's Wiki Syntax](#) , one can develop quite advanced sites using just this approach. The video tutorial [Building Websites Using Yioop](#) explains how the Seekquarry.com site was built using Yioop Software in this way.

### Building a Site using Yioop as Framework

For more advanced, dynamic websites than the wiki approach described above, the Yioop code base can still serve as the code base for new custom search web sites. The web-app portion of Yioop uses a [model-view-adapter \(MVA\) framework](#). This is a common, web-suitable variant on the more well-known Model View Controller design pattern. In this set-up, sub-classes of the Model class should handle file I/O and database function, sub-classes of Views should be responsible for rendering outputs, and sub-classes of the Controller class do calculations on data received from the web and from the models to give the views the data they finally need to render. In the remainder of this section we describe how this framework is implemented in Yioop and how to add code to the WORK\_DIRECTORY/app folder to customize things for your site. In this discussion we will use YIOOP\_DIR to refer to the Yioop folder, APP\_DIR to refer to WORK\_DIRECTORY/app and BASE\_DIR to refer to YIOOP\_DIR/src.

To understand Yioop's MVA architecture, we begin by tracing how a typical request is processed by Yioop. A typical web app request is handled by the file YIOOP\_DIR/index.php. It defines a constant seekquarry\yioop\configs\REDIRECTS\_ON as true and then loads the file BASE\_DIR/index.php whose bootstrap() method is then called. Since seekquarry\yioop\configs\REDIRECTS\_ON is true, bootstrap() will call a function configureRewrites() to see if the incoming URL should be rewritten. To check for rewrite, configureRewrites() makes use of an associative array \$route\_maps consisting of incoming\_route => routeHandlerFunctionName pairs. For example, the pair 'admin' => 'routeController' says that an incoming url of the form YIOOP\_LOCATION/admin should be rewritten by calling the function routeController. Before performing a rewrite lookup, configureRewrites() checks if a static method Routes::getRoutes() exists, and if so, tacks on the associate array this

getRoutes method in a Routes class in a file APP\_DIR/Routes.php. Typically, when one defines a handler function for a given route, one also defines a helper function, that let's one create urls for links which match the given pattern. For example, function controllerUrl(\$name, \$with\_delim = false) can be used to create urls which could be handled routeController. This function should check if REDIRECTS\_ON is true and output a url that works for Yioop if url rewriting is on and a different one if it is off.

Once url rewriting has been performed, bootstrap() tries to determine which controller should be used for the incoming request. To do this bootstrap() has an array \$available\_controllers which lists the controllers available to the script. The names of the controllers in this array are lowercase. Based on whether the \$\_REQUEST['c'] variable is in this array index.php either loads the class upperCaseFirstLetter(\$\_REQUEST['c'])Controller or loads whatever the default controller is. bootstrap() completes by instantiating this class and calling its processRequest() method. To add to the list of \$available\_controllers, a developer can create function localControllers in their BASE\_DIR/configs/LocalConfig.php which returns an array of additional controller names. Yioop's autoloader first check the directory APP\_DIR/controllers for the existence of a controller and if this fails looks in the folder BASE\_DIR/contollers. This means that new, developer-defined controller class can be added to APP\_DIR/controllers folder and the behavior of existing controllers can be altered by creating a new controller of the same name in APP\_DIR/controllers. A controller file should have in it a file which extends the class Controller. Controller files should always have names of the form SomenameController.php and the class inside them should be named SomenameController. Notice it is Somename rather than SomeName. These general naming conventions are used for models, views, etc. Any Controller subclass has methods component(\$name), model(\$name), view(\$name), and plugin(\$name). These methods load, instantiate, and return a class with the given name. For example, \$my\_controller->model("crawl"); checks to see if a CrawlModel has already been instantiated, if so, it returns it; if not, it tries to load it, first from the file APP\_DIR/models/CrawlModel.php, and if that fails, from BASE\_DIR/models /CrawlModel.php. Once loaded, it instantiates a CrawlModel, saves a reference to this object, and returns it.

As we indicated above, if a class needs to be loaded, Yioop first looks in APP\_DIR. For example, \$my\_controller->view("search") would first look for a file: APP\_DIR/views/SearchView.php to include, if it cannot find such a file then it tries to include BASE\_DIR/views/SearchView.php. So to change the behavior of an existing BASE\_DIR file one just has a modified copy of the file in the appropriate place in your APP\_DIR. This holds in general for other program files such as components, models, and plugins. It doesn't hold for resources such as images -- we'll discuss those in a moment. Notice because it looks in APP\_DIR first, you can go ahead and create new controllers, models, views, etc which don't exists in BASE\_DIR and get Yioop to load them.

A Controller must implement the abstract method processRequest. Recall this is the method that bootstrap() will call. The processRequest method should make use of data gotten out of the loaded models as well as data from the web request to do some calculations. Typically, to determine the calculation performed, the controller cleans and looks at \$\_REQUEST['a'], the request activity, and uses the method call(\$activity) to call a method that can handle the activity. When a controller is constructed it makes use of its static variable \$component\_activities to know which components have been associated with it and what activities from these components are allowed. The call() method checks if there is a Component responsible for the requested activity, if there is, it calls that Component's \$activity method, otherwise, the method that handles \$activity is assumed to come from the controller itself. The results of the calculations done in \$activity would typically be put into an associative array \$data. After the call() method completes, processRequest typically takes \$data and calls the base Controller method displayView(\$view, \$data). Here \$view is the whichever loaded view object you would like to display.

To complete the picture of how Yioop eventually produces a web page or other output, we now describe how subclasses of the View class work. Subclasses of View have a field \$layout and two methods helper(\$name), and element(\$name). A subclass of View has at most one Layout and it is used for rendering the header and footer of the page. It is included and instantiated by setting

either the file APP\_DIR/views/layouts/WebLayout.php or BASE\_DIR/views/layouts/WebLayout.php. This file is expected to have in it a class WebLayout extending Layout. The constructor of a Layout takes as argument a view which it sets to an instance variable. The way Layout's get drawn is as follows: When the controller calls displayView(\$view, \$data), this method does some initialization and then calls the render(\$data) of the base View class. This in turn calls the render(\$data) method of whatever Layout was on the view. This render method then draws the header and then calls \$this->view->renderView(\$data); to draw the view, and finally draws the footer.

The methods helper(\$name) and element(\$name) of View load and instantiate, if necessary, and return the Helper or Element \$name in a similar fashion to the model(\$name) method of Controller. Element's have render(\$data) methods and can be used to draw out portions of pages which may be common across Views. Helper's on the other hand are used typically to render UI elements. For example, OptionsHelper has a render(\$id, \$name, \$options, \$selected) method and is used to draw select dropdowns.

