**GURU Documentation**

**Purpose:**

This is a document to describe how to support features of the GURU added and modified during the summer of 2014 through the Google Search Appliance (GSA) admin console and Authorit files. For a comprehensive overview of using the GSA admin console, refer to the sites listed below:

Admin Console Documentation: <http://www.google.com/support/enterprise/static/gsa/docs/admin/70/admin_console_help/helpdoc4.html>

GSA Help Center:

<https://support.google.com/gsa/?ctx=global_support#topic=2707841>

For reference to the frontend XSLT/XSL files and scripts used to configure the GSA, visit In-Young Jo’s Guru Stuff Folder here:

[I:\Everyone\In-Young Jo\Guru Stuff](file:///I:\Everyone\In-Young%20Jo\Guru%20Stuff)

**Prerequisites:**

In order to provide the most effective technical support for GURU, a person should have experience with the following:

* **Web Design**
  + Reason: Support for the frontend of GURU requires knowledge of designing webpages and programming.
  + Preferred Languages: HTML, CSS, JavaScript, XML/XSL/XSLT
* **Networking**
  + Reason: The GSA interacts with networks/servers for crawling content, authentication of users, and many other important backend tasks. Understanding networking and how systems can be integrated using active directory is especially important for maintenance of the backend.

A person supporting Guru should also have the following “soft” skills:

* **Passion for Web Design/Programming**
  + Reason: A large part of developing new features for GURU is coding/designing.
* **Good organization skills**
  + Reason: Currently, with the vast number of elements in the frontend XSLT file for GURU’s interface, it is difficult to navigate through the file and change things in an efficient manner. Need to be eager to maintain/improve the organization/design of the frontend file and other components of the system to make maintenance more efficient.
* **Self-motivation and willingness to learn new things**
  + Reason: Not all features of GURU were put in place by simply turning things on and off in the admin console. Adding new/novel features to GURU often requires a lot of learning and research. There are also many features on the GSA for GURU that have not been fully explored and need to be researched in order to realize the full potential of GSA for GURU.
* **Quick response time**
  + Reason: GURU interacts with many other systems at QL. Need to be able to respond to things quickly in order to accomplish task efficiently and in a timely manner.

**Contents:**

**List of Added/Modified Features:**

This section of the documentation will provide further description of each GSA feature in the list below, including information about what the feature is, its intended application in GURU, and how it can be configured.

**Google Search Appliance**

* Result Biasing Policy
  + Source, Date, and Metadata/Entity Biasing
* OneBox Modules
  + Setting Configuration
  + XSL Modification
* Query Settings
  + Adding Expanded Queries
* Related Queries
* Key Match
* Secure Search
  + Universal Login/Kerberos Setup
  + Getting Data from Reports
* Frontend XSLT
  + Changing Banners
  + Adding Helpful Links
  + Adding Collection Filter Links
  + Going from test to production

**Author-It Pages**

* Headers/Footers
  + index.htm
  + webhelp.css

**List of Loose Ends:**

The rest of this documentation will provide information about tasks that were started but not completed and the current view on how to proceed with each.

* **Sharepoint Integration**
  + Setup
  + MBA/AmazeU Training Materials
  + Product Updates
* **Single Sign-On**
* **Highlighting Search Terms**
  + webhelp.js
* **Product Updates**
* **Helpful Links**
* **Dictionary Results**
  + Dictionary
  + Quicktionary

**Result Biasing Policy**

Result biasing policies enable the admin to control the relative strengths of URL patterns, collections, and feed data. The stronger a group of results is made, the higher priority it is given when search results are displayed. Result biasing policies can also give results priorities based on dates and metadata/entities.

In GURU, the majority of the sources for results are collections. Because end users did not want documents within the ‘Legal’ collection to appear before results from other collections, a result biasing policy had to be made to lower the strength of these collections relative to all the other collections searched. A snapshot of the source biasing portion of the result biasing page, which can be accessed by going to **Search > Search Features >** **Result Biasing** and creating a new result biasing policy, is shown in figure 1 below.

