Query Builder & Summary Tool Users Guide

|  |  |
| --- | --- |
|  | Query Builder Tool |
|  | Summary Tool |

# Disclaimer

The tools presented in this guide are in the early stages of development and should be treated only as a prototype for proof-of-concept. The core functionality is the integration of open source off-the-shelf methods/code/etc. for the purposes of collection processing. As such, the developers cannot guarantee the integrity of dependency packages.

# How to use this software users guide

This guide is meant to serve as a light intro into the interfaces of the Query Builder and Summary Tools. It is only an introduction to using the GUI and does not serve as a reference for the backend code. It is recommended that new users read through the guide before attempting to use the GUIs.

# Contents

### Disclaimer ……………………………………………………………………………………….. 1

### How to Use the Software User’s Guide …………………………………………………….. 1

## Section 1 – Query Builder Tool

### What is RegEx? ………………………………………………………………………………….. 3

### Launching the Query Builder Tool …………………………………………………………... 3

### Query Builder UI Basics ……………………………………………………………………….. 4

### Query Builder Example Usage ………………………………………………………………... 5

## Section 2 – Summary Tool

### Launching the Summary Tool ………………………………………………………………... 11

### Summary Tool UI Basics ………………………………………………………………..…….. 12

### Summary Tool Example Usage ……………………………………………………………….. 15

## Appendices

### Known Issues ……………………………………………………………………………………. 19

### Limitations ………………………………………………………………………………………. 20

### Suggested Future Work ……………………………………………………………………….. 21

### Other Areas of Potential Improvement …………………………………………………….. 23

# Query Builder tool

The query builder tool is designed to simplify the process of creating complex interdependent RegEx patterns. The tool allows queries to be constructed in an additive manner and exported for use in the Summary Tool, or saved for later use. The current version of the Query Builder Tool is only a prototype, and should be used with that knowledge. Known issues are listed in Appendix A1. Additional bugs/issues can be reported to [zach.welz@gtri.gatech.edu](mailto:zach.welz@gtri.gatech.edu) using the subject line “Query Builder Tool Issue”.

## What is RegEx and how is it used?

Regular Expressions, commonly referred to as RegEx or RegExp, are text strings that describe patterns. These expressions allow code to rapidly identify instances of text patterns even in large text. The Summary Tool uses RegEx to find matches within the collection and reduce the overall collection into a more appropriate composite document that can be used to effectively identify key words/phrases and summarize information.

While RegEx patterns are extremely useful, they are not straightforward or easy to interpret without training. To reduce this inconvenience, the Query Builder Tool makes the process simpler.

There are many references online about the complexities of RegEx strings for the interested reader.

## Launching the Query Builder Tool

The tool is not designed to be a standalone executable, so it must be called within a python process.

1. Open shell
2. Start python interpreter (“python”)
3. Import the tool (“from milnlp.gui import query\_builder”)
4. Call the run function (“query\_builder.run()”)

## Query Builder UI Basics

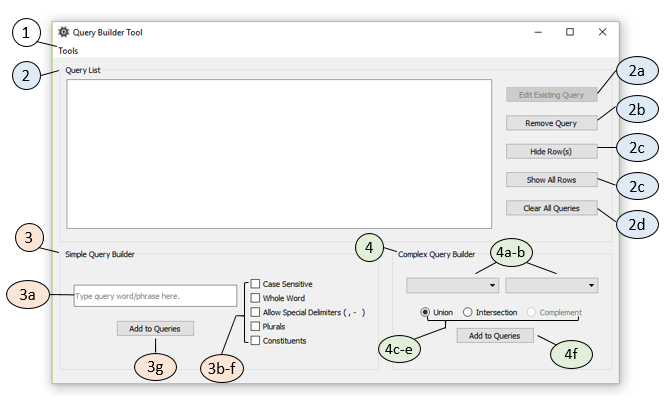


Figure 1 – Query Builder Tool with numbered UI elements

The Query Builder interface is shown in Figure1 with each element of the UI labeled. The elements are briefly described below:

1. Tools: General UI tools including Save, Load, and Refresh
2. The Query List: Display/manage all active queries
   1. Button to initiate query editing (not enabled in prototype)
   2. Button to remove a selected query
   3. Button to hide selected row(s)
   4. Button to reveal any (and all) hidden rows
   5. Button to clear all queries from the active list (does not modify file)
3. Simple Query Group: Used to specify individual patterns for the query
   1. Phrase box where the text of interest is entered
   2. Case Sensitive option
   3. Whole Word option: only matches phrase with non-words around it
   4. Special Delimiters option: allows multi-word phrases to be separated by delimiters other than space
   5. Plurals option: Allows phrase to also end with an ‘*s*’
   6. Constituents option: Allows multi-word phrases to also match any constituent words
   7. Button to add phrase with options to the Query List
4. Complex Query Group: Used to combine two queries (simple and/or complex)
   1. Left query selection: Used to specify the first query to use in the combination\*
   2. Right query selection: Used to specify the second query to use in the combination\*
   3. Union Query: Takes matches if matched from left OR right query
   4. Intersection Query: Takes matches if matched from left AND right query
   5. Complement Query: Takes matches if matched from left but NOT right query
   6. Button to add complex query object to the Query List

The sections of the tool outlined above are each covered in the following example.

