|  |
| --- |
|  |
| A Preliminary User’s Guide for Rhapsode – a Lightweight Document Search Prototype, v0.3.2-BETA |
|  |
| October, 2017  DRAFT |
|  |
| Version 0.3.2-BETA |
| The views, opinions, and/or findings contained in this report are those of The MITRE Corporation and should not be construed as official government position, policy, or decision unless so designated by other documentation.  This document was prepared for authorized distribution only. It has not been approved for public release.  © 2017, The MITRE Corporation. All Rights Reserved. |
| MITRE |
| 7515 Colshire Drive  McLean, VA 22102 |

Table of Contents

[1. Introduction 6](#_Toc494895086)

[1.1 About MITRE 6](#_Toc494895087)

[1.2 Purpose and Scope of Rhapsode 6](#_Toc494895088)

[1.3 System Requirements 6](#_Toc494895089)

[1.4 Changes 7](#_Toc494895090)

[2. Notes on Upgrading to Rhapsode Version 0.30 9](#_Toc494895091)

[3. Quick Start Guide 11](#_Toc494895092)

[4. Building an Index 14](#_Toc494895093)

[4.1 Basic: Indexing a Directory of Files 14](#_Toc494895094)

[5. Searching an Index 15](#_Toc494895095)

[5.1 Basic Search 15](#_Toc494895096)

[5.2 Concordance Search 16](#_Toc494895097)

[5.2.1 Concordance Search: Basic 16](#_Toc494895098)

[5.2.2 Concordance Search: Advanced, Filter Query 19](#_Toc494895099)

[5.2.3 Exporting Concordance Results 20](#_Toc494895100)

[5.3 Co-Occurrences 20](#_Toc494895101)

[5.3.1 Co-Occurrences Basic 20](#_Toc494895102)

[5.3.2 Co-Occurrences: Advanced 24](#_Toc494895103)

[5.4 Variant Counter 24](#_Toc494895104)

[5.4.1 Variant Counter: Basic 24](#_Toc494895105)

[5.4.2 Variant Counter (Advanced) 26](#_Toc494895106)

[6. Query Parser Syntax 30](#_Toc494895107)

[6.1 Query Parser Syntax 31](#_Toc494895108)

[6.1.1 Key word Query 31](#_Toc494895109)

[6.1.2 Fuzzy Query 31](#_Toc494895110)

[6.1.3 Wildcard Query 31](#_Toc494895111)

[6.1.4 Boolean Query 31](#_Toc494895112)

[6.1.5 Phrasal searches 31](#_Toc494895113)

[6.1.6 Range Searches 32](#_Toc494895114)

[6.1.7 Regular Expression Searches 32](#_Toc494895115)

[6.1.8 Directional Phrasal Searches 32](#_Toc494895116)

[6.1.9 Recursive phrasal searches 32](#_Toc494895117)

[6.1.10 Important Caveats 33](#_Toc494895118)

Executive Summary

Rhapsode is a free-text search prototype that will enable users to define requirements for their search needs. These requirements can be used in selecting a commercial tool or in guiding developers. Rhapsode is designed to work on a desktop with a single user at a time. Rhapsode is not designed to solve enterprise wide information retrieval tasks.

This users’ guide has not yet been fully updated for Rhapsode 0.3.2-BETA. I offer it as a

License

NOTICE

Licensed to the Apache Software Foundation (ASF) under one or more

contributor license agreements. See the NOTICE file distributed with

this work for additional information regarding copyright ownership.

The ASF licenses this file to You under the Apache License, Version 2.0

(the "License"); you may not use this file except in compliance with

the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

This software was produced for the U.S. Government

under Basic Contract No. W15P7T-13-C-A802, W15P7T-12-C-F600,

W15P7T-13-C-F600, and TIRNO-99-D-00005, and is

subject to the Rights in Noncommercial Computer Software

and Noncommercial Computer Software Documentation

Clause 252.227-7014 (FEB 2012)

(C) 2013- 2017 The MITRE Corporation. All Rights Reserved.

# Introduction

## About MITRE

The MITRE Corporation is a not-for-profit organization chartered to work in the public interest. As a national resource, we apply our expertise in systems engineering, information technology, operational concepts, and enterprise modernization to address our sponsors' critical needs.[[1]](#footnote-1)

## Purpose and Scope of Rhapsode

Rhapsode is a free-text search prototype. The goal of the prototype is to help users define requirements for their search needs. These requirements can then be used in selecting a commercial tool or in guiding developers. With a few exceptions, Rhapsode is a lightweight integration of open source software libraries. While these libraries offer a tremendous amount of flexibility in indexing and searching, we have selected the most basic options for this prototype. The sponsor documents are primarily in English, and they are mostly comprised of the following formats: Microsoft Office, PDF and Ascii or UTF-8 html/text files. Rhapsode is designed to work on a desktop with a single user at a time. Rhapsode is not intended to solve enterprise-wide search tasks.

We have tried to make this user’s guide as accessible as possible. There is a quick-start guide, and we have general/basic sections as well as advanced sections and topics. We try to label each as such so that the reader can skip the advanced sections and options when getting acquainted with Rhapsode.

In ancient Greece, a *rhapsode* was an epic poet who wove together syntactic and narrative elements from the long tradition of oral poetry in order to create a new narrative. Knowledge workers today often weave together disparate pieces of information to gain new insight into their area of study.

## System Requirements

Rhapsode is written in Java, and was developed on a Windows 10 operating system. Rhapsode requires at least Java 8. We have not tested Rhapsode on Windows XP or Linux operating systems. We have only tested the Rhapsode indexer and searcher on a 64-bit Windows 10 operating system with Internet Explorer 11 and Google Chrome. Rhapsode now includes a 64-bit Java Runtime Environment (JRE) under “resources/jres”. Version 0.30 of Rhapsode includes Java version 1.8.0\_92-b14.

From a port security standpoint, the Rhapsode search server communicates only through 127.0.0.1 (localhost) on port 8092. As a security feature, we have compiled these settings into the application, and they are not modifiable without recompiling the source code.

## Changes

Version 0.3.2-BETA (October, 2017)

First release

# Quick Start Guide

The first step is to unzip the zip file and maintain the directory hierarchies. The code needs to be executed in a mapped directory, not a directory identified by a network path. If you are working in Windows 10, click on the file location, and if it starts with a letter followed by a colon, as in Figure ‎3‑1, the code will run. However, if the path starts with two backslashes, that directory points to a network location and is a UNC path. If this is the case, check your documentation for how to map a network drive.

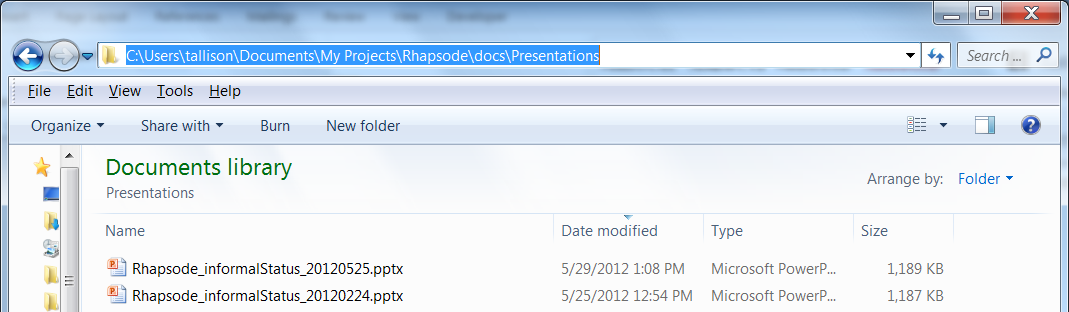


Figure ‎3‑1: Example of a Directory Path that Starts with a Directory Letter

After completing the unzipping process, you will see folders and batch files as shown in Figure ‎3‑2.