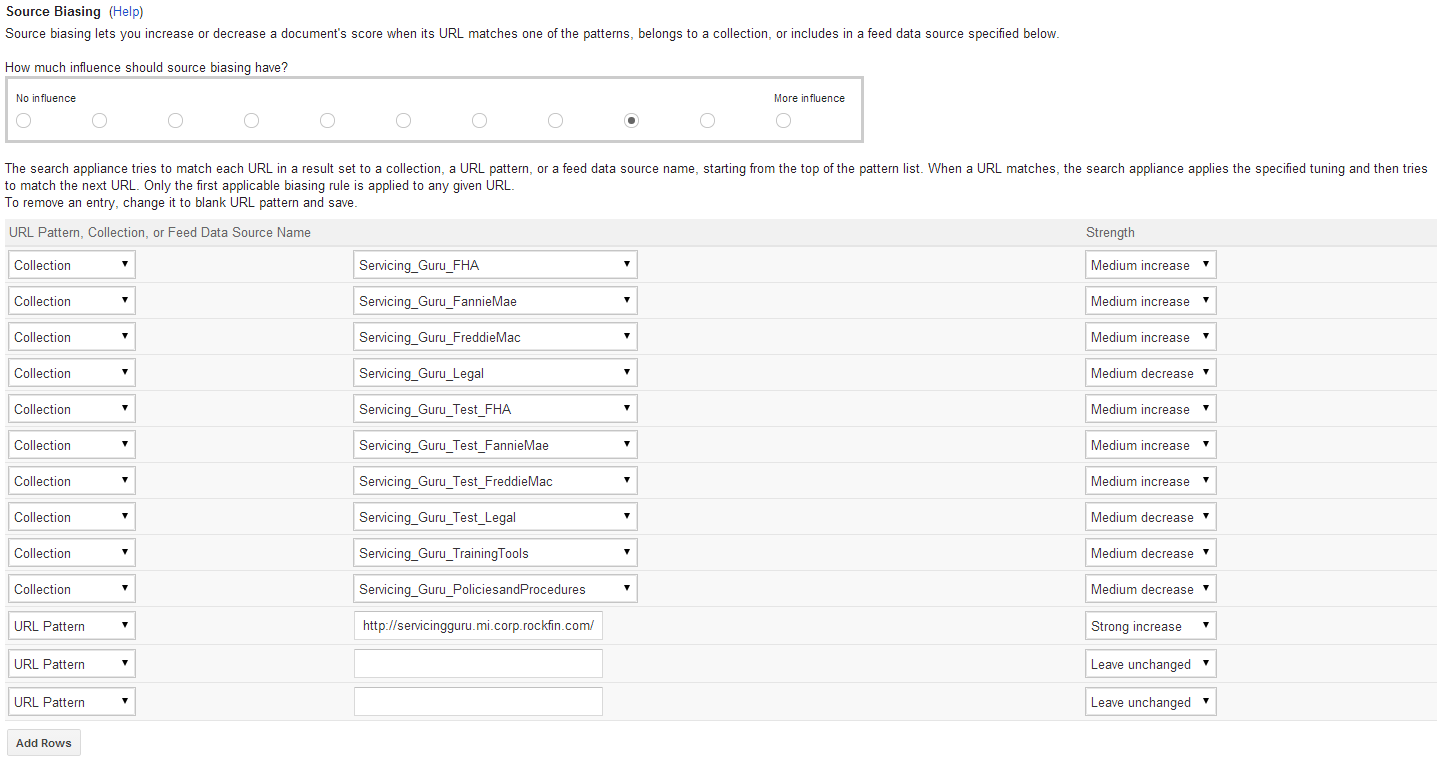


Figure 1: Source biasing interface

The snapshot indicates how the strength of each collection can be set and how the amount of influence source biasing is given overall on the prioritization of results can also be set within this window.

Similar procedures can be used for biasing by date and by metadata or entities within the result biasing page. Once the settings for the policy are configured, the policy can be saved and applied to a front end by going to **Search > Search Features > Front Ends > Filters** and choosing the proper policy for the front end with the Result Biasing Policy drop down menu.

To learn more about Result Biasing Policies, go to the link below:

<http://www.google.com/support/enterprise/static/gsa/docs/admin/70/admin_console_help/serve_scoring.html>

**OneBox Modules**

OneBox modules can give real-time access to data from an external source or another collection on the same appliance. This is useful for displaying results from one collection and results for the main collection being searched at the same time but on different parts of the page.

In GURU, end users expressed the need to see a list of training materials relevant to a search query that would come up beside the normal search results. OneBox modules were used in order to accomplish this, as they were built for this purpose and they are customizable via XSLT. The snapshot in figure 2 below shows the page for configuring settings for a OneBox module, which can be accessed by going to **Content Sources > OneBox Modules** and defining a new module.