\*the order does not matter unless the complement query is implemented

## Query Builder Example Usage

The example below walks through an arbitrary use of the Query Builder Tool, and describes the available options.

### First Steps

The query tool is used to create both simple and complex queries. The eventual usage of the queries is to reduce the amount of text from a corpus of documents so that the information can be quickly and more accurately summarized for topics of interest. As such, the query builder tool will typically be used to construct queries that are related (i.e. Open Systems Architecture topics, hypersonic weapons, machine learning). The queries constructed using this tool will be saved and loaded into the Summary Tool to perform collection processing.

The first step to using the tool (after launching) is to begin constructing the lowest level (simple) queries that will be needed.

### Constructing Simple Queries

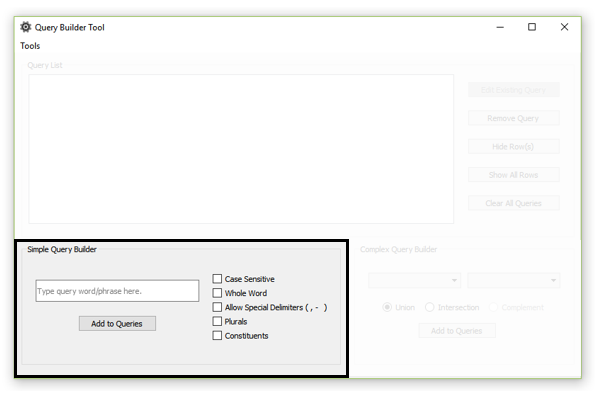


Figure 2 – Query Builder Tool with focus on the Simple Query Builder section

The Simple Query Builder section of the GUI (see Figure 2) is where phrases of interest are added. These phrases (with options) are the building blocks of the final query that will be applied to the collection.

In this example, let’s assume that we have a large collection of documents that are generally related to graphic novels, and let’s pretend that we want to know what our collection says about superheroes and villains. To make matters more complicated, let’s say that the collection is from multiple authors, writing styles, etc. We wish to construct several queries that will help extract out information on superheroes, villains, and specific heroes that we like best.

The first step might be to construct a few simple queries to reduce the original collection. We know that DC superheroes are the best, so let’s create a query for “DC”. The letters “DC” are an acronym (for Detective Comics), but no one uses the old phrase so we want to match the acronym “DC” exactly. To do this, we would enter *DC* into the phrase box and choose the options “Case Sensitive” and “Whole Word”. The Case Sensitive option will match “DC” but will not match “dc”; this option is often used for acronyms and pronouns The Whole Word option will only match “DC” if it is not contained within a larger word; it will match “DC” in “DC Comics”, but will not match “dc” in “handcuff”. Now let’s add the query to the list.

At this point, the query would match any occurrence of DC in the collection, but we are only interested in DC superheroes and villains. Let’s add another simple query for each. Enter the phrase “super hero” in the phrase box. In this case we have to consider whether or not we want to match any heroes in our collection or only ***super*** heroes. If we cared about anything that is “super” or a “hero” than we could choose the “Constituents” option in the GUI. This would match the entire phrase, or any word within the phrase. We decide that we only care if something is super and a hero so we choose to not allow constituents. However, during this consideration, we realize that we want to match both “super hero” and “superhero”. To do this, we will have to create two separate simple queries. If we also wanted to make sure we catch instances of “superheroes” we would need three separate queries, and so on. This process of making queries exhaustive can become long and is avoided for this example. Going back to our original interests, we also want to match the phrase “villains” so that we can see who our superhero is fighting; enter “villain” into the phrase box. In this case, we may want to allow our query to match instances where multiple villains are mentioned. Since the plural of “villain” is “villains”, we can select the “Plurals” option, which will match to both (note: this could also have been achieved using two queries). Click “Add to Queries” to add to the Query List.

Lastly, let’s consider that we want to know about a specific city where our superheroes may exist. In this case we want to know about Dayton Ohio, home of the hero “Black Alice”. Add “Dayton Ohio” to the phrase box. In this case, “Dayton Ohio” often comes separated by a comma or a space. We want our query to match either instance, so we will select the option “Allow Special Delimiters”, which will allow words within a phrase to be separated by a comma, hyphen, or space. Don’t forget to click “Add to Queries”.

At this point, our Query List should look similar to the one in Figure 3.

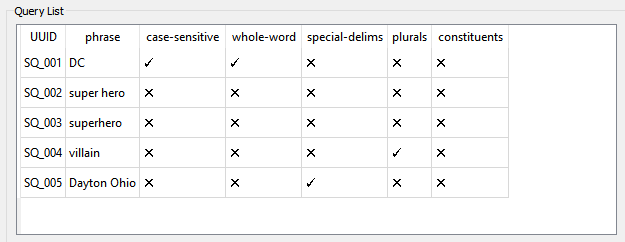


Figure 3 –Query List for example problem

We could use any of these to make a single query on the collection, but we want to make more complicated queries using combinations of the simple queries. To do this, we will need to use Complex Queries.