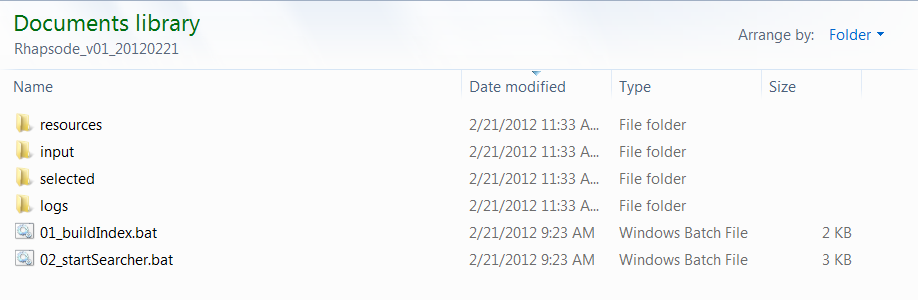


Figure ‎3‑2: Main Directory for Rhapsode

The first step is to build an index. Place your documents in the “input” directory, and double-click “01\_buildIndex.bat”. A command window will pop up and show some status messages. When the index is completed, the command window will output the results and wait for the user to press the return key.

Once the index has been built and the user has closed the indexer command window, the second step is to launch the Rhapsode search server. Double-click “02\_startSearcher.bat”, and a command window will open. At the same time, your default browser *should* automatically open to the correct address: <http://localhost:8092/rhapsode/index.html>. **Leave the command window open.**

Once the server has started, if the browser window hasn’t opened automatically already, open a web browser and navigate to <http://localhost:8092/rhapsode/index.html>. You should see something similar to Figure ‎3‑3.

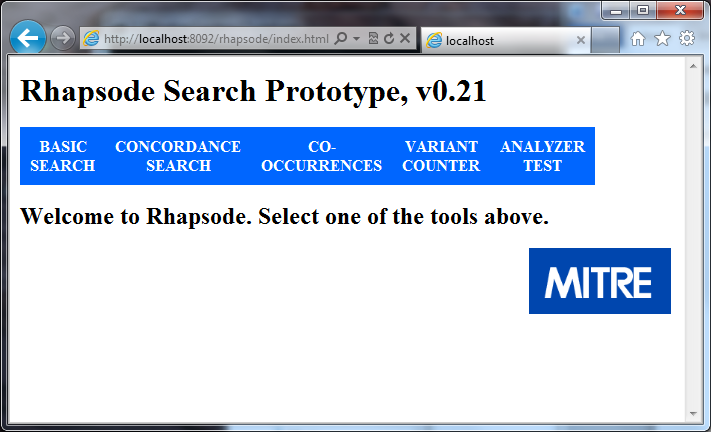


Figure ‎3‑3: Main Screen for Rhapsode

The options:

**Basic Search**. This allows the user to see a few relevant snippets for each document; this format is similar to what most web search engines offer.

**Concordance Search**. This search returns every mention of the search term(s). The user can sort the results in various ways.

**Co-Occurrences**. This performs a concordance search in the background and counts the number of terms that co-occur with your search term.

**Variant Counter**. This allows the user to see how wild card and fuzzy queries match terms in the index. For example, if you type “f?x,” and your corpus contains both “fax” and “fox”, these words will be displayed along with their document frequencies. This also allows the user to see how normalization is working.

**Analyzer Test**. Behind the scenes, an open source component breaks document text and queries into words. This process is called tokenization. It is often helpful to see how this component is carrying out that step. For example, if you enter “http://www.cnn.com”, you will see that the current analyzer is breaking that string into two tokens. If you enter “the quick brown fox,” you will see that the current analyzer is deleting common words (“the” in this case).

Select one of the options and explore.

When you have finished working with the search application, either type the [Enter] key in the command window or click on the red *X* at the top of the command window.

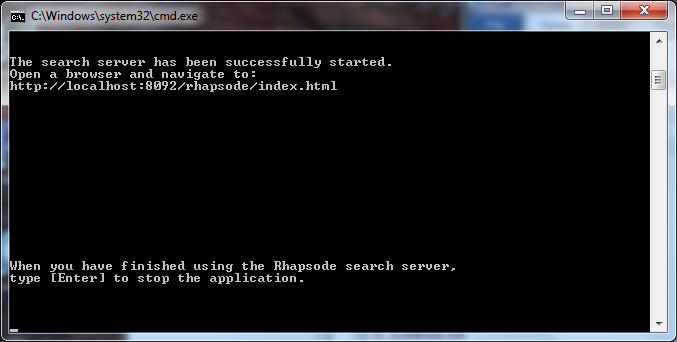


Figure ‎3‑4: Server Command Window

# Building an Index

## Basic: Indexing a Directory of Files

There are several parameters that can be set before building an index. These parameters include such items as:

1. the directory locations of the index directory and the input directory
2. whether or not to append to or overwrite the current index
3. the maximum file of a size to be processed
4. the maximum number of words to be indexed per document
5. the number of threads to use during indexing

To run the indexer, **make sure that the search server is turned off.** Place your files in the “input” folder and double-click the “01\_buildIndex.bat” file. A command window will open and report on its status (see Figure ‎4‑1). When the indexer completes, the command window will report that the indexing has completed, and the user then closes the command window, either by pressing the “Enter” key in the command window or by clicking on the close icon in the top right.

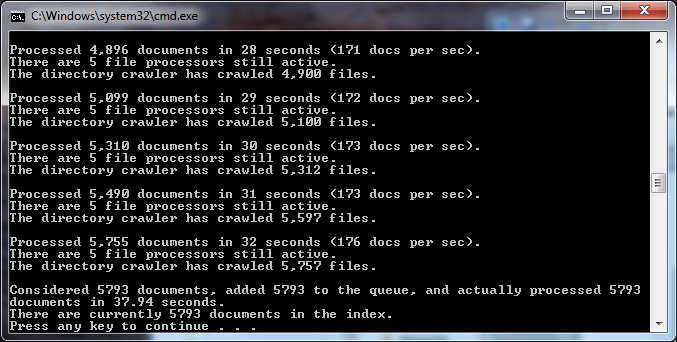


Figure ‎4‑1: Indexer Command Window

The indexer reports exceptions and warnings in a file called “indexer.log” which is located in the “logs” directory at the main directory level. **Do not start the searcher until the indexer has completed and the command window has been closed.**

# Searching an Index

## Basic Search

The goal of basic search is to allow users to identify documents that contain terms of interest. The interface has only three main options: the query, the filter query and the number of results to display per page (see Figure ‎5‑1).