When rendering a View or Element one often has css, scripts, images, videos, objects, etc. In BASE\_DIR, the targets of these tags would typically be stored in the css, scripts, or resources folders. The APP\_DIR/css, APP\_DIR/scripts, and APP\_DIR/resources folder are a natural place for them in your customized site. One wrinkle, however, is that APP\_DIR, unlike BASE\_DIR, doesn't have to be under your web servers DOCUMENT\_ROOT. So how does one refer in a link to these folders? To handle this one uses Yioop's ResourceController class which can be invoked by a link like:

```

```

Here c=resource specifies the controller, a=get specifies the activity -- to get a file, n=myicon.png specifies we want the file myicon.png -- the value of n is cleaned to make sure it is a filename before being used, and f=resources specifies the folder -- f is allowed to be one of css, script, or resources. This would get the file APP\_DIR/resources/myicon.png .

This completes our description of the Yioop framework and how to build a new site using it. It should be pointed out that code in the APP\_DIR can be localized using the same mechanism as in BASE\_DIR. More details on this can be found in the section on [Localizing Yioop](#) .

[Return to table of contents](#) .

## Embedding Yioop in an Existing Site

One use-case for Yioop is to serve search result for your existing site. There are three common ways to do this: (1) On your site have a web-form or links with your installation of Yioop as their target and let Yioop format the results. (2) Use the same kind of form or links, but request an OpenSearch RSS Response from Yioop and then you format the results and display the results within your site. (3) Your site makes functions calls of the Yioop Search API and gets either PHP arrays or a string back and then does what it wants with the results. For access method (1) and (2) it is possible to have Yioop on an different machine so that it doesn't consume your main web-site's machines resources. As we mentioned in the configuration section it is possible to disable each of these access paths from within the Admin portion of the web-site. This might be useful for instance if you are using access methods (2) or (3) and don't want users to be able to access the Yioop search results via its built in web form. We will now spend a moment to look at each of these access methods in more detail...

## Accessing Yioop via a Web Form

A very minimal code snippet for such a form would be:

```
<form method="get" action='YIOOP_LOCATION'>
<input type="hidden" name="its" value="TIMESTAMP_OF_CRAWL_YOU_WANT" />
<input type="hidden" name="l" value="LOCALE_TAG" />
<input type="text" name="q" value="" />
<button type="submit">Search</button>
</form>
```

In the above form, you should change YIOOP\_LOCATION to your instance of Yioop's web location,

the IndexArchive folder that you want Yioop to serve results from, LOCALE\_TAG should be the locale you want results displayed in, for example, en-US for American English. In addition, to embedding this form on some page on your site, you would probably want to change the resources/yioop.png image to something more representative of your site. You might also want to edit the file views/search\_view.php to give a link back to your site from the search results.

If you had a form such as above, clicking Search would take you to the URL:

```
YIOOP_LOCATION?its=TIMESTAMP_OF_CRAWL_YOU_WANT&l=LOCALE_TAG&q=QUERY
```

where QUERY was what was typed in the search form. Yioop supports two other kinds of queries: Related sites queries and cache look-up queries. The related query format is:

```
YIOOP_LOCATION?its=TIMESTAMP_OF_CRAWL_YOU_WANT&l=LOCALE_TAG&a=related&arg=URL
```

where URL is the url that you are looking up related URLs for. To do a look up of the Yioop cache of a web page the url format is:

```
YIOOP_LOCATION?its=TIMESTAMP_OF_CRAWL_YOU_WANT&l=LOCALE_TAG&q=QUERY&a=cache&arg=URL
```

Here the terms listed in QUERY will be styled in different colors in the web page that is returned; URL is the url of the web page you want to look up in the cache.

### Accessing Yioop and getting and OpenSearch RSS, JSON, or JSONP Response

The same basic urls as above can return RSS or JSON results simply by appending to the end of the them &f=rss or &f=json. If you would like to embed json return in a callback Javascript function (JSONP), you can append &f=json&callback=myCallbackFunction . This of course only makes sense for usual and related url queries -- cache queries return web-pages not a list of search results. Here is an example of what a portion of an RSS result might look like:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
xmlns:atom="http://www.w3.org/2005/Atom"
>
<channel>
  <title>PHP Search Engine - Yioop : art</title>
  <language>en-US</language>
  <link>http://localhost/git/yioop/?f=rss&q=art&its=1317152828</link>
  <description>Search results for: art</description>
  <opensearch:totalResults>1105</opensearch:totalResults>
  <opensearch:startIndex>0</opensearch:startIndex>
  <opensearch:itemsPerPage>10</opensearch:itemsPerPage>
  <atom:link rel="search" type="application/opensearchdescription+xml"
href="http://localhost/git/yioop/yioopbar.xml"/>
  <opensearch:Query role="request" searchTerms="art"/>

  <item>
    <title> An Online Fine Art Gallery U Can Buy Art -
Buy Fine Art Online</title>

    <link>http://www.ucanbuyart.com/</link>
    <description> UCanBuyArt.com is an online art gallery
and dealer designed... art gallery and dealer designed for art
sales of high quality and original... art sales of high quality
and original art from renowned artists. Art</description>
  </item>
  ...
  ...
</channel>
</rss>
```

Notice the opensearch: tags tell us the totalResults, startIndex and itemsPerPage. The opensearch:Query tag tells us what the search terms were.

### Accessing Yioop via the Function API

The last way we will consider to get search results out of Yioop is via its function API. The Yioop Function API consists of the following three methods in src/controllers/SearchController.php :

```
function queryRequest($query, $results_per_page, $limit = 0)

function relatedRequest($url, $results_per_page, $limit = 0,
  $crawl_time = 0)