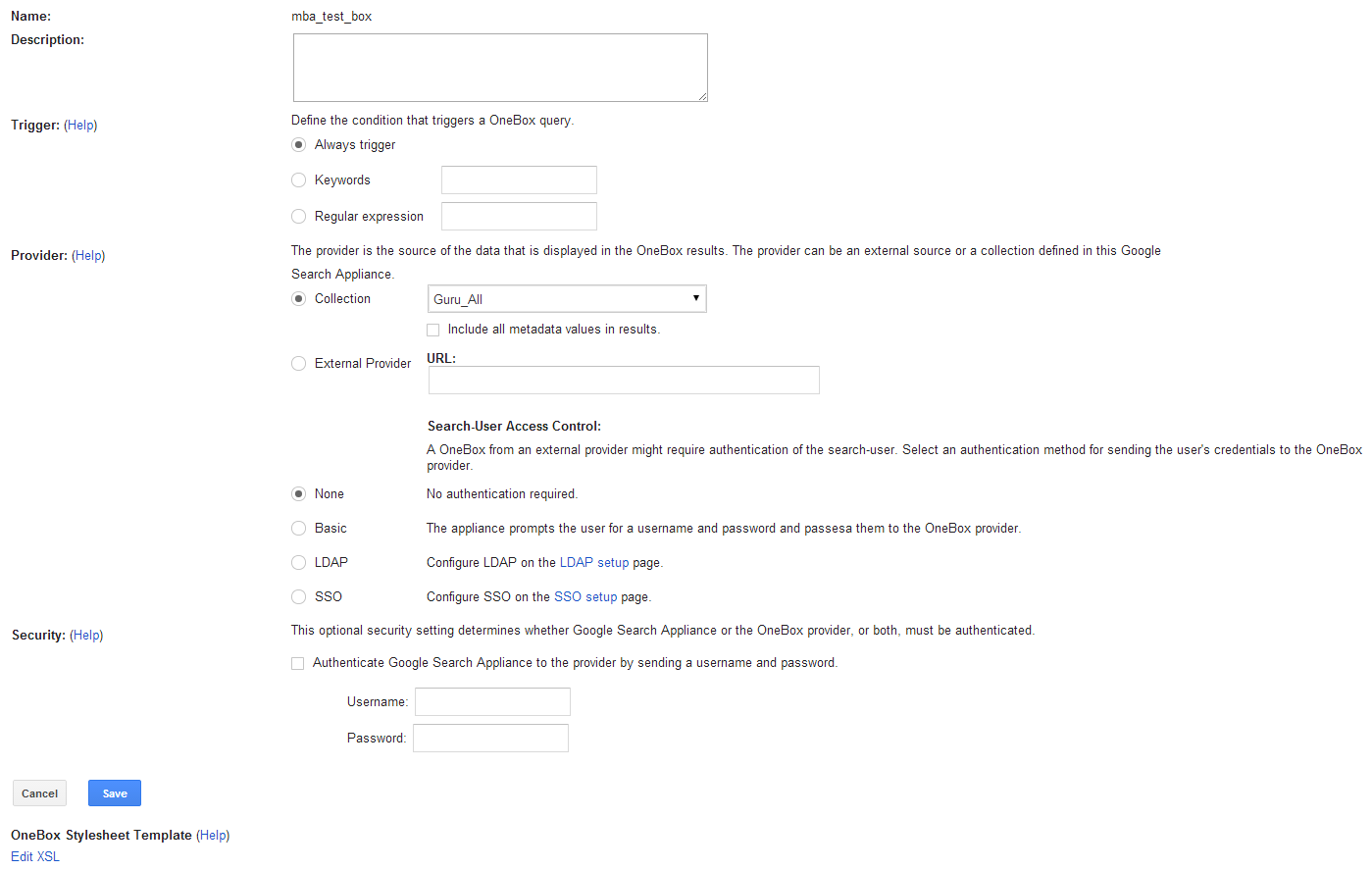


Figure 2: OneBox Module configuration page

From looking at the configuration page for the OneBox module, it is evident that there are a variety of settings that can be controlled. The most relevant settings for accomplishing this task are the “Collection” selection dropdown menu and the “Edit XSL” link at the bottom of the page. The dropdown allows the admin to select a collection for the OneBox to search through. The “Edit XSL” link takes the admin to another page with a text box for XSL code that defines the styling of the OneBox. This XSL can access the CSS styling defined within the frontend XSLT file that the OneBox is applied to. Thus, when designing the OneBox for GURU, styling elements defined in the frontend XSLT file were used in order to make results look like the items in other sidebar elements (which are also a new feature in the GURU interface).