### Constructing Complex Queries

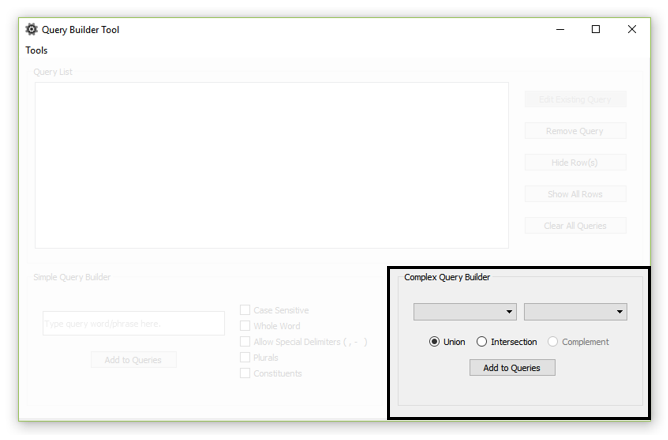


Figure 4 - Query Builder Tool with focus on the Complex Query Builder section

The Complex Query Builder section of the GUI (see Figure 4) is where queries are merged. Complex queries allow users to combine simple queries (and other complex queries) to improve the results when processing a collection. There are three types of complex query that are based on how matches are merged. An example of each is shown in Figure 4 below.

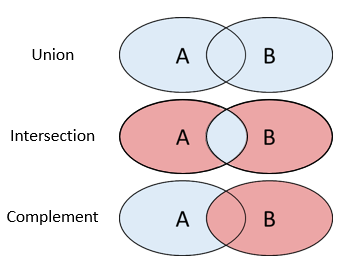


Figure 5 – Example of each type of complex query (blue is included, red is excluded)

The union of two queries accepts all the matches from query A **or** query B. The intersection of two queries takes only matches that are found in query A **and** query B. The complement of two queries takes all the matches from query A that are **not** matches in query B. Together, these operations can be used to filter down matches to very specific query requests. The following paragraphs apply these concepts to the superhero example.

In the previous section where the simple queries were constructed, we have 5 simple queries of interest: DC, super hero, superhero, villain, and Dayton Ohio. When constructing complex queries, it is often best to start with the simple decisions first. We obviously want to combine matches that are found for “super hero” with “superhero” since they are essentially the same query. We will choose SQ\_002 and SQ\_003 in the complex query dropdown boxes, and select “Union” (should be selected by default) before clicking “Add to Queries”. This will create CQ\_001, which is the operation “SQ\_002 **or** SQ\_003”. Another obvious operation is for villains; we also want to add any matches for villains to our query, which is simply another Union operation. This time, we want to add the villain query (SQ\_004) to our complex query (CQ\_001), which is essentially the union of SQ\_002, SQ\_003, and SQ\_004. To do this, select CQ\_001 in the left dropdown and SQ\_0004 in the right dropdown. Union should still be selected, so click “Add to Queries”. Now there should be an entry for CQ\_002, which is the operation “CQ\_001 **or** SQ\_004”.

When we started building our simple queries, we said that we wanted to know about superheroes and villains, but we only care about superior DC characters. This means that we want to match all of the instances of superheroes/villains, but only if the document also contains a match to DC. To do this, we create a complex query that is the intersection of CQ\_002 and SQ\_001. After clicking “Add to Queries”, this will create CQ\_003, which is the operation “CQ\_002 **and** SQ\_001”.

At this point, we decided that we don’t care about Dayton Ohio matches anymore, so we want to remove the query from our list. To do this, highlight the entire row for SQ\_005 and click the “Remove Query” button. Let’s go ahead and hide SQ\_002 while we’re at it so we can see all our complex queries at once; highlight the row and click “Hide Row(s)”. The Query List should now look similar to Figure 6. We are happy with the queries we have made, so let’s save them to a file so we can use them in the collection tool. To do that, click Tools -> Save As, navigate to a memorable directory, give the file a good name, and click save. The query file should be ready for use in the Summary Tool!

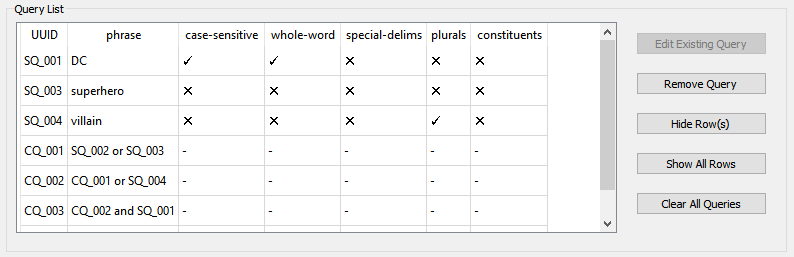


Figure 6 – Query List after adding complex queries

### An Aside to the Example

In the previous example, the order in which the complex queries were constructed was very specific. Rather than creating the intersection between the DC and super hero queries first, we created the unions. This is because the order of operations is important; that is to say, the following two operations are not equivalent:

* (SQ\_001 **and** SQ\_002) **or** SQ\_003
* SQ\_001 **and (**SQ\_002 **or** SQ\_003)

Care should be taken to ensure that at each complex query, the order of operations makes sense, and will accurately return what is expected. In practice, it is best not to make queries too complex or the results may be difficult to predict.