function cacheRequest($url, $highlight=true, $terms = "",
  $crawl_time = 0)
```

These methods handle basic queries, related queries, and cache of web page requests respectively.



to cacheRequest are to specify whether or not you want syntax highlighting of any of the words in the returned cached web-page. If wanted then \$terms should be a space separated list of terms.

An example script showing what needs to be set-up before invoking these methods as well as how to extract results from what is returned can be found in the file examples/SearchApi.php . If you are building a project that uses this API, it might be useful to develop your project using the very convenient PHP package manager known as Composer. Please check out the [Tutorial on Using Yioop with Composer](#) .

[Return to table of contents](#) .

## Localizing Yioop to a New Language

The Manage Locales activity can be used to configure Yioop for use with different languages and for different regions. If you decide to customize your Yioop installation by adding files to WORK\_DIRECTORY/app as described in the [Building a Site using Yioop as a Framework](#) section, then the localization tools described in this section can also be used to localize your custom site. Clicking the Manage Locales activity one sees a page like:

The screenshot shows two forms. The first is 'Add Locale' with fields for 'Locale Name', 'Locale Tag', 'Writing Mode' (a dropdown menu), and 'Locale Enabled' (a checked checkbox). Below these is a 'Submit' button. The second form is 'Locale List' with a pagination control 'Row 0 to 21 of 21 Show 50 [Search]' and a table of existing locales.

Name	Tag	Mode	Enabled	Percent	Actions
العربية	ar	rl-tb	true	35	<a href="#">Edit</a> <a href="#">Delete</a>
Deutsch	de	lr-tb	true	0	<a href="#">Edit</a> <a href="#">Delete</a>
English	en-US	lr-tb	true	100	<a href="#">Edit</a> <a href="#">Delete</a>
Español	es	lr-tb	true	15	<a href="#">Edit</a> <a href="#">Delete</a>

The first form on this activity allows you to create a new locale -- an object representing a language and a region. The first field on this form should be filled in with a name for the locale in the language of the locale. So for French you would put Français. The locale tag should be the IETF language tag. The **Writing Mode** element on the form is to specify how the language is written. There are four options: lr-tb -- from left-to-write from the top of the page to the bottom as in English, rl-tb from right-to-left from the top the page to the bottom as in Hebrew and Arabic, tb-rl from the top of the page to the bottom from right-to-left as in Classical Chinese, and finally, tb-lr from the top of the page to the bottom from left-to-right as in non-cyrillic Mongolian or American Sign Language. lr-tb and rl-tb support work better than the vertical language support. As of this writing, Internet Explorer and WebKit based browsers (Chrome/Safari) have some vertical language support and the Yioop stylesheets for vertical languages still need some tweaking. For information on the status in Firefox check out this [writing mode bug](#). Finally, the **Locale Enabled** checkbox controls whether or not to present the locale on the Settings Page. This allows you to choose only the locales you want for your website without having to delete the locale data for other locales you don't want, but may want in the future as more translate strings become available.

Beneath the Add Locale form is an alphabetical in the local-tah table listing some of the current locales. The Show Dropdown let's you control let's you control how many of these locales are displayed in one go. The Search link lets you bring up an advance search form to search for particular locales and also allows you to control the direction of the listing. The Locale List table first colume has a link with the name of the locale. Clicking on this link brings up a page where one can edit the strings for that locale. The next two columns of the Locale List table give the locale tag and writing direction of the locale, this is followed by the percent of strings translated. Clicking the Edit

Delete link let's one delete a locale and all its strings.

To translate string ids for a locale click on its name link. This should display the following forms and table of string id and their translated values:

**Edit Locale: English** [Back](#)  
Show:  Filter:  

admin_controller_need_cookies	Cookies needed to login!!
admin_controller_login_successful	Login Successful!!
admin_controller_account_not_active	Account not active. Maybe re
admin_controller_no_back_button	No Back Button During Logir
admin_controller_login_failed	
admin_controller_login_to_config	Login to continue configurat
admin_controller_status_updates_stopped	Status updates have stoppec
admin_controller_account_access	Account Access
admin_controller_social	Social

In the above case, the link for English was clicked. The Back link in the corner can be used to written to the previous form. The drop down controls whether to display all localizable strings or just those missing translations. The Filter field can be used to restrict the list of string id's and translations presented to just those containing what is in this field. Beneath this dropdown, the Edit Locale page mainly consists of a two column table: the right column being string ids, the left column containing what should be their translation into the given locale. If no translation exists yet, the field will be displayed in red. String ids are extracted by Yioop automatically from controller, view, helper, layout, and element class files which are either in the Yioop Installation itself or in the installation WORK\_DIRECTORY/app folder. Yioop looks for tl() function calls to extract ids from these files, for example, on seeing tl('search\_view\_query\_results') Yioop would extract the id search\_view\_query\_results; on seeing tl('search\_view\_calculated', \$data['ELAPSED\_TIME']) Yioop would extract the id, 'search\_view\_calculated'. In the second case, the translation is expected the translation to have a %s in it for the value of \$data['ELAPSED\_TIME']. Note %s is used regardless of the the type, say int, float, string, etc., of \$data['ELAPSED\_TIME']. tl() can handle additional arguments, whenever an additional argument is supplied an additional %s would be expected somewhere in the translation string. If you make a set of translations, be sure to submit the form associated with this table by scrolling to the bottom of the page and clicking the Submit link. This saves your translations; otherwise, your work will be lost if you navigate away from this page. One aid to translating is if you hover your mouse over a field that needs translation, then its translation in the default locale (usually English) is displayed. If you want to find where in the source code a string id comes from the ids follow the rough convention file\_name\_approximate\_english\_translation. So you would expect to find admin\_controller\_login\_successful in the file controllers/admin\_controller.php . String ids with the prefix db\_ (such as the names of activities) are stored in the database. So you cannot find these ids in the source code. The tooltip trick mentioned above does not work for database string ids.

### Localizing Wiki Pages

When a user goes to a wiki page with a URL such as

YIOOP\_LOCATION/group/id\_of\_a\_group/Some\_Page\_Name

or

YIOOP\_LOCATION/group/id\_of\_a\_group/Some\_Page\_Name

or for the public group possibly with

YIOOP\_LOCATION/p/Some\_Page\_Name

the page that is displayed is in the locale that has been most recently set for the user. If no locale was set, then Yioop tries to determine the locale based on browser header info, and if this fails, falls back to the Default Locale set when Yioop was configure. When one edits a wiki page the locale that

Main Page - [[Read](#) | [Edit](#) | [Pages](#)]

Locale: en-US

Page: Main [[Settings](#)]



To edit the page for a different locale, choose the locale you want using the Settings page while logged in and then navigate to the wiki page you would like to edit (using the same name from the original language). Suppose you were editing the Dental\_Floss page in en-US locale. To make the French page, you click Settings on the top bar of Yioop, go to your account settings, and choose French (fr-FR) as the language. Now one would navigate back to the wiki page you were on, the Dental\_Floss page, which doesn't exist for French. You could click Edit now and make the French page at this location, but this would be sub-optimal as the French word for dental floss is dentrifice. So instead, on the fr-FR Dental\_Floss edit page, you edit the page Settings to make this page a Page Alias for Dentrifice, and then create and edit the French Dentrifice article. If a user then starts on the English version of the page and switches locales to French they will end up on the Dentrifice page. You should also set up the page alias in the reverse direction as well, to handle when someone start on the French Dentrifice page and switches to the en-US Dentrifice.

### Adding a stemmer, segmenter or supporting character n-gramming for your language

Depending on the language you are localizing to, it may make sense to write a stemmer for words that will be inserted into the index. A stemmer takes inflected or sometimes derived words and reduces them to their stem. For instance, jumps and jumping would be reduced to jump in English. As Yioop crawls it attempts to detect the language of a given web page it is processing. If a stemmer exists for this language it will call the Tokenizer class's stem(\$word) method on each word it extracts from the document before inserting information about it into the index. Similarly, if an end-user is entering a simple conjunctive search query and a stemmer exists for his language settings, then the query terms will be stemmed before being looked up in the index. Currently, Yioop comes with stemmers for English, French, German, Italian, and Russian. The English stemmer uses the Porter Stemming Algorithm [ [P1980](#) ], the other stemmers are based on the algorithms presented at [snowball.tartoros.org](http://snowball.tartoros.org). Stemmers should be written as a static method located in the file `WORK_DIRECTORY/locale/en_US/resources/Tokenizer.php`. The [snowball.tartoros.org](http://snowball.tartoros.org) link points to a site that has source code for stemmers for many other languages (unfortunately, not written in PHP). It would not be hard to port these to PHP and then add modify the Tokenizer.php file of the appropriate locale folder. For instance, one could modify the file `WORK_DIRECTORY/locale/pt/resources/Tokenizer.php` to contain a class PtTokenizer with a static method stem(\$word) if one wanted to add a stemmer for Portuguese.

The class inside Tokenizer.php can also be used by Yioop to do word segmentation. This is the process of splitting a string of words without spaces in some language into its component words. Yioop comes with an example segmenter for the zh-CN (Chinese) locale. It works by starting at the end of the string and trying to greedily find the longest word that can be matched with the portion of the suffix of the string that has been processed yet (reverse maximal match). To do this it makes use of a word Bloom filter as part of how it detects if a string is a word or not. We describe how to make such filter using token\_tool.php in a moment.

In addition to supporting the ability to add stemmers and segmenters, Yioop also supports a default technique which can be used in lieu of a stemmer called character n-grams. When used this technique segments text into sequences of n characters which are then stored in Yioop as a term. For instance if n were 3 then the word "thunder" would be split into "thu", "hun", "und", "nde", and "der" and each of these would be associated with the document that contained the word thunder. N-grams are useful for languages like Chinese and Japanese in which words in the text are often not separated with spaces. It is also useful for languages like German which can have long compound words. The drawback of n-grams is that they tend to make the index larger. For Yioop built-in locales that do not have stemmer the file, the file `WORK_DIRECTORY/locale/LOCALE-TAG/resources`