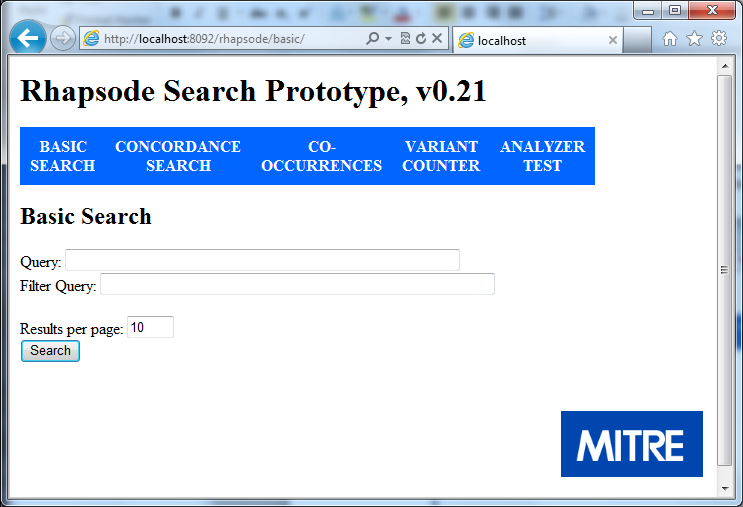


Figure ‎5‑1: Basic Search First Window

Once we make a query, we see results as shown in Figure ‎5‑2.

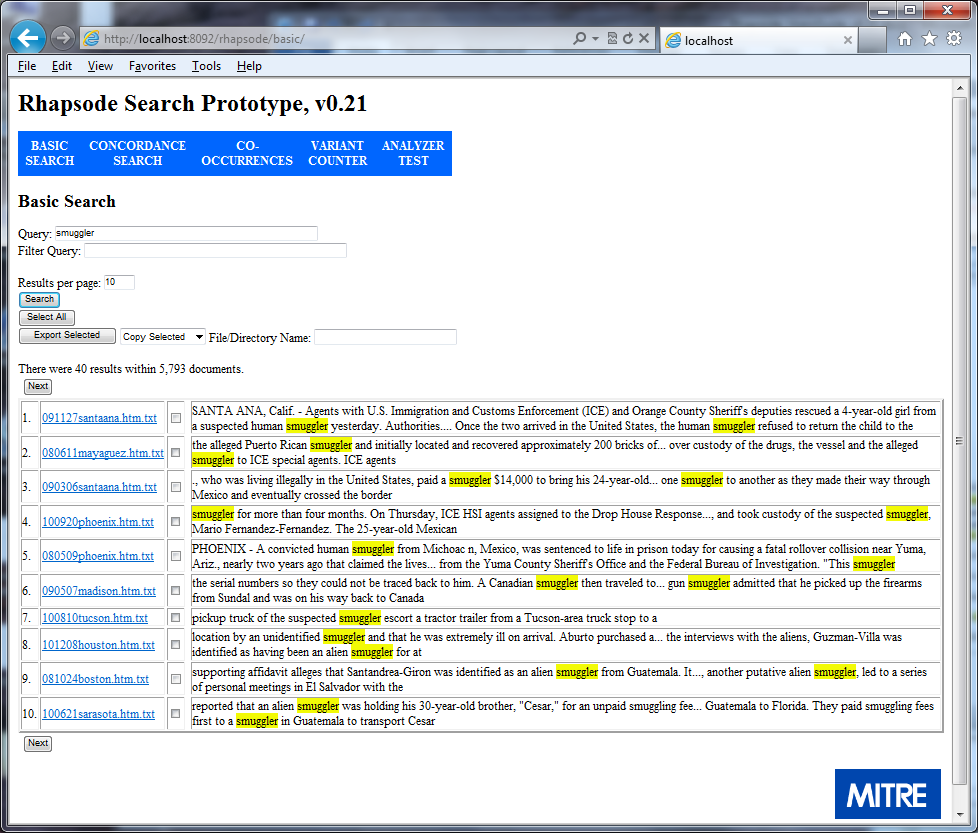


Figure ‎5‑2: Basic Search Results

The user can iterate through the results by hitting the “Next” button. The user may choose to view the document by clicking on the link to the left. The user is able to adjust the results per page, and the user may submit a new query.

## Concordance Search

### Concordance Search: Basic

While basic search allows a user to identify documents that contain a query term, concordance search allows the user to see every time a given term appears in the document set. We show the first screen in Figure ‎5‑3.

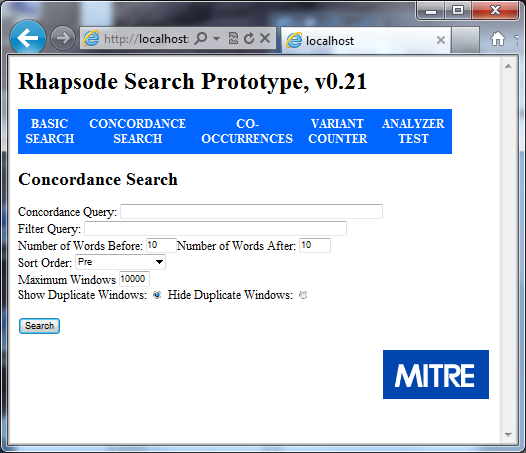


Figure ‎5‑3: First Screen for Concordance Search

For general usage, the user enters a query term in the “Concordance Query” window and then clicks the “Search” button. The results are shown in Figure ‎5‑4.

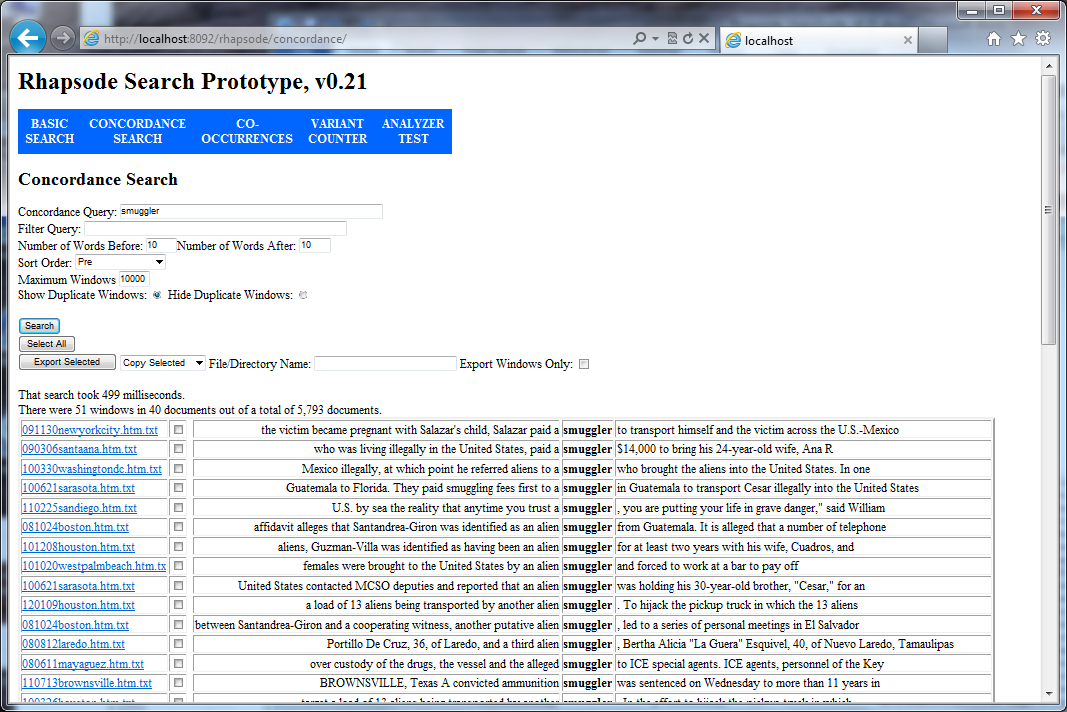


Figure ‎5‑4: Concordance Results

By default, the windows are sorted by words before the target term. The algorithm looks to the first word before the target term, and if there is a match, it looks to the second word and so on. For example, “a” comes before “alien,” and “paid” comes before “to.” There are several other sorting options as shown in Figure ‎5‑5.