The XSL for the OneBox allows for not only changing the appearance of the results, but also the information that appears for each result. In the underlying XML that is styled by the XSL, the information for each result of the OneBox is held within “MODULE\_RESULT” tags. The “MODULE\_RESULT” tags hold several pieces of information. One item is the “Title,” which is the title of the result. Another item is “U,” which is the URL of the result. The values of these items can be referenced within the XSL using <xsl:value-of select=""/> tags.

Once a OneBox module has been defined with the desired configurations and styling, it can be applied to a specific frontend. This can be done by going to the OneBox Module tab in a desired frontend (**Search > Search Features > Front Ends > OneBox Modules**) and moving the new OneBox module from the “Available Modules” box to the “Selected Modules” box by using the arrows in between the boxes. It is possible to have multiple OneBox modules applied to a frontend and their order of appearance can be determined by selecting a OneBox in the “Selected Modules” box and using the “Move Up” and “Move Down” buttons. A view of the OneBox module selection page for a frontend is shown in figure 3 below.

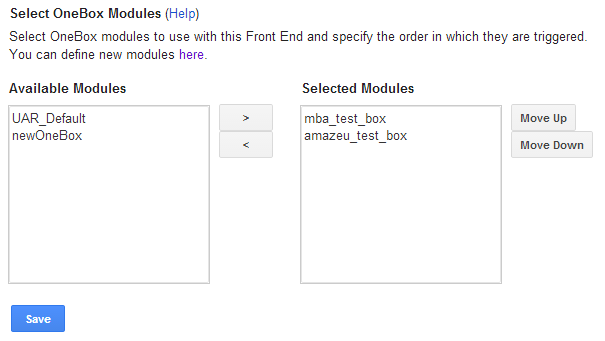


Figure 3: OneBox Module selection page for a Frontend

Within the frontend XSLT file, all of the OneBox results come up together wherever <xsl:call-template name="onebox"/> is present. Thus the location of the OneBox results can be controlled by determining where the “onebox” template is called. For GURU, it was decided that the OneBox results would appear on the right side of the main results as a sidebar element, so the template was called within the right sidebar rather than above the main results (their default location).

To learn more about OneBox Modules, go to the link below:

<http://www.google.com/support/enterprise/static/gsa/docs/admin/70/gsa_doc_set/oneboxguide/oneboxguide.html#1074314>

**Query Settings**

Query Settings, which can be accessed by going to **Search > Search Features > Query Settings** allow for expanding the scope of a search by expanding the query to include synonyms. For instance, if the user wants to see results for “Wolf” included with their search for “Dog,” the admin can make “Wolf” a synonym for “Dog” in an expanded query file. This will make results pertaining to wolves come up automatically with results for dogs when “Dog” is searched. Query Settings also allows for limiting the scope of a search by defining terms to which the variants should not be added (“blacklists”)

In GURU, there was a need to have searching “shortcuts” to make searching faster and easier. Users expressed the need to be able to use acronyms to search instead of having to type out an entire word or phrase to get to their desired results. For instance, a search for “bk” should automatically bring up results for “bankruptcy.” This was accomplished using Query Expansion within the query settings. In the query expansion files, acronyms such as “bk” were given synonyms such as “bankruptcy” in order to have results relating to both terms to come up at the same time.

Synonyms files for query expansion must be formatted properly. They may not include the following characters:

!"#$%()\*+,-/.:;<?@[\]^`{|}~

Another note to take when creating a synonym file is that query expansion is not transitive so only the original query term is expanded (synonyms are not expanded).

To define a query-synonym pair within the synonym file, the following format must be followed:

Query > Synonym

Each query-synonym pair defined is line-separated. Here is an example of a short query expansion file that includes 5 query-synonym pairs:

BK>bankruptcy

FC>foreclosure

PIW>Property Inspection Waiver

PMI>mortgage insurance

POA>power of attorney

From the sample of query-synonym pairs shown above, it is evident that synonyms do not have to be single words, they can also be phrases. The case of letters within the query and synonym is not considered during search. A .txt file with properly formatted query-synonym pairs can be uploaded via the “Add Query Expansion File” section of the Query Settings page.