```
public static $char_gram_len = 5;
```

This number is the length of string to use in doing char-gramming. If you add a language to Yioop and want to use char gramming merely add a tokenizer.php to the corresponding locale folder with such a line in it.

### Using TokenTool.php to improve search performance and relevance for your language

configs/TokenTool.php is used to create suggest word dictionaries and word filter files for the Yioop search engine. To create either of these items, the user puts a source file in Yioop's WORK\_DIRECTORY/prepare folder. Suggest word dictionaries are used to supply the content of the dropdown of search terms that appears as a user is entering a query in Yioop. They are also used to do spell correction suggestions after a search has been performed. To make a suggest dictionary one can use a command like:

```
php TokenTool.php dictionary filename locale endmarker
```

Here *filename* should be in the current folder or PREP\_DIR, locale is the locale this suggest (for example, en-US) file is being made for and where a file suggest\_trie.txt.gz will be written, and endmarker is the end of word symbol to use in the trie. For example, \$ works pretty well. The format of *filename* should be a sequence of line, each line containing a word or phrase followed by a space followed by a frequency count. i.e., the last thing on the line should be a number. Given a corpus of documents a frequency for a word would be the number of occurrences of that word in the document.

TokenTool.php can also be used to make filter files used by a word segmenter. To make a filter file TokenTool.php is run from the command line as:

```
php TokenTool.php segment-filter dictionary_file locale
```

Here dictionary\_file should be a text file with one word/line, locale is the IANA language tag of the locale to store the results for.

### Obtaining data sets for TokenTool.php

Many word lists with frequencies are obtainable on the web for free with Creative Commons licenses. A good starting point is:

[http://en.wiktionary.org/wiki/Wiktionary:Frequency\\_lists](http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists)

A little script-fu can generally take such a list and output it with the line format of "word/phrase space frequency" needed by TokenTool.php and as the word/line format used for filter files.

### Spell correction and romanized input with locale.js

Yioop supports the ability to suggest alternative queries after a search is performed. These queries are mainly restricted to fixing typos in the original query. In order to calculate these spelling corrections, Yioop takes the query and for each query term computes each possible single character change to that term. For each of these it looks up in the given locale's suggest\_trie.txt.gz a frequency count of that variant, if it exists. If the best suggestion is some multiple better than the frequency count of the original query then Yioop suggests this alternative query. In order for this to work, Yioop needs to know what constitutes a single character in the original query. The file locale.js in the WORK\_DIRECTORY/locale/LOCALE\_TAG/resources folder can be used to specify this for the locale given by LOCALE\_TAG. To do this, all you need to do is specify a Javascript variable alpha. For example, for French (fr-FR) this looks like:

```
var alpha = "aâàbcçdeéëfghiiijklmnoöpqrstuûvwxyz";
```

The letters do not have to be in any alphabetical order, but should be comprehensive of the non-punctuation symbols of the language in question.

Another thing locale.js can be used for is to given mappings between roman letters and other scripts for use in the Yioop's autosuggest dropdown that appears as you type a query. As you type, scripts/suggest.js function onTypeTerm is called. This in turn will cause a particular locale's locale.js function transliterate(query) if it exists. This function should return a string with the result of the

## Thesaurus Results and Part of Speech Tagging

As mentioned in the [Search Basics](#) topic, for some queries Yioop displays a list of related queries to one side of the search results. These are obtained from a "computer thesaurus". In this subsection, we describe how to enable this facility for English and how you could add this functionality for other languages. If enabled, the thesaurus also can be used to modify search ranking as described in the [Final Reordering](#) of the Yioop Ranking Mechanisms document.

In order to generate suggested related queries, Yioop first tags the original query terms according to part of speech. For the en-US locale, this is done by calling a method: `tagTokenizePartOfSpeech($text)` in `WORK_DIRECTORY/locale/en_US/resources/Tokenizer.php`. For en-US, a simple Brill tagger (see Ranking document for more info) is implemented to do this. After this method is called the terms in `$text` should have a suffix `~part-of-speech` where `~part-of-speech` where `part-of-speech` is one of NN for noun, VB for verb, AJ for adjective, AV for adverb, or some other value (which would be ignored by Yioop). For example, the noun dog might become `dog~NN` after tagging. To localize to another language this method in the corresponding `tokenizer.php` file would need to be implemented.

The second method needed for Thesaurus results is `scoredThesaurusMatches($term, $word_type, $whole_query)` which should also be in `tokenizer.php` for the desired locale. Here `$term` is a term (without a part-of-speech tag), `$word_type` is the part of speech (one of the ones listed above), and `$whole_query` is the original query. The output of this method should be an array of (score => array of thesaurus terms) associations. The score representing one word sense of term. In the case, of English, this method is implemented using [WordNet](#). So for thesaurus results to work for English, WordNet needs to be installed and in either the `config.php` file or `local_config.php` you need to define the constant `WORDNET_EXEC` to the path to the WordNet executable on your file system. On a Linux or OSX system, this might be something like: `/usr/local/bin/wn` .

## Using Stop Words to improve Centroid Summarization

While crawling, Yioop makes use of a summarizer to extract the important portions of the web page both for indexing and for search result snippet purposes. There are three summarizers that come with Yioop a Basic summarizer, which uses an ad hoc approach to finding the most important parts of the document; a Graph-base summarizer, which convert the sentences into vertices, connecting them with weighted edges, and tries to a page-rank like computation, and a centroid summarizer which tries to compute an "average sentence" for the document and uses this to pick representative sentences based on nearness to this average. The summarizer that is used can be set under the Crawl Time tab of [Page Options](#) . This centroid summarizer works better if certain common words (stop words) from the documents language are removed. When using the centroid summarizer, Yioop check to see if `tokenizer.php` for the current locale contains a method `stopwordsRemover($page)`. If it does it calls it, this method takes a string of words are returns a string with all the stop words removed. This method exists for en-US, but, if desired, could also be implemented for other locales to improve centroid summarization.

[Return to table of contents](#) .

## Advanced Topics

### Modifying Yioop Code

One advantage of an open-source project is that you have complete access to the source code. Thus, Yioop can be modified to fit in with your existing project. You can also freely add new features onto Yioop. In this section, we look a little bit at some of the common ways you might try to modify Yioop as well as ways to examine the output of a crawl in a more technical manner. If you decide to modify the source code it is recommended that you look at the [Summary of Files and Folders](#) above again, as well as look at the [online Yioop code documentation](#).

### Handling new File Types

One relatively easy enhancement to Yioop is to enhance the way it processes an existing file type or

projects. So the PHP processors for Microsoft file formats and for PDF are only approximate. These processors can be found in `src/library/processors`. To write your own processor, you should extend either the `TextProcessor` or `ImageProcessor` class. You then need to write in your subclass a static method `process($page, $url)`. Here `$page` is a string representation of a downloaded document of the file type you are going to handle and `$url` is the a canonical url from which this page is downloaded. This method should return an array of the format:

```
$summary['TITLE'] = a title for the document
$summary['DESCRIPTION'] = a text summary extracted from the document
$summary['LINKS'] = an array of links (canonical not relative) extracted
                    from the document.
```

A good reference implementation of a `TextProcessor` subclass can be found in `HtmlProcessor.php`. If you are trying to support a new file type, then to get Yioop to use your processor you need to add lines to your classes constructor specifying the `self::$indexed_file_types` and `self::$mime_processor` static fields. As an example, in the `RssProcessor` class; constructor these lines are:

```
self::$indexed_file_types[] = "rss";
self::$mime_processor["application/rss+xml"] = "RssProcessor";
self::$mime_processor["application/atom+xml"] = "RssProcessor";
```

`self::$indexed_file_types` says that there exists a processor for a given file extension, `self::$mime_processor` says what processor should be used to handle a given mime type. Both these fields are static and inherited from the base class. They are used to get effectively global array of all handled file types and which processors should be used for which extensions. If you create a new processor, it is cool, only relies on code you wrote, and you want to contribute it back to the Yioop, please feel free to e-mail it to [chris@pollett.org](mailto:chris@pollett.org).

### Writing a MediaJob

A `MediaJob` is an activity to be performed by the `MediaUpdater` process on a periodic basis. Currently, three `MediaJob`'s have already been created in the `library/media_jobs` folder: `NewsUpdateJob`, used to update the news feeds which show in the `News` subsearch; `BulkEmailJob`, used to send out emails to thread or group participants when an update has occurred to that thread or group; and `VideoConvertJob`, which is used to convert uploaded videos to a web streaming friendly mp4 format. One can write ones own `MediaJob`'s for any activity that one would like Yioop to do on a periodic basis. One simply creates a subclass of `MediaJob` and places it in the `WORK_DIRECTORY/app/library/media_jobs` folder. The code for the three `MediaJobs` above serve as a useful starting point for writing such jobs. Below though is a summary of the principal methods that a `MediaJob` needs to have in order for it to be run by the `MediaUpdater`:

#### **init()**

called after the class' constructor and allows a user entry point to performs initializations for the job.

#### **checkPrerequisites()**

should return true or false depending on whether the job should run. It can be used to check things like the system time to determine when a job should run.

#### **nondistributedTasks()**

is called only on the name server and only called when media updater is run in non-distributed mode. You can put the nondistributed version of your job in this method.

#### **prepareTasks()**

is called on the name server's media updater only, but is intended to be used in the distributed `MediaUpdater` setting. It is supposed to get the data needed by the job ready before it is sent to a client machine's `MediaUpdater`.

#### **getTasks()**

is called by the name server's web app when a client machine's `MediaUpdater` makes an HTTP request for data for the `MediaJob`. It takes data output by `prepareTasks()` and sends the client its portion of this data (we imagine that a data set might be split between several clients).

#### **doTasks()**

is called after the client `MediaUpdater` has received the `getTasks()` data. It then processes this data.

is called by the name server's web app when a client makes an HTTP request to send processed information back to the web server.

#### **finishTasks()**

is called the name server's MediaUpdater only. It performs any final computations needed on data after it has been sent back to name server.

#### **Writing an Indexing Plugin**

An indexing plugin provides a way that an advanced end-user can extend the indexing capabilities of Yioop. Bundled with Yioop are three example indexing plugins. These are found in the `library/indexing_plugins` folder. We will discuss the code for the recipe and word filter plugin here. The code for the address plugin, used to extract snail mail address from web pages follows the same kind of structure. If you decide to write your own plugin or want to install a third-party plugin you can put it in the folder: `WORK_DIRECTORY/app/library/indexing_plugins`. The recipe indexing plugin can serve as a guide for writing your own plugin if you don't need your plugin to have a configure screen. The recipe plugin is used to detect food recipes which occur on pages during a crawl. It creates "micro-documents" associated with found recipes. These are stored in the index during the crawl under the meta-word "recipe:all". After the crawl is over, the recipe plugin's `postProcessing` method is called. It looks up all the documents associated with the word "recipe:all". It extracts ingredients from these and does clustering of recipes based on ingredient. It finally injects new meta-words of the form "ingredient:some\_food\_ingredient", which can be used to retrieve recipes most closely associated with a given ingredient. As it is written, recipe plugin assumes that all the recipes can be read into memory in one go, but one could easily imagine reading through the list of recipes in batches of the amount that could fit in memory in one go.

The recipe plugin illustrates the kinds of things that can be written using indexing plugins. To make your own plugin, you would need to write a subclass of the class `IndexingPlugin` with a file name of the form `MypluginnamePlugin.php`. Then you would need to put this file in the folder `WORK_DIRECTORY/app/library/indexing_plugins`. `RecipePlugin` subclasses `IndexingPlugin` and implements the following four methods: `pageProcessing($page, $url)`, `postProcessing($index_name)`, `getProcessors()`, `getAdditionalMetaWords()` so they don't have their return NULL default behavior. We explain what each of these is for in a moment. During a web crawl, after a fetcher has downloaded a batch of web pages, it uses a page's `mimetype` to determine a page processor class to extract summary data from that page. The page processors that Yioop implements can be found in the folder `lib/processors`. They have file names of the form `someprocessorname_processor.php`. As a crawl proceeds, your plugin will typically be called to do further processing of a page only in addition to some of these processors. The static method `getProcessors()` should return an array of the form `["someprocessorname1", "someprocessorname2", ...]`, listing the processors that your plugin will do additional processing of documents for. A page processor has a method `handle($page, $url)` called by Yioop with a string `$page` of a downloaded document and a string `$url` of where it was downloaded from. This method first calls the `process($page, $url)` method of the processor to do initial summary extraction and then calls method `pageProcessing($page, $url)` of each `indexing_plugin` associated with the given processor. A `pageProcessing($page, $url)` method is expected to return an array of subdoc arrays found on the given page. Each subdoc array should have a `CrawlConstants::TITLE` and a `CrawlConstants::DESCRIPTION`. The `handle` method of a processor will add to each subdoc the fields: `CrawlConstants::LANG`, `CrawlConstants::LINKS`, `CrawlConstants::PAGE`, `CrawlConstants::SUBDOCTYPE`. The `SUBDOCTYPE` is the name of the plugin. The resulting "micro-document" is inserted by Yioop into the index under the word `nameofplugin:all`. After the crawl is over, Yioop will call the `postProcessing($index_name)` method of each indexing plugin that was in use. Here `$index_name` is the timestamp of the crawl. Your plugin can do whatever post processing it wants in this method. For example, the recipe plugin does searches of the index and uses the results of these searches to inject new meta-words into the index. In order for Yioop to be aware of the meta-words you are adding, you need to implement the method `getAdditionalMetaWords()`. Also, the web snippet you might want in the search results for things like recipes might be longer or shorter than a typical result snippet. The `getAdditionalMetaWords()` method also tells Yioop this information. For example, for the recipe plugin,