### Market Research Example

The following provides a Query List that can be copied and used for the Market Research dataset as part of the EW/EMS ECCT efforts. The file containing these queries will be used in the Summary Tool Collection Processing example.

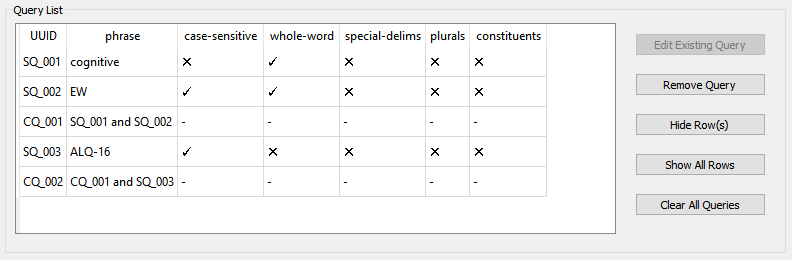


Figure 7 – Example queries used in Summary Tool example

# Summary tool

The Summary Tool is designed to assist users that wish to summarize a collection of documents using a query created with the Query Builder Tool. The current version of the Query Builder Tool is only a prototype, and should be used with that knowledge. Known issues are listed in Appendix A1. Additional bugs/issues can be reported to [zach.welz@gtri.gatech.edu](mailto:zach.welz@gtri.gatech.edu) using the subject line “Summary Tool Issue”.

## Launching the Summary Tool

The tool is not designed to be a standalone executable, so it must be called within a python process.

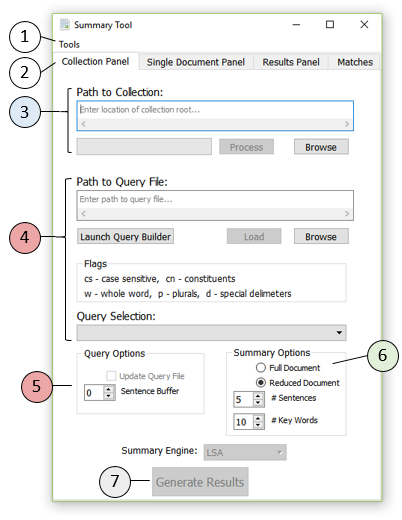
1. Open shell
2. Start python interpreter (“python”)
3. Import the tool (“from milnlp.gui import summary\_tool”)
4. Call the run function (“summary\_tool.run()”)

Note: Executing the program in this manner comes with a slight caveat. If the Query Builder Tool is launched using the Summary Tool button and the Summary Tool is closed before the Query Builder Tool, it will leave a ghost process running in the shell. The Query Builder Tool can still be used, but closing the window will not kill the process. In order to escape this, you **must** ctrl+C the shell after exiting the Query Builder Tool. This is a bug associated with launching python subprocesses.

## Summary Tool UI Basics

The Summary Tool consists of 4 panels: Collection Panel, Single Document Panel, Results Panel, and Matches. Each panel has a specific use and will be described in the following sections.

### Collection Panel



The primary purpose of the Summary Tool is to help identify critical information across large document collections. To this end, processing large collections and summarizing based on created queries is the primary capability that the collection panel provides.

Figure 8 - Summary Tool Collection Panel

The first step in using the collection panel is specifying a path to the collection or sub-collection. This can be done by pasting a path into the collection path box, or browsing for a directory (Figure 8, Label 3). Once the path is entered, the “Process” button will activate and can be clicked. This process action analyzes the entire collection, parses PDF files into raw text, pre-processes the text, and extracts out useful metadata. This allows the Summary Tool to analyze the contents of files much faster than parsing on-demand. If applied to existing large collections, this step will take a significant amount of time to complete. Once the collection is processed, it can be analyzed for information of interest (subsequent initializations of the collection will also be much faster since the bulk of the time is due to the speed of parsing).

The next step is to load a query file containing query objects. If the user has already generated a query file it can be specified by pasting the path into the query path box, or browsing for a compatible file (Figure 8, Label 4). If the user has not created a query file yet, they can launch the query builder with the available button. Once a query file is specified, the “Load” button will activate and can be clicked. This action loads the query file and populates the Query Selection dropdown giving the user the ability to select the query they wish to apply to the collection. To reduce clutter in the dropdown box, shorthand notation for the queries is applied and can be compared to the “Flags” label above the dropdown menu. This can be used to help identify which query was desired.