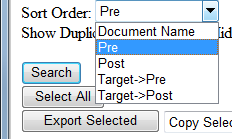


Figure ‎5‑5: Concordance Sort Menu Options

As mentioned, the default is “Pre.” One can also sort on words after the target term (“Post”). One can sort on the target term and then the words before it if there is a match (“Target->Pre”); one can sort on the target term and then the words before it if there is a match (“Target->Post”); one can sort by the “Document Name.” Finally, if the indexer was able to load dates for the documents either through the file names or the directory structure, one can sort by document date. These sort orders will work for both left-to-right languages and right-to-left languages.

In addition to controlling the sort order, the user may select “Show Duplicate Windows” or “Hide Duplicate Windows.” If “Show Duplicate Windows” is selected, all windows will be shown. However, if “Hide Duplicate Windows” is selected, only the first unique window is shown. For this purpose, we say that one window is a duplicate of another if all of the alphanumeric characters are exactly the same and in the same order. If there is an extra “a” in one, both windows would appear because one is not a “duplicate” of another. This option has nothing to do with windows that appear in the same document. If you only want one row per document, use Basic Search.

The “Copy Selected” feature works exactly for concordance search as it does for basic search.

### Concordance Search: Advanced, Filter Query

The general concordance query parser works well for most use cases. However, there are some limitations, including:

1. The concordance query parser can only search one field.
2. The concordance query parser cannot handle range queries.
3. The concordance query parser cannot perform a Boolean “NOT” search.
4. The concordance query parser cannot perform a Boolean “AND” search across terms that are not in the same phrase.

There are times when a user may want to see a target term in only a subset of documents. For example, a user may want to see “smuggler” in the concordance but only in documents that contain “cocaine.” To accomplish this type of search, we have made available the "Filter Query" window (see Figure ‎5‑6). Rhapsode applies the basic Lucene query parser to this query (see section ‎6.1 below) to identify a subset of documents from which to build the concordance.

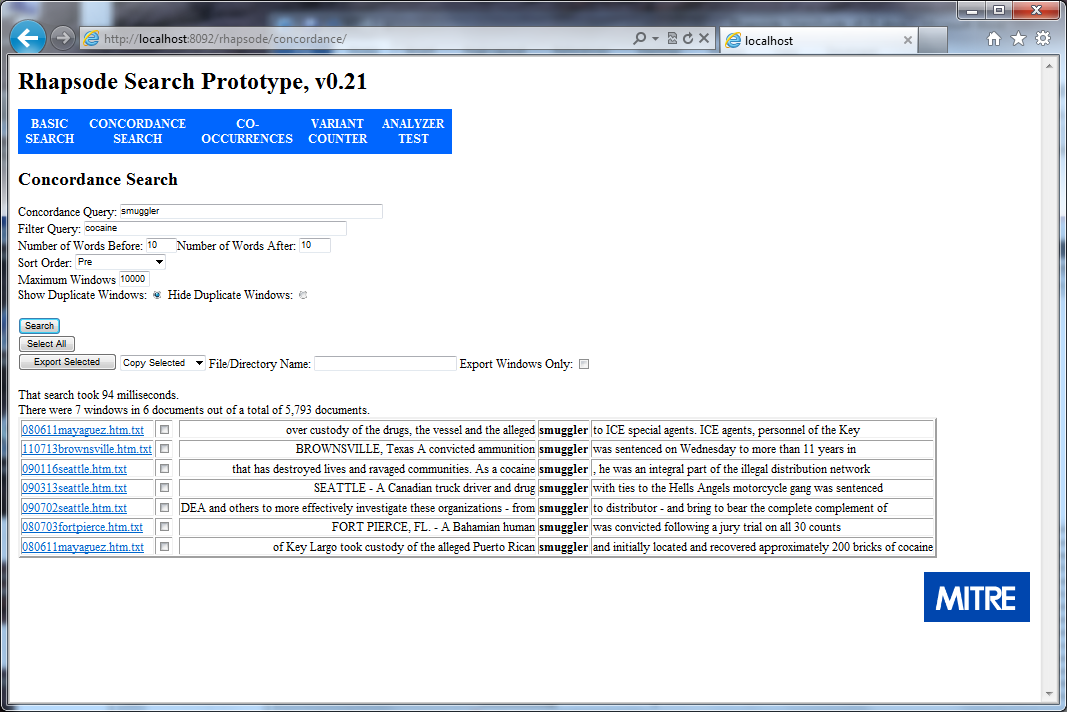


Figure ‎5‑6: Concordance Search with a Filter Query

### Exporting Concordance Results

The concordance searcher has exactly the same export options as the basic searcher.

## Co-Occurrences

### Co-Occurrences Basic

This tool runs the concordance searcher in the background, and it counts the terms that appear before and after the target term. For example, let’s say that the user is interested in brands of counterfeit merchandise and searches on “gucci.” The concordance results are shown in Figure ‎5‑7.