To learn more about Query Settings, go to the link below:

<http://www.google.com/support/enterprise/static/gsa/docs/admin/70/admin_console_help/serve_query_expansion.html>

**Related Queries**

Related queries provide the user with alternate search options. For example, if a user searches “salary”, the gsa will suggest trying “income” instead, and provide a link to search for the new query. The difference between this and expanded queries is that related queries do not automatically include the alternate query in the search results, but instead provide an optional link to use the term in a new, separate search.

Related queries are implemented through the admin console. Each frontend has its own set of related queries, and they can be viewed and edited in the related queries tab of the specific frontend. They appear in a list with the search term on the left and the suggestion that the user will see on the right. There are two ways to edit the pairs: in the console itself or by using a .csv file. To edit directly in the console, select either “Edit Related Queries” or “Add Related Queries”, then fill out or edit the given form. A snapshot of the interface on the admin console used to edit related queries is presented in figure 4 below.

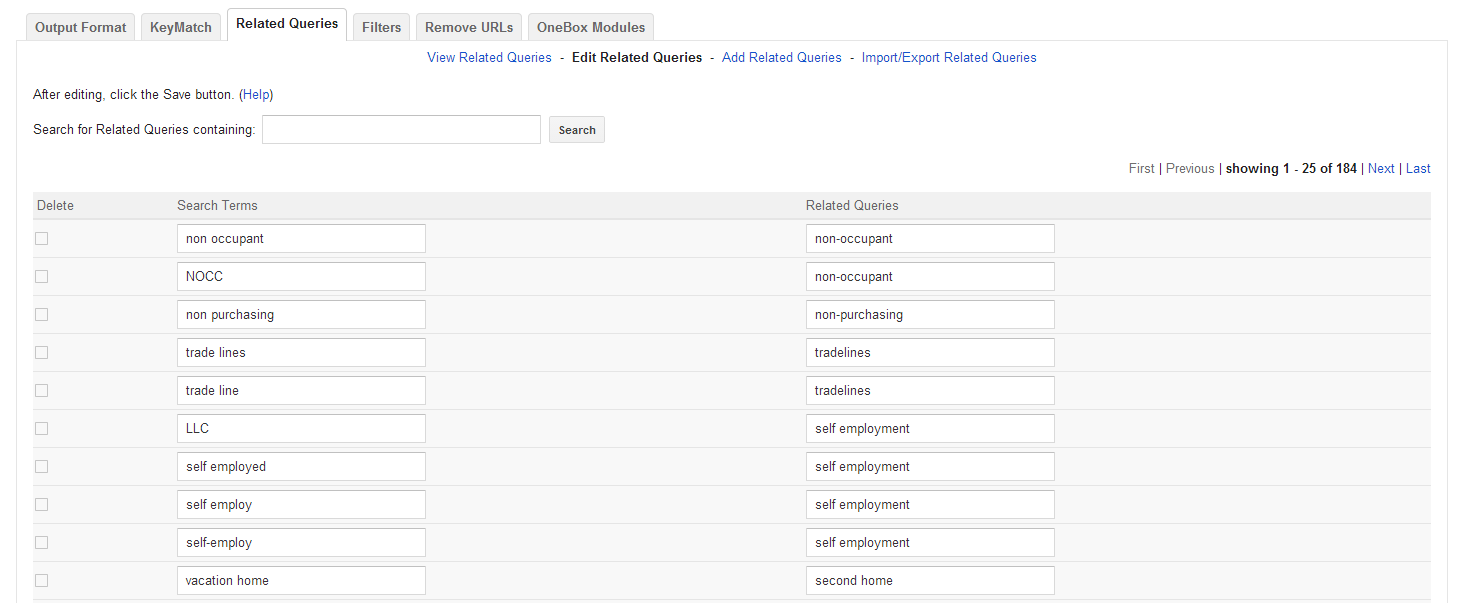


Figure 4: Admin console Related Queries editing interface

To edit externally, select “Import/Export Related Queries” then choose a location to save the file. Open it with Excel, make your edits and then make sure to save the file as a .csv file. Then return to the Import/export page and import the updated file. A snapshot of the interface on the admin console used to import/export related queries .csv files is presented in figure 5 below.