Once the collection has been processed, and the query has been selected, the user will be able to click the “Generate Results” button (Figure 8, Label 7) using the default configuration options (Figure 8, Labels 5 & 6). These options can be changed to modify the method that the results are generated, and are described below:

* Summary Options
  + Document method: Full vs Reduced
    - Full Document: for a query match, the entire document that contained the match is added to the composite document used for analysis (this method is NOT recommended)
    - Reduced Document: for a query match, only the specific pages containing a match are added to the composite document used for analysis. This vastly reduces the processing time.
  + # Sentences: The number of sentences to return for the generated summary
  + # Key Words: The number of extracted key words/phrases to return for the composite document
* Query Options:
  + Sentence Buffer
    - Only applicable with summary option “Reduced Document”
    - Adds additional sentences from previous and subsequent pages of each match to ensure that information near the page limits isn’t lost
    - A sentence buffer of 0 or 1 is recommended
* Greyed-out Options:
  + Update query file: this method adds matches to the query file so that they can be saved for later reprocessing. The option is removed from the prototype due to bugs.
  + Summary Engine: The default unsupervised summary engine is the Latent Semantic Analysis (LSA) summarizer. It is widely accepted as the baseline industry standard summarizer for natural language corpora. Other engines exist in the codebase, but are not currently implemented in the front-end GUI.

Once the “Generate Results” button is activated, the backend will go through the process of finding matches to the specified query, creating a composite document, extracting key words/phrases, and generating a summary. The process is of polynomial time complexity, and therefore takes significantly longer for longer composite documents; thus, a good query that adequately reduces the size of the overall collection is recommended.

### Single Document Panel

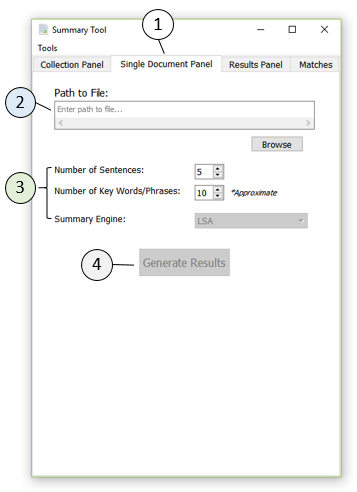


Figure 9 – Summary Tool Single Document Panel

With the limitations on the collection panel due to bugs and time complexity issues, it is often equally important to be able to quickly generate keywords and summaries for individual documents. To accomplish this, the lightweight Single Document Panel was implemented. It is similar to the Collection Panel but only requires the user to specify a path to a single file.

Once a valid file path is specified, the “Generate Results” button will become active. As with the Collection Panel, the summary options for sentence and key word/phrase count are available and provide the same functionality.

While the Single Document Panel does not require a query to create a composite document (the composite document is constructed from the complete file), the complexity of the summarizer is still O(n2) at best, and therefore is slow to process large files. Therefore, it is not recommended to attempt to summarize large files (e.g. >50 pages). Large file topic extraction is currently a big area of research. To accomplish this task, alternative methods that are not provided by the Summary Tool should be investigated.

### Results and Matches Panels

The Result Panel and Matches are two useful ways to visualize different output. The Results Panel generates a lightly formatted output containing information about the composite document, query (if applicable), the generated summary, and the extracted key words/phrases. It can be saved to a plaintext file if desired. The Matches tab of the Summary Tool lists the files that matched the applied query (Collection Panel matches only) and the specific pages that were matched within each file. This is useful information that helps direct the user towards potential locations of interest within the collection. Examples of these results are given in the following section.

## Summary Tool Example Usage

Referring back to the example queries shown in Figure 7, this example will go through the process of using the Summary Tool to extract information from the Market Research document collection.

### Processing the Collection

The market research document collection contains 528 files spanning 148 folders and weighs in at almost 600 MB. Files range from 1 to 2000 pages and are all PDF format. Ideally, files would be processed as they are added to the collection rather than all at once, but the tool can handle processing the entire collection.

To process the collection, the path should be entered into the “Path to Collection” text box on the Collection Panel. The path can be added by using the “Browse” button, and navigating to the market research folder. Once a valid path is specified, the “Process” option will be available. Clicking this will start the process of parsing and preprocessing every file in the collection. Naïve logs are printed to the console to help the user follow what is being done. The first stage is to read the collection and identify the files and folders present. Once that is done, the tool will check to see if the files have already been processed and are up-to-date. If they are, the tool will skip them. If not, a metadata file will be created for each, the PDF file will be parsed to raw text, the text file will be processed, and the metadata file will be updated with critical information. Each stage will be shown in the GUI progress bar. Once the processing is complete, the progress bar will be filled and display “Done!” as shown in Figure 10.

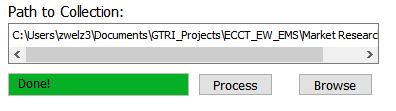


Figure 10 – Example of completed collection processing step

The next step is to load the example queries from their file. This is done by clicking the “Browse” button and navigating to the location of the saved file (or alternately pasting the path into the box). Once a valid path is specified, the “Load” button will become active and can be pressed. This button loads the query file and populated the “Query Selection” dropdown as shown in Figure 11.