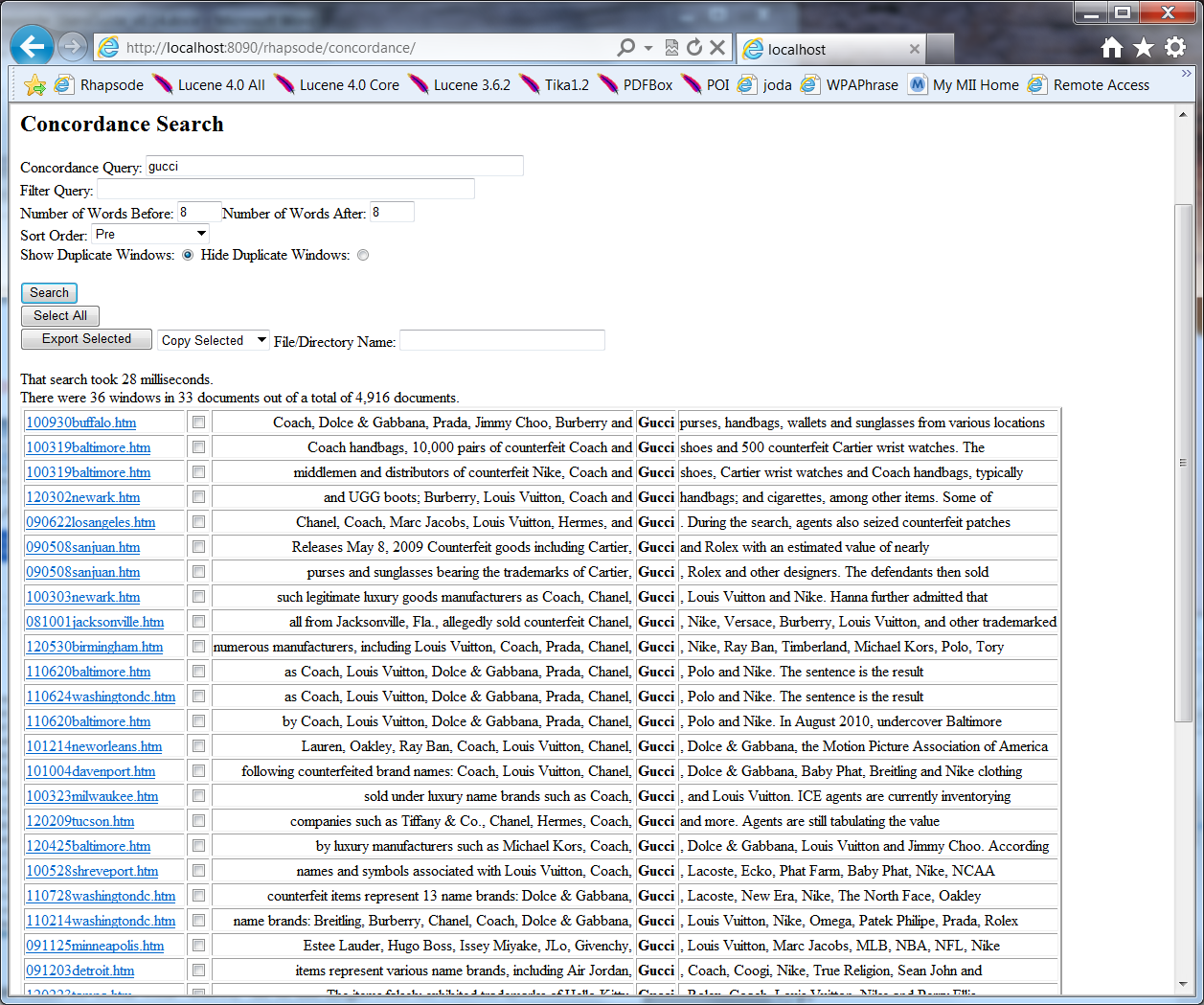


Figure ‎5‑7: Concordance Results for "gucci"

If the user is interested in what terms co-occur with “Gucci,” he or she might read the context to get a sense of the terms, or he could use the Co-Occurrences tool to count terms that appear within a window of the target term. See Figure ‎5‑8 for the results. The results are sorted by a classic (and simple) measure of interestingness, known as term frequency \* inverse document frequency or TF\*IDF. See section ‎5.3.2 below for a discussion of inverse document frequency.

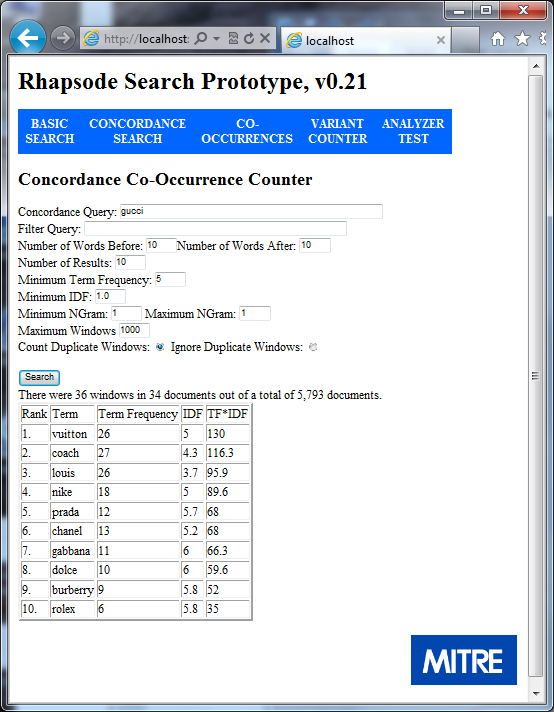


Figure ‎5‑8: Co-Occurrences with "gucci"

As with the Concordance Search, the user selects a query, optionally a filter query, the number of words before and the number of words after; and the user may choose to ignore duplicate windows or count duplicate windows as in the basic Concordance Search. However, the options vary slightly for a Co-occurrences search. The user may modify the number of results to see. The user may modify the minimum term frequency of terms for consideration; for example, the user could require that a term co-occur 5 times with the target term before it might appear in the list. The user may modify the minimum IDF; for example, the user may want to rule out common words (those with a low IDF) as potential candidates. The user may modify the minimum phrasal size (known as an “ngram”) and the maximum phrasal size. For example, the user may want to see phrases that have two or more words and fewer than four words; to accomplish this, the user would enter “2” in the “Minimum NGram” box and “4” in the “Maximum NGram” box. As we show in Figure ‎5‑9, it is sometimes useful to capture phrases; this is especially true in non-whitespace languages and other languages that tend to be more phrasal than English, such as Persian.

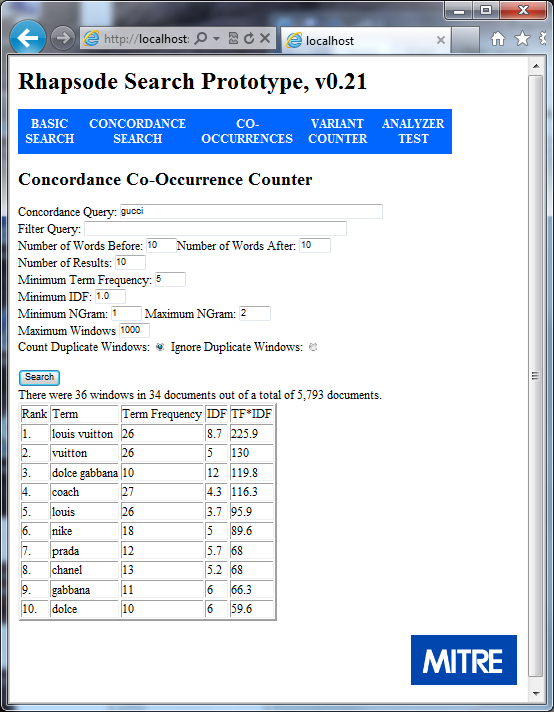


Figure ‎5‑9: Co-Occurrences that are Single Terms or Two Word Phrases

**One main caveat** to this tool is that it is very simple and is only counting words. Users should not rely on its output as if Rhapsode understood what it is reading. I’ve cherry-picked the examples above to show how well this technique *can* work. For some query terms in some corpora, many of the co-occurrences make no sense and/or may be useless or misleading from an analytical perspective. Use the results from this tool carefully!

There are currently two less serious but annoying issues with the phrasal search. First, the technique I implemented does not discount shorter terms after a longer term has been selected. For example, “louis” appears as a unigram even though it has already been counted in the phrase “louis vuitton.” The other annoying issue is that “phrases” can appear that no human would judge to be a phrase, for example, “coach louis”. The solutions to both would require more computational power, and I’ve chosen lightweight for now.

Despite these limitations, this can be a useful tool as part of a larger analytical process.

### Co-Occurrences: Advanced

The minimum IDF option is straightforward if the user is only searching for single terms. However, if the user is searching for phrases (ngrams with *n* > 1), Rhapsode actually computes wgrams.[[2]](#footnote-2) One way to think of a wgram is as an ngram, where there are *n* terms; *w* of them must be above the IDF threshold, and the phrase must neither start nor end with a word below the threshold. In other words, a wgram can neither begin nor end with a term that has lower than the threshold IDF; for example, if the minimum IDF were 1.0, then “bank of” would probably be rejected as a wgram because “of” typically has a lower IDF than 1.0. Further, a wgram does not count internal words that fall below the IDF threshold towards the *n*. For example, again with an IDF of 1.0, “Bank of America” would be counted as a two word phrase or a bigram.

If your index has multiple languages, this simple TF\*IDF method can yield poor results. For this purpose, if possible, it is better to construct a separate index for each language.