```
[ "recipe:" => HtmlProcessor::MAX_DESCRIPTION_LEN,  
  "ingredient:" => HtmlProcessor::MAX_DESCRIPTION_LEN]
```

The WordFilterPlugin illustrates how one can write an indexing plugin with a configure screen. It overrides the base class' pageSummaryProcessing(&\$summary) and getProcessors() methods as well as implements the methods saveConfiguration(\$configuration), loadConfiguration(), setConfiguration(\$configuration), configureHandler(&\$data), and configureView(&\$data). The purpose of getProcessors() was already mentioned under recipe plugin description above. pageSummaryProcessing(&\$summary) is called by a page processor after a page has been processed and a summary generated. WordFilterPlugin uses this callback to check if the title or the description in this summary have any of the words the filter is filtering for and if so takes the appropriate action. loadConfiguration, saveConfiguration(\$configuration), and setConfiguration are three methods to handle persistence for any plugin data that the user can change. The first two operate on the name server, the last might operate on a queue server or a fetcher. loadConfiguration is be called by configureHandler(&\$data) to read in any current configuration, unserialize it and modify it according to any data sent by the user. saveConfiguration(\$configuration) would then be called by configureHandler(&\$data) to serialize and write any \$configuration data that needs to be stored by the plugin. For WordFilterPlugin, a list of filter terms and actions are what is saved by saveConfiguration(\$configuration) and loaded by loadConfiguration. When a crawl is started or when a fetcher contacts the name server, plugin configuration data is sent by the name server. The method setConfiguration(\$configuration) is used to initialize the local copy of a fetcher's or queue server's process with the configuration settings from the name server. For WordFilterPlugin, the filter terms and actions are stored in a field variable by this function.

As has already been hinted at by the configuration discussion above, configureHandler(&\$data) plays the role of a controller for an index plugin. It is in fact called by the AdminController activity pageOptions if the configure link for a plugin has been clicked. In addition, to managing the load and save configuration process, it also sets up any data needed by configureView(&\$data). For WordFilterPlugin, this involves setting a variable \$data["filter\_words"] so that configureView(&\$data) has access to a list of filter words and actions to draw. Finally, the last method of the WordFilterPlugin we describe, configureView(&\$data), outputs using \$data the HTML that will be seen in the configure screen. This HTML will appear in a div tag on the final page. It is initially styled so that it is not displayed. Clicking on the configure link will cause the div tag data to be displayed in a light box in the center of the screen. For WordFilterPlugin, this methods draws a title and textarea form with the currently filtered terms in it. It makes use of Yioop's tl() functions so that the text of the title can be localized to different languages. This form has hidden field c=admin, a=pageOptions option-type=crawl\_time, so that hte AdminController will know to call pageOption and pageOption will know in turn to let plugin's configureHandler methods to get a chance to handle this data.

[Return to table of contents](#) .

### Yioop Command-line Tools

In addition to [TokenTool.php](#) which we describe in the section on localization, and to [ExportPublicHelpDb.php](#) which we describe in the section on the Yioop folder structure, Yioop comes with several useful command-line tools and utilities. We next describe these in roughly their order of likely utility:

- [src/configs/ConfigureTool.php](#) : Used to configure Yioop from the command-line
- [src/executables/ArcTool.php](#) : Used to examine the contents of WebArchiveBundle's and IndexArchiveBundles's
- [src/executables/QueryTool.php](#) : Used to query an index from the command-line
- [src/executables/CodeTool.php](#) : Used to help code Yioop and to help make clean patches for Yioop.
- [src/executables/ClassifierTool.php](#) : Used to make Yioop a Yioop classifier from the command line rather than using the GUI interface.

### Configuring Yioop from the Command-line



machine -- all the other machines might be on virtual private servers to which one has only command-line access. Hence, it is useful to be able to set up a work directory and configure Yioop through the command-line. To do this one can use the script configs/ConfigureTool.php. One can run it from the command-line within the configs folder, with a line like:

```
php ConfigureTool.php
```

When launched, this program will display a menu like:

```
Yioop CONFIGURATION TOOL
+++++

Checking Yioop configuration...
=====
Check Passed.
Using configs/LocalConfig.php so changing work directory above may not work.
=====

Available Options:
=====
(1) Create/Set Work Directory
(2) Change root password
(3) Set Default Locale
(4) Debug Display Set-up
(5) Search Access Set-up
(6) Search Page Elements and Links
(7) Name Server Set-up
(8) Crawl Robot Set-up
(9) Exit program