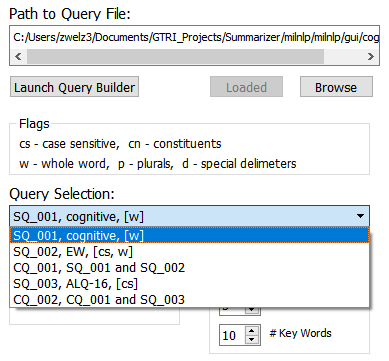


Figure 11 – Example of loaded query file

Here we can see the queries from the Query List populated into the dropdown with shorthand for the simple query options. The “Flags” label above the dropdown serves as a legend for the shorthand. Any of the queries can be selected and applied to the collection, however most will not be constraining enough to reduce the size of the composite document to within acceptable limits. Thus, we will choose CQ\_002 to apply. Select CQ\_002 from the dropdown and the “Generate Results” button should become active. Click it.

The tool does not implement a progress bar for generating results due to issues with the base GUI package. Instead, information about the backend processing being conducted is printed to the console. Users can refer to this to see query matches being made, the construction of the composite document, and the kickoff of the summary. In this case the console should quickly reach the summary stage, which should look similar to:

Using method ‘reduced’ results in: <DOM with 280 paragraphs>\_

It should take just a few seconds to run. When complete, the results should be printed to the console, and the GUI tab should automatically be switched to the Results Panel. The output should look similar to that in Figure 12. The matches that were used to generate the composite document should be added to the Matches tab of the GUI and should look similar to Figure 13.

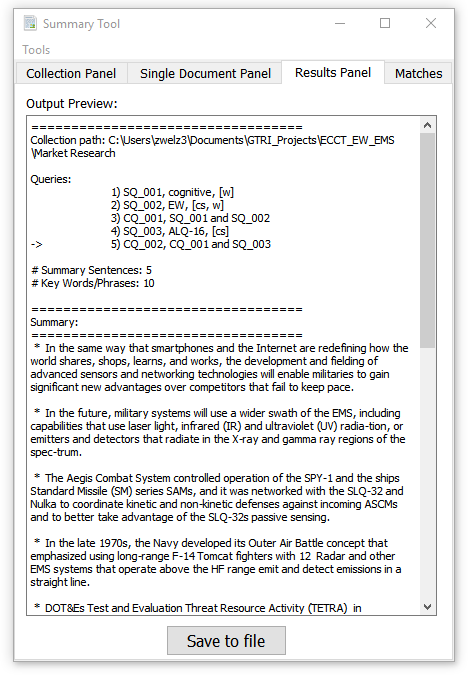
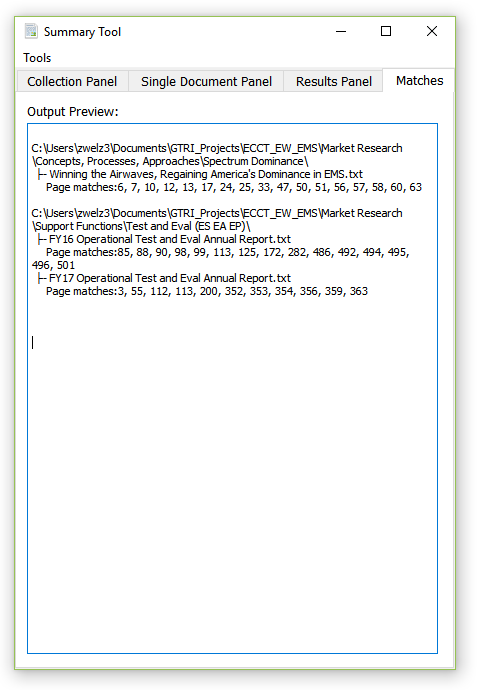
 

Figure 12 – Example of Results Panel Figure 13 - Example of Matches

From the results above, we can see that the 5th query (and its contents) was applied to the collection. The summary was properly generated, and we can see that three matching documents from two directories were used to generate the composite document. The matches (unions only) were found on around 40 pages (note: the matches are returned for the parsed raw text files not the pdfs).

While the exact summary may not be as accurate as a human interpretation of the material, it provides some knowledge related to what the collection says about cognitive EW and related platforms. The generated key words/phrases (not shown in this example) provide some information about the topics present in the composite document and may help narrow down future queries.

### Generating Results for a Single Document

The process for using the summary tool is relatively straightforward compared to processing the collection. A single file needs to be specified in the “Path to File” box. The tool only supports PDF files, so the specified path must be to a valid PDF file. If the PDF has not been parsed, the tool will process the file (although it will not write the results to files like when processing the collection). An example of executing the tool for the single document “*Winning the Airwaves, Regaining America's Dominance in EMS.pdf”* is shown in the figures below.