## Variant Counter

### Variant Counter: Basic

Behind the scenes, Lucene expands fuzzy queries (for example: f?x) into the actual terms in the index. It is often helpful to see how Lucene is expanding the queries. This is also a helpful way to learn about spelling variants in your index. If you are only interested in the variants of a single term, click on the “Simple Single Term Search” option. This is the fastest option, and it will return document frequencies. If the user is interested in searching for phrasal variants, or if the user wants the term frequencies in addition to the document frequencies, select “Advanced Search.”

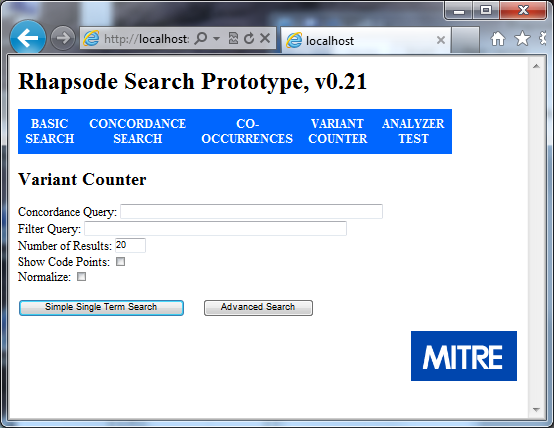


Figure ‎5‑10: Variant Counter Main Window

In Figure ‎5‑11, we show the results for a fuzzy query of “mohammad.” The output of the variant counter also includes the number of documents in which the term appears, and it includes the inverse document frequency (IDF).[[3]](#footnote-3)

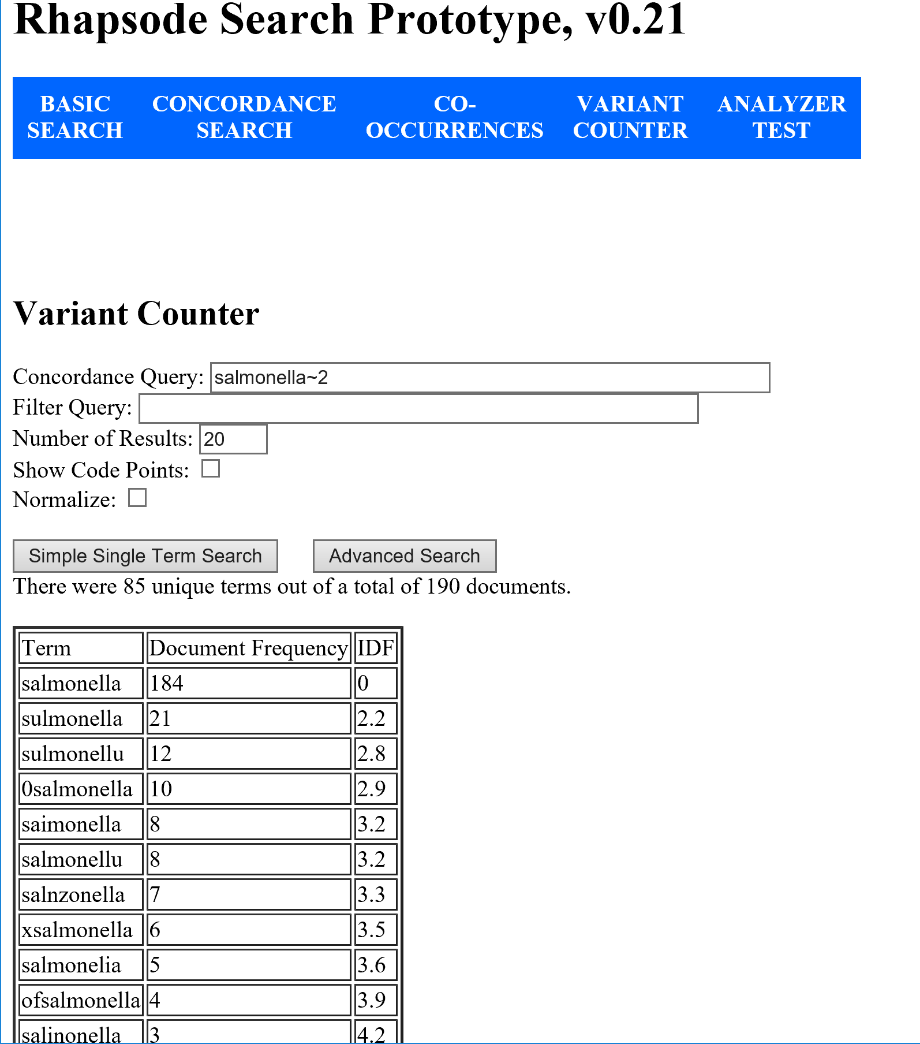


Figure ‎5‑11: Results from the Variant Counter

In the above example, if the user is satisfied with how the fuzzy term is being expanded, he may use the fuzzy term as it is in the query. However, sometimes a fuzzy term captures more variants than the user intends, and in this case, the user may copy only those spelling variants that are relevant to the query and then use those variants.

### Variant Counter (Advanced)

As described above, the “Simple Single Term Search,” works on only a single term, and it returns only the document frequencies for the variants. If the user selects the “Advanced Search” button, Rhapsode will run a concordance search in the background and count the variants of the target. In Figure ‎5‑12, we show the results from that type of search. Notice that there are different capitalizations. If the “Normalize” checkbox is not checked, Rhapsode will return the literal strings as they appear in the documents (Figure ‎5‑12). If the “Normalize” checkbox is checked, Rhapsode will normalize the results, as we show in Figure ‎5‑13.