Please choose an option:
```

Except for the Change root password option, these correspond to the different fieldsets on the Configure activity. The command-line forms one gets from selecting one of these choices let one set the same values as were described earlier in the [Installation](#) section. The change root password option lets one set the account password for root. I.e., the main admin user. On a non-nameserver machine, it is probably simpler to go with a sqlite database, rather than hit on a global mysql database from each machine. Such a barebones local database set-up would typically only have one user, root

Another thing to consider when configuring a collection of Yioop machines in such a setting, is, by default, under Search Access Set-up, subsearch is unchecked. This means the RSS feeds won't be downloaded hourly on such machines. If one unchecks this, they can be.

### Examining the contents of WebArchiveBundle's and IndexArchiveBundles's

The command-line script src/executables/ArcTool.php can be used to examine and manipulate the contents of a WebArchiveBundle or an IndexArchiveBundle. Below is a summary of the different command-line uses of arc\_tool.php:

**php ArcTool.php count bundle\_name**

or

**php ArcTool.php count bundle\_name save**

returns the counts of docs and links for each shard in bundle as well as an overall total. The second command saves the just computed count into the index description (can be used to fix the index count if it gets screwed up).

**php ArcTool.php dict bundle\_name word**

returns index dictionary records for word stored in index archive bundle.

**php ArcTool.php info bundle\_name**

return info about documents stored in archive.

**php ArcTool.php inject timestamp file**

injects the urls in file as a schedule into crawl of given timestamp. This can be used to make a closed index unclosed and to allow for continued crawling.

**php ArcTool.php list**

returns a list of all the archives in the Yioop! crawl directory, including non-Yioop! archives in the /archives sub-folder.

**php ArcTool.php mergetiers bundle\_name max\_tier**

merges tiers of word dictionary into one tier up to max\_tier

or

**php ArcTool.php posting bundle\_name generation offset num**

returns info about the posting (num many postings) in bundle\_name at the given generation and offset

**php ArcTool.php rebuild bundle\_name**

Re-extracts words from summaries files in bundle\_name into index shards then builds a new dictionary

**php ArcTool.php reindex bundle\_name**

Reindex the word dictionary in bundle\_name using existing index shards

**php ArcTool.php shard bundle\_name generation**

Prints information about the number of words and frequencies of words within the generation'th index shard in the bundle

**php ArcTool.php show bundle\_name start num**

outputs items start through num from bundle\_name or name of non-Yioop archive crawl folder

The bundle name can be a full path name, a relative path from the current directory, or it can be just the bundle directory's file name in which case WORK\_DIRECTORY/cache will be assumed to be the bundle's location. The following are some examples of using arc tool. Recall a backslash in Unix/OSX terminal is the line continuation character, so we can image lines where it is indicated below as being all on one line. They are not all from the same session:

```
|chris-polletts-macbook-pro:executables:108>php ArcTool.php list
Found Yioop Archives:
=====
0-Archive1334468745
0-Archive1336527560
IndexData1334468745
IndexData1336527560

Found Non-Yioop Archives:
=====
english-wikipedia2012

...

|chris-polletts-macbook-pro:executables:158>php ArcTool.php info \
/Applications/XAMPP/xamppfiles/htdocs/crawls/cache/IndexData1293767731

Bundle Name: IndexData1293767731
Bundle Type: IndexArchiveBundle
Description: test
Number of generations: 1
Number of stored links and documents: 267260
Number of stored documents: 16491
Crawl order was: Page Importance
Seed sites:
  http://www.ucanbuyart.com/
  http://www.ucanbuyart.com/fine_art_galleries.html
  http://www.ucanbuyart.com/indexucba.html
Sites allowed to crawl:
  domain:ucanbuyart.com
  domain:ucanbuyart.net
Sites not allowed to be crawled:
  domain:arxiv.org
  domain:ask.com
Meta Words:
  http://www.ucanbuyart.com/(.+)/(.+)/(.+)/(.+)/

|chris-polletts-macbook-pro:executables:202>php ArcTool.php show \
/Applications/XAMPP/xamppfiles/htdocs/crawls/cache/Archive1293767731 0 3

BEGIN ITEM, LENGTH:21098
[URL]
http://www.ucanbuyart.com/robots.txt
[HTTP RESPONSE CODE]
404
[MIMETYPE]
text/html
[CHARACTER ENCODING]
ASCII
[PAGE DATA]
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <base href="http://www.ucanbuyart.com/" />
</pre>
...

|chris-polletts-macbook-pro:bin:117>php arc_tool.php reindex IndexData1317414152
```

```
.....
[Sat, 01 Oct 2011 11:05:28 -0700] Merging tiers of dictionary
```

```
Final Merge Tiers
```

```
Reindex complete!!
```

The `mergetiers` command is like a partial reindex. It assumes all the shard words have been added to the dictionary, but that the dictionary still has more than one tier (tiers are the result of incremental log-merges which are made during the crawling process). The `mergetiers` command merges these tiers into one large tier which is then usable by Yioop for query processing.

### Querying an Index from the command-line

The command-line script `src/executables/QueryTool.php` can be used to query indices in the Yioop `WORK_DIRECTORY/cache`. This tool can be used on an index regardless of whether or not Apache is running. It can be used for long running queries that might timeout when run within a browser to put their results into memcache or filecache. The command-line arguments for the query tool are:

```
php QueryTool.php query num_results start_num lang_tag
```

The default `num_results` is 10, `start_num` is 0, and `lang_tag` is `en-US`. The following shows how one could do a query on "Chris Pollett":

```
|chris-polletts-macbook-pro:executables:141>php QueryTool.php "Chris Pollett"
=====
TITLE: ECCC - Pointers to
URL: http://eccc.hpi-web.de/static/pointers/personal_www_home_pages_of_complexity_theorists/
IPs: 141.89.225.3
DESCRIPTION: Homepage of the Electronic Colloquium on Computational Complexity located
at the Hasso Plattner Institute of Potsdam, Germany Personal WWW pages of
complexity people 2011 2010 2009 2011...1994 POINTE
Rank: 3.9551158411
Relevance: 0.492443777769
Proximity: 1
Score: 4.14
=====
=====
TITLE: ECCC - Pointers to
URL: http://www.eccc.uni-trier.de/static/pointers/personal_www_home_pages_of_complexity_theorists/
IPs: 141.89.225.3
DESCRIPTION: Homepage of the Electronic Colloquium on Computational Complexity located
at the Hasso Plattner Institute of Potsdam, Germany Personal WWW pages of
complexity people 2011 2010 2009 2011...1994 POINTE
Rank: 3.886318974
Relevance: 0.397622570289
Proximity: 1
Score: 4.03
=====
.....
```

The index the results are returned from is the default index; however, all of the Yioop meta words should work so you can do queries like `"my_query i:timestamp_of_index_want"`. Query results depend on the kind of language stemmer/char-gramming being used, so French results might be better if one specifies `fr-FR` then if one relies on the default `en-US`. `QueryTool` can also be used to explain what index iterators Yioop will use to execute a query. To see this information one can type a command like:

```
php QueryTool.php query plan
```

or

```
php QueryTool.php query explain
```

for example,

```
php QueryTool.php 'chris pollett' explain
```

### A Tool for Coding and Making Patches for Yioop

`src/executables/CodeTool.php` can perform several useful tasks to help developers program for the Yioop environment. Below is a brief summary of its functionality:

#### php CodeTool.php clean path

Replaces all tabs with four spaces and trims all whitespace off ends of lines in the folder or file path.

Adjusts all lines in the files in the folder at path (or if path is a file just that) of the form 2009 - \d\d\d\d to the form 2009 - this\_year where this\_year is the current year.

#### **php CodeTool.php longlines path**

Prints out all lines in files in the folder or file path which are longer than 80 characters.

#### **php CodeTool.php replace path pattern replace\_string**

or

#### **php CodeTool.php replace path pattern replace\_string effect**

Prints all lines matching the regular expression pattern followed by the result of replacing pattern with replace\_string in the folder or file path. Does not change files.

#### **php CodeTool.php replace path pattern replace\_string interactive**

Prints each line matching the regular expression pattern followed by the result of replacing pattern with replace\_string in the folder or file path. Then it asks if you want to update the line. Lines you choose for updating will be modified in the files.

#### **php CodeTool.php replace path pattern replace\_string change**

Each line matching the regular expression pattern is update by replacing pattern with replace\_string in the folder or file path. This format does not echo anything, it does a global replace without interaction.

#### **php CodeTool.php search path pattern**

Prints all lines matching the regular expression pattern in the folder or file path.

### **A Command-line Tool for making Yioop Classifiers**

**src/executables/ClassifierTool.php** is used to automate the building and testing of classifiers, providing an alternative to the web interface when a labeled training set is available.

**ClassifierTool.php** takes an activity to perform, the name of a dataset to use, and a label for the constructed classifier. The activity is the name of one of the 'un\*' functions implemented by this class, without the common 'run' prefix (e.g., 'TrainAndTest' for the method runTrainAndTest). The dataset is specified as the common prefix of two indexes that have the suffixes "Pos" and "Neg", respectively. So if the prefix were "DATASET", then this tool would look for the two existing indexes "DATASET Pos" and "DATASET Neg" from which to draw positive and negative examples. Each document in these indexes should be a positive or negative example of the target class, according to whether it's in the "Pos" or "Neg" index. Finally, the label is just the label to be used for the constructed classifier.

Beyond these options (set with the -a, -d, and -l flags), a number of other options may be set to alter parameters used by an activity or a classifier. These options are set using the -S, -I, -F, and -B flags, which correspond to string, integer, float, and boolean parameters respectively. These flags may be used repeatedly, and each expects an argument of the form NAME=VALUE, where NAME is the name of a parameter, and VALUE is a value parsed according to the flag. The NAME should match one of the keys of the options member of this class, where a period ('.') may be used to specify nesting. For example:

```
-I debug=1          # set the debug level to 1
-B cls.use_nb=0    # tell the classifier to use Naive Bayes
```

To build and evaluate a classifier for the label 'spam', trained using the two indexes "DATASET Neg" and "DATASET Pos", and a maximum of the top 25 most informative features:

```
php ClassifierTool.php -a TrainAndTest -d 'DATASET' -l 'spam' -I cls.chi2.max=25
```

The above assume we are in the folder of ClassifierTool.php.

## **References**

### **[APC2003]**

Serge Abiteboul and Mihai Preda and Gregory Cobena. [Adaptive on-line page importance computation](#). In: Proceedings of the 12th international conference on World Wide Web. pp.280-290. 2003.

Bloom, Burton H. [Space/time trade-offs in hash coding with allowable errors](#). Communications of the ACM Volume 13 Issue 7. pp. 422–426. 1970.

**[BSV2004]**

Paolo Boldi and Massimo Santini and Sebastiano Vigna. [Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations](#). Algorithms and Models for the Web-Graph. pp. 168–180. 2004.

**[BP1998]**

Brin, S. and Page, L. [The Anatomy of a Large-Scale Hypertextual Web Search Engine](#). In: Seventh International World-Wide Web Conference (WWW 1998). April 14-18, 1998. Brisbane, Australia. 1998.

**[BCC2010]**

S. Büttcher, C. L. A. Clarke, and G. V. Cormack. [Information Retrieval: Implementing and Evaluating Search Engines](#). MIT Press. 2010.

**[DG2004]**

Jeffrey Dean and Sanjay Ghemawat. [MapReduce: Simplified Data Processing on Large Clusters](#). OSDI'04: Sixth Symposium on Operating System Design and Implementation. 2004

**[GGL2003]**

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. [The Google File System](#). 19th ACM Symposium on Operating Systems Principles. 2003.

**[GLM2007]**

A. Genkin, D. Lewis, and D. Madigan. [Large-scale bayesian logistic regression for text categorization](#). Technometrics. Volume 49. Issue 3. pp. 291–304, 2007.

**[H2002]**

T. Haveliwala. [Topic-Sensitive PageRank](#). Proceedings of the Eleventh International World Wide Web Conference (Honolulu, Hawaii). 2002.

**[KSV2010]**

Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. [A Model of Computation for MapReduce](#). Proceedings of the ACM Symposium on Discrete Algorithms. 2010. pp. 938-948.

**[KC2004]**

Rohit Khare and Doug Cutting. [Nutch: A flexible and scalable open-source web search engine](#). CommerceNet Labs Technical Report 04. 2004.

**[LDH2010]**

Jimmy Lin and Chris Dyer. [Data-Intensive Text Processing with MapReduce](#). Synthesis Lectures on Human Language Technologies. Morgan and Claypool Publishers. 2010.

**[LM2006]**

Amy N. Langville and Carl D. Meyer. [Google's PageRank and Beyond](#). Princeton University Press. 2006.

**[MRS2008]**

C. D. Manning, P. Raghavan and H. Schütze. [Introduction to Information Retrieval](#). Cambridge University Press. 2008.

**[MKSR2004]**

G. Mohr, M. Kimpton, M. Stack, and I. Ranitovic. [Introduction to Heritrix, an archival quality web crawler](#). 4th International Web Archiving Workshop. 2004.

**[PTSHVC2011]**

Manish Patil, Sharma V. Thankachan, Rahul Shah, Wing-Kai Hon, Jeffrey Scott Vitter, Sabrina Chandrasekaran. [Inverted indexes for phrases and strings](#). Proceedings of the 34th Annual

pp 555--564. 2011.

**[P1997a]**

J. Peek. [Summary of the talk: The AltaVista Web Search Engine](#) by Louis Monier. USENIX Annual Technical Conference Anaheim, California. ;login: Volume 22. Number 2. April 1997.

**[P1997b]**

J. Peek. [Summary of the talk: The Inktomi Search Engine by Louis Monier](#). USENIX Annual Technical Conference. Anaheim, California. ;login: Volume 22. Number 2. April 1997.

**[P1994]**

B. Pinkerton. [Finding what people want: Experiences with the WebCrawler](#). In Proceedings of the First World Wide Web Conference, Geneva, Switzerland. 1994.

**[P1980]**

M.F. Porter. [An algorithm for suffix stripping](#). Program. Volume 14 Issue 3. 1980. pp 130-137. On the same website, there are [stemmers for many other languages](#).

**[PDGQ2006]**

Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan. [Interpreting the Data: Parallel Analysis with Sawzall](#). Scientific Programming Journal. Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure. Volume 13. Issue 4. 2006. pp.227-298.

**[W2009]**

Tom White. [Hadoop: The Definitive Guide](#). O'Reilly. 2009.

**[ZCTSR2004]**

Hugo Zaragoza, Nick Craswell, Michael Taylor, Suchi Saria, and Stephen Robertson. [Microsoft Cambridge at TREC-13: Web and HARD tracks](#). In Proceedings of 3th Annual Text Retrieval Conference. 2004.

[Return to table of contents](#) .