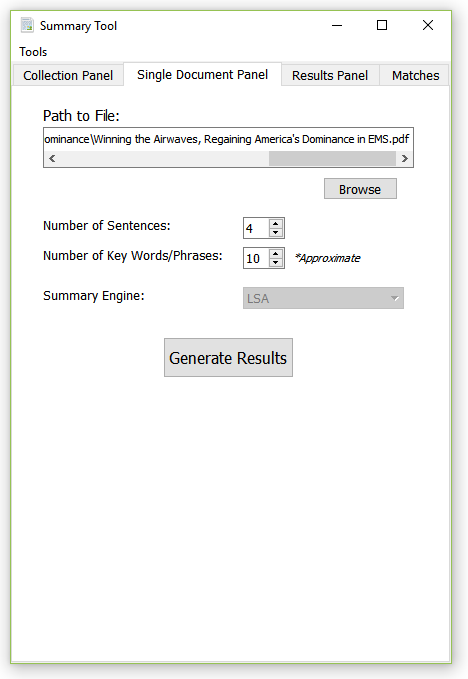
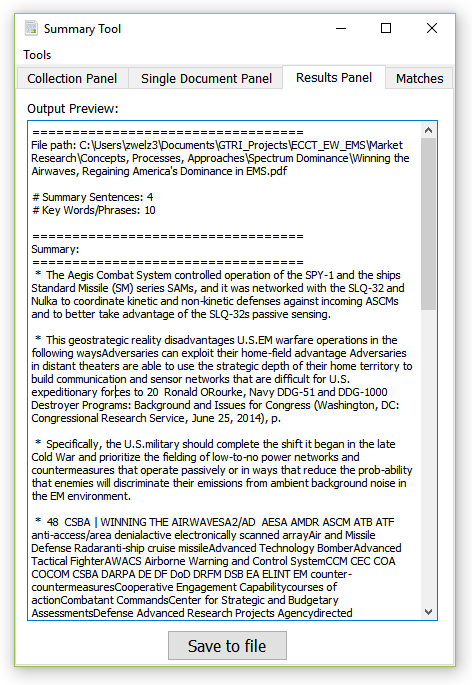
 

Figure 14 – Example of Single Document Panel Figure 15 – Example of Single Document Results

# Appendices

## Appendix A1 – Known Issues

### Summary Tool

|  |  |
| --- | --- |
| Description | Severity |
| When loading a query file, if an existing file was previous loaded into the tool (during the same session) and the current file contains fewer queries, the dropdown box will improperly display extra entries. | Low – Minor visual inconvenience |
| The current raw text processing suite attempts to merge sentences back together that are divided during the PDF to TXT conversion. This is not complex enough for the files being processed, and often results in improperly merged lines. These lines are often very long and cause the summarizer to improperly score them. | High – Impacts value of generated results |
| The current raw text processing suite attempts to break up sentences that exist on the same line to avoid multiple real sentences being added to the same sentence object. This method is not complex enough and results in unexpected sentence fragmentation. | High – Impacts value of generated results |
| The key word/phrase extractor does not have any default lemmatization, therefore words and phrases that should be handled together (i.e. technology and technologies) are being treated as unique instances. | Medium – Increases redundancy in results |
| The GUI becomes unresponsive during some large processing tasks (i.e. collection processing, large composite document summarization). The tool is still running and generating printouts to the console, but is not an active “alive” window. | Low – Minor visual inconvenience |

### Query Builder Tool

|  |  |
| --- | --- |
| Description | Severity |
| Allow Special Delimiters option adds a patters between words to match commas, but not space after comma. I.e. if phrase is “Dayton OH” and the option is selected, “Dayton,OH” will match, but not “Dayton, OH” | Low – Not a common search pattern |
| When removing (deleting) a query from the Query List, a visual table artifact persists. | Low – Minor visual inconvenience |
| When executing the tools from outside the GUI root path, the tool images are not properly loaded (just the icon visuals). | Low – Minor visual inconvenience. |

## Appendix A2 – Limitations

The original intent of this tool was to help conduct market research on large collections of documents, and reduce the amount of manual labor to find information of interest. The goal was to help identify topics within the collection, and summarize sub-collections. It quickly became apparent that off-the-shelf tools were limited in their ability to process the entire collection, which is what caused the requirement for users to specify constraining queries. The following paragraphs discuss the limitations of the current prototype.

Due to the low complexity of raw text processing capabilities, the text version of documents is often not structurally accurate when compared to the original PDF (i.e. paragraphs and sentences). As such, the resulting sentences, key words/phrases, and summary are not as effective as they would be if processing occurred on an accurate text representation.

Since the primary interest was being able to summarize topics of interest from the collection, the tool is centered on the execution of the summary engine. This means that there is no good way to generate query matches of interest without also generating a summary (through the GUI), which can be time expensive. Large composite documents are slow to summarize, and the tool currently implement no method of escaping the summarization process.