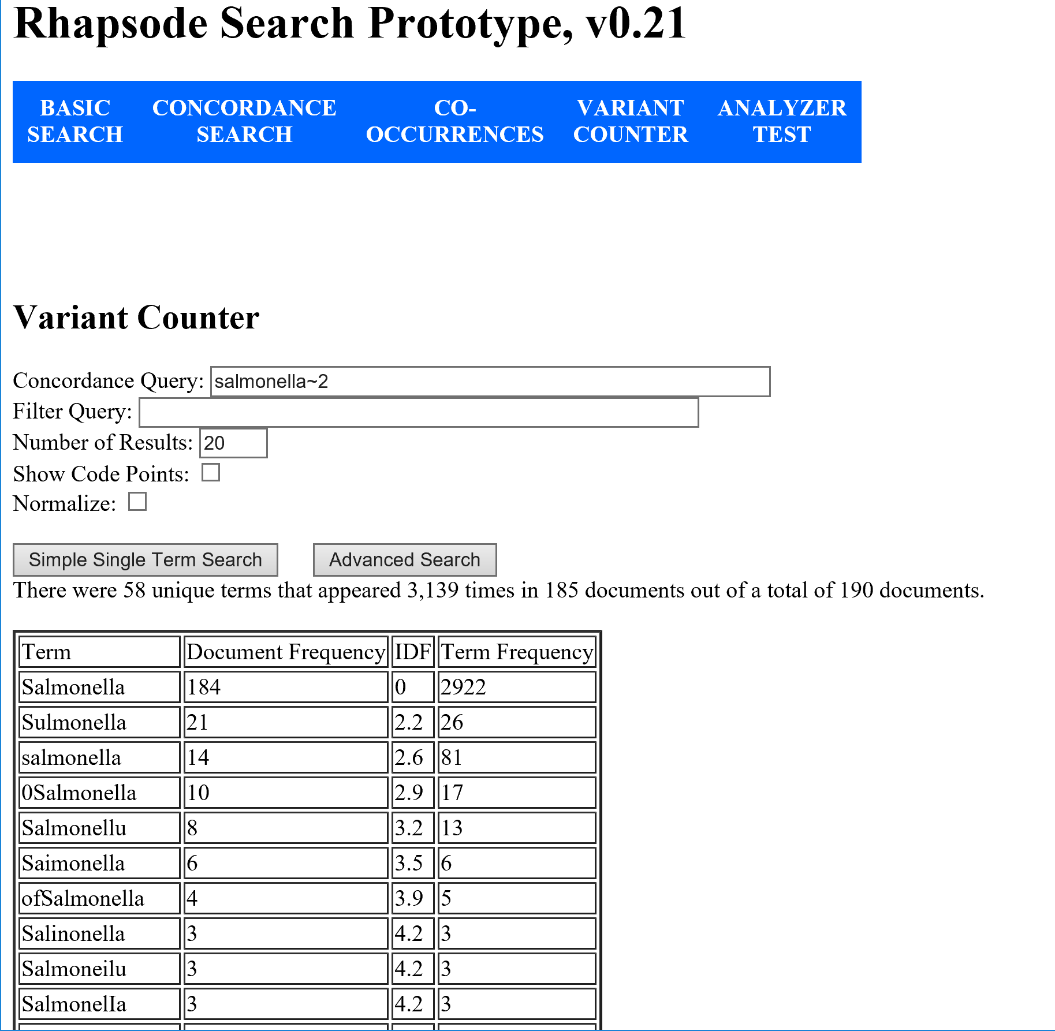


Figure ‎5‑12: Results from the "Advanced Search" Button in the Variant Counter

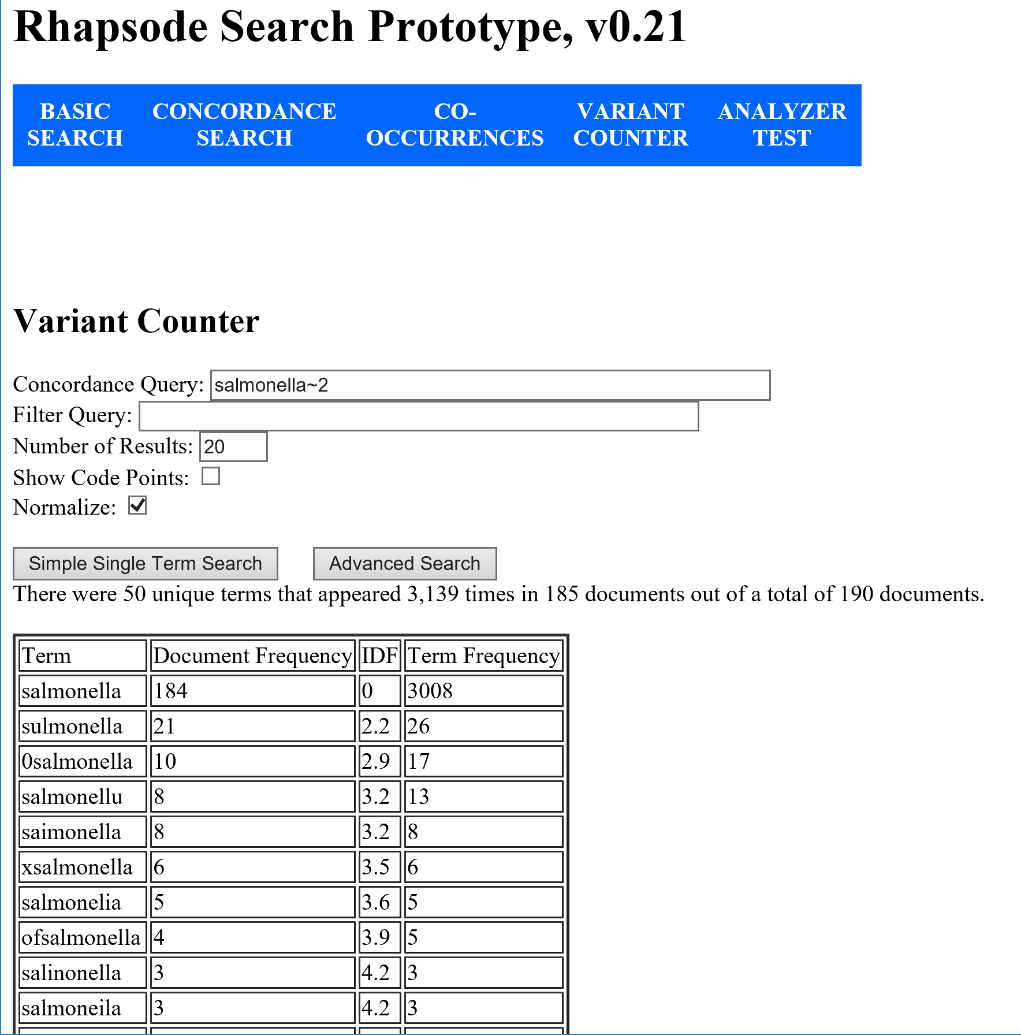


Figure ‎5‑13: Results from the "Advanced Search" Button with “Normalize” Checked in the Variant Counter

If the user is interested in searching for phrasal variants, such as (find a variant of “Adam” within three words before “Barguthi”), the user would enter [adham~1 barguthi~2]~>3, and then click the “Advanced Search” button. As we show in Figure ‎5‑14, there are two variants. In the first, the first name has no “h” and a middle name, and in the second variant, the first name has an extra “h” and no middle name.

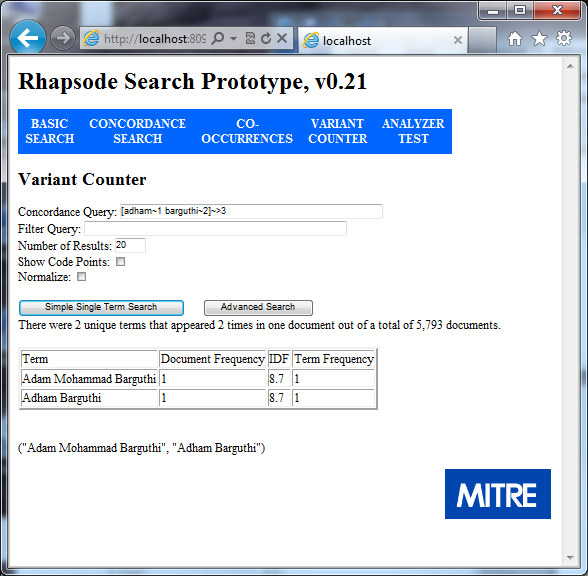


Figure ‎5‑14: Example of a Phrasal Variant Search

Finally, the user may be interested in seeing variants that may not be visible in the presentation form. In Figure ‎5‑15, we show the results of searching for the Persian word for “America” in a fairly large corpus. Note that we have entered a single non-fuzzy term in the query box. We have not selected “Normalize,” and we have selected “Show Code Points.” The results show that there are 17 different combinations of Unicode code points for the word “America.” If the indexer and the query parser had not used Unicode normalization, there would have been only one “variant.” The user is also able to see the exact Unicode code points that appear in the text.

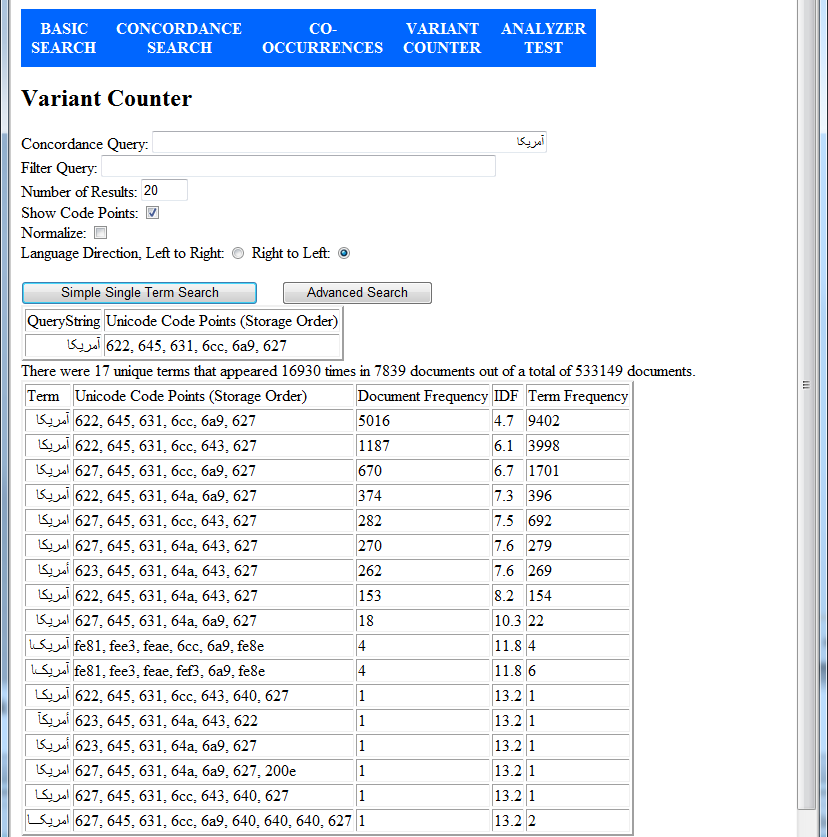


Figure ‎5‑15: Unicode Variants of "America" in Persian

# Query Parser Syntax

As of version 0.30 of Rhapsode, there is one query syntax available in both the main query window and the filter query window. In earlier versions, there were subtly different parser syntax options in the different windows.

## Query Parser Syntax

A useful tutorial for this syntax is available here: <http://lucene.apache.org/core/old_versioned_docs/versions/3_0_0/queryparsersyntax.html>

In the following, we show some of the highlights.

### Key word Query

Find documents that contain “mississippi”:

mississippi

### Fuzzy Query

Find documents that contain a term that differs by two key strokes from the target term:

mississippi~2

### Wildcard Query

Find documents that contain a term with wildcards: “?” means any one character and “\*” means zero or any number of characters.

Find documents that match “mississippi” or “missippi”

miss\*ppi

Find documents that match matches only “missippi”

miss?ppi

### Boolean Query

Find documents that contain “mississippi” but don’t contain “barbecue”

+mississippi -barbecue

Find documents that contain “Memphis” or “Mississippi” and “barbecue”

+(mississippi memphis) +barbecue

### Phrasal searches

Find documents that include a phrase such as “mississippi barbecue”

"mississippi barbecue"

Find documents that include “Mississippi” within 5 words of “barbecue.” The term “barbecue” could appear before or after “Mississippi.”

"Mississippi barbecue"~5

### Range Searches

If your input documents are in directories that identify the year and month for the documents (Figure ‎6‑1), you can search for a range of dates.

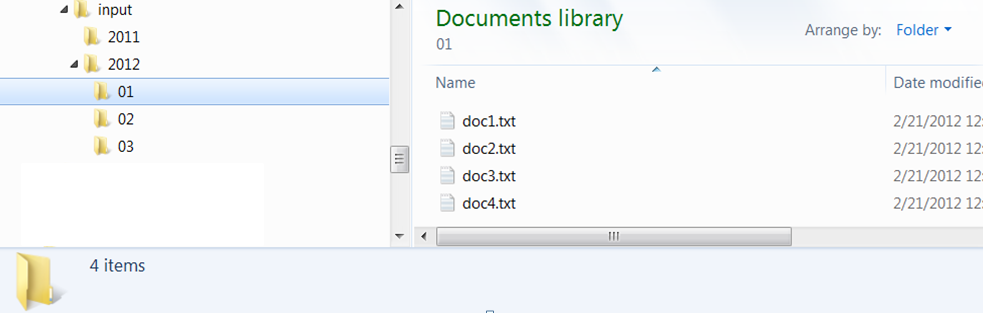


Figure ‎6‑1: Directory Structure with Optional Date Hierarchy

For example, to find “barbecue” documents ranging from June, 2011 to February, 2012, the query would be:

+barbecue +date:[201106 TO 201202]

The “+” signs mean that both have to be true. Make sure that “TO” is capitalized in the range search.

### Regular Expression Searches

In addition to the fuzzy search and the wildcard search, the concordance query parser can parse term-level regular expressions. With regular expressions, users can search with finer grained detail than with wildcard searching. For example, find “cat” or “cut” but not “cot”:

c[au]t

A quick tutorial for regular expressions is available here: <http://www.regular-expressions.info/quickstart.html>. For a full tutorial on regular expressions, see (<http://www.regular-expressions.info/tutorial.html>). Note that the regular expressions only work on a per term basis within the Lucene framework. It is not possible to use regular expressions to search for one term near another. For example, this will not work:

/mississippi(\W+\w+){0,5}\W+barbecue/

### Directional Phrasal Searches

In the traditional Lucene query parser syntax, it is not possible to require directionality if one is using slop. We have added syntax to allow the user to require that one term come before another. For example, find “barbecue” within 5 words **after** “mississippi:”

"mississippi barbecue"~>5

### Recursive phrasal searches

Find a fuzzy match on “Mississippi” within 5 words of a regular expression for “barbecue”:

[mississippi~2 /barbecue|bbq/]~5

To capture recursive phrasal searches, we have added square brackets as an optional way to mark the beginning and the end of a phrase. For example, find “Mississippi” or “Memphis” within 5 words of “barbecue,” and that match should occur 10 words before “competition:”

[[(mississippi memphis) barbecue]~5 competition]~>10

### Important Caveats

* As of version 0.20, it is now possible to specify a negative filter query. To find all documents that do not contain “miami”, type the following in the filter query window.

NOT miami

“NOT” must come first, and it must be fully capitalized.

* Behind the scenes, Lucene expands the wild card and fuzzy queries into matching terms from the index and then creates a Boolean query. For example, if the query is “f?x”, Lucene will search the index for terms that match that query, and it will convert this wildcard query to the Boolean query "fax" or "fox". In a large index, if a user searches for "a\*", the Boolean query would contain thousands of terms. In Lucene’s documentation, the limit on the number of terms in a query is 1024. If the expanded query contains more than that number of terms, you should receive a parse exception error message. In one test we ran, however, with 1500 terms, we did not receive the error.

1. Source: <http://www.mitre.org/about/> [↑](#footnote-ref-1)
2. The “w” stands for Wilson, as in George V. Wilson, my colleague who explained this idea to me. [↑](#footnote-ref-2)
3. The formula for IDF in Rhapsode is ln((TotalNumberOfDocuments+1)/NumberOfDocumentsWithTargetTerm). I add one to the total number of documents so that IDF is never 0. [↑](#footnote-ref-3)