The summary engines themselves are limited to the capabilities of the included off-the-shelf algorithms. The GUI utilizes Lex Rank summarization when processing the collection, and Latent Semantic Analysis summarization when constructing a composite document summary. This means that the results inherit the limitations of these algorithms as well.

During initial collection processing, there is no good way to cancel the task without killing the entire python process. This processing stage can take a significant amount of time depending on the size of each document and the number of documents in the collection. While this is an issue, the ability to skip up-to-date files makes it less painful to kill the process and re-execute.

The requirement of queries to filter down the size of the composite document causes several limitations. Queries must currently be reapplied each time the tool is executed due to issues saving query matches in previous versions. The queries must be applied to generate any results for a collection and/or sub-collection. This means there is currently no generalized topic extraction for the entire collection, and no way to summarize multiple document without first applying a query (due to limitations on composite document size).

Lastly, the off-the-shelf methods used to extract key words/phrases and create summary sentences are purely unsupervised learning algorithms. This means that the algorithms assume the user does not know what to look for. As a result, the algorithms attempt to identify valuable information without any understanding of what the user is looking for. A consequence of this is that a user may apply a very specific set of queries to the collection, but the resulting key words/phrases and summary sentences may not have anything to do with the queries (if other topics are stronger in the text that was matches). This is a common issue with unsupervised methods, and (together with the requirement for queries) significantly limits the value of generated summary results.

## Appendix A3 – Suggested Future Work

### Refactor the Raw Text Processing Module

One of the primary limitations of the tool in its current state is the inaccuracy of the PDF to TXT conversion process. The off-the-shelf tools used to make the conversion to not retain sentence/paragraph structure and cause challenges attempting to conduct reconstruction. Naïve attempt to improve this text processing resulted in mixed results for different document types (see next section on document classification), and as a result cause clutter to be returned in the summary (and sometimes key phrases). One of the primary efforts of future work should be identifying a more accurate process for converting PDF to TXT, or implement more effecting processing methods to reconstruct the original document structure after conversion.

### Document Classification

One of the major challenges of the Market Research collection was the diversity of document types that exist. Most off-the-shelf methods are designed to work with natural language documents that are structures similar to a high-school essay. The Market Research collection contains documents that are snippets from the web, pamphlets/brochures, technical documents, military documents, etc. This makes applying one processing methodology to all document types rather inaccurate. To help improve accuracy across collections, some method of document classification should be applied to help identify custom methods of processing for each document. This increases the complexity of the overall tool, but is necessary if the user desires all types of document to be analyzed for valuable information.

### Improved Topic Extraction

While summarization is valuable, it is significantly limited by the off-the-shelf summary engine currently implemented in the tool. What will likely foster more accurate analysis of collections for market research is an improved topic extraction process. This would allow users to identify what information is present across the entire collection (or sub-collections) and build queries according to their needs. Advanced in graph implementations for python would also make it easier to link directories and documents by their topics, thus making is easier and faster to locate relevant information.

### Lemmatization

In natural language, an idea/word/topic/etc. often exists across a collection in many different representations. For example, the idea of “emergent technology” may be represented as “emerging technologies”, “emergent tech”, etc. While these are all the same topic, naïve methods treat each representation as an independent concept. To improve the process of key word/phrase and topic extraction, lemmatization can be applied to merge all representations into a single idea of “emerging technology”. This not only reduces clutter when analyzing topics within a collection, but also helps the summary engines identify connections between topics within each sentence, and would improve the likelihood that summary sentences returned are related to the topics of the sub-collection as well as any applied queries.

### Composite Document Construction

The idea of merging matching text into a composite document for summarization makes sense, but in practice causes several issues. First, the size of this document quickly becomes too large to quickly analyze and summarize. Another issue is that the information in the composite document is no longer structured the way natural language documents are (i.e. the order that information is presented), which can confuse the summarization engine. To improve this process, a more complex composite document should be created that attempt to avoid the current issues.

### Misc. GUI Changes

Since the user primarily interacts with the GUI front-end, there should be improvements to both the Query Builder Tool and the Summary Tool. Some improvements include:

* Ability to edit queries
* Ability to export queries directly to Summary Tool
* Auto-remove queries with dependency on any query that is deleted

Other changes to simplify the interface and improve the ease-of-use should also be implemented based on feedback from users of the prototype.

## Appendix A4 – Other Areas of Potential Improvement

### The following sections list a few additional areas of potential improvement for traceability.

### Core Back-end Code

* Complement Query
  + **Not** operation (see pg. 8)
* Add optional context to query matches
  + Display information about text surrounding match to identify false positives and context
* Add platforms and systems to results
  + ex. F-35 and ALQ-161
* Implement option for platform specific keywords/phrases
  + i.e. “F-35 Radar”
* Threading for PDF->txt conversion
  + Would speed up initial collection processing only
* Skip or alternative processing method for large files
* Testing and logging

### GUI

* Query file update after execution
  + Save time reapplying queries
* Save after a Save As operation

### Misc.

* Improved file management
  + Clean-up metadata files
  + Set collection-wide hidden/not-hidden attribute for parsed files and metadata