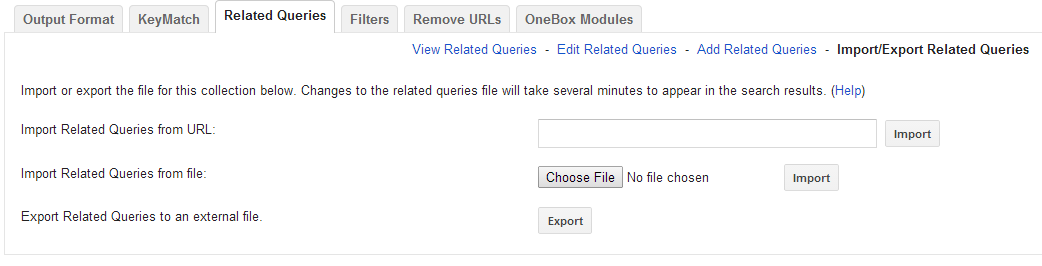


Figure 5: Related Queries file import/export interface

In Guru, related queries are used to help the user find pertinent information more easily by suggesting terms that might lead to more accurate results.

To learn more about related queries, go to the link below:

<http://www.google.com/support/enterprise/static/gsa/docs/admin/70/admin_console_help/serve_synonym.html>

**KeyMatch**

The keymatch function of the GSA allows the user to select a specific link as a result given a specific query. Essentially, it allows the administrator to select the top result for a search term.

Within the admin console, implementation of keymatch is fairly straightforward. Each individual frontend has its own set of keymatch settings, which can be viewed in the frontends section, keymatch tab. Within that tab is a list of queries, match types and urls that can be viewed, edited, added to and deleted. NOTE: order in this list does matter. If one query has two urls attached to it, the one higher on this list will appear above the lower keymatch results. The appearance of these results can be edited in the XSLT, in a section called “single keymatch result”. A screenshot of the Keymatch configuration page, which can be accessed by going to **Search > Search Features > Front Ends > KeyMatch**, is given in figure 4 below:

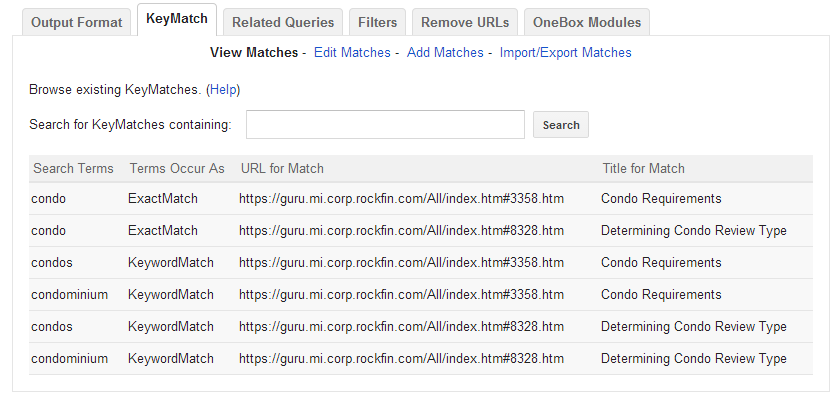


Figure 6: KeyMatch configuration page screenshot

In GURU, there are currently only two keymatch results, both of which are displayed for the three queries condo, condos, and condominium. Product Lab expressed the need to have two specific results appear first for these searches and the keymatch feature accomplished this.

To learn more about KeyMatch, go to the link below:

<http://www.google.com/support/enterprise/static/gsa/docs/admin/70/admin_console_help/serve_keymatch.html>

**Secure Search**

Secure Search is a feature of the GSA that enables the requirement of proper authentication credentials in order to access certain results. When enabled and configured with a Universal Login Authentication Mechanism, such as Kerberos, user data can also be retrieved from the Servicing Logs.

For GURU, there is a need to gather data about users in order to determine their role in the company so that a better search experience can be provided to them. However, data about how people from different parts of the company use GURU must first be gathered in order to figure out ways to present more relevant search results to different groups of users. In order to gather this data, an authentication system/mechanism needs to be established. QL systems use Kerberos for authentication, so this was integrated into GSA by configuring the Secure Search Settings.

Setting up Kerberos as a Universal Login Authentication Mechanism for the GSA involved first going into **Search > Secure Search > Universal Login Auth Mechanisms > Kerberos** and filling out the information on the page. More information on how to perform the Kerberos setup can be found by following one of the links at the end of this section. A snapshot of the Kerberos setup page is shown in figure 7 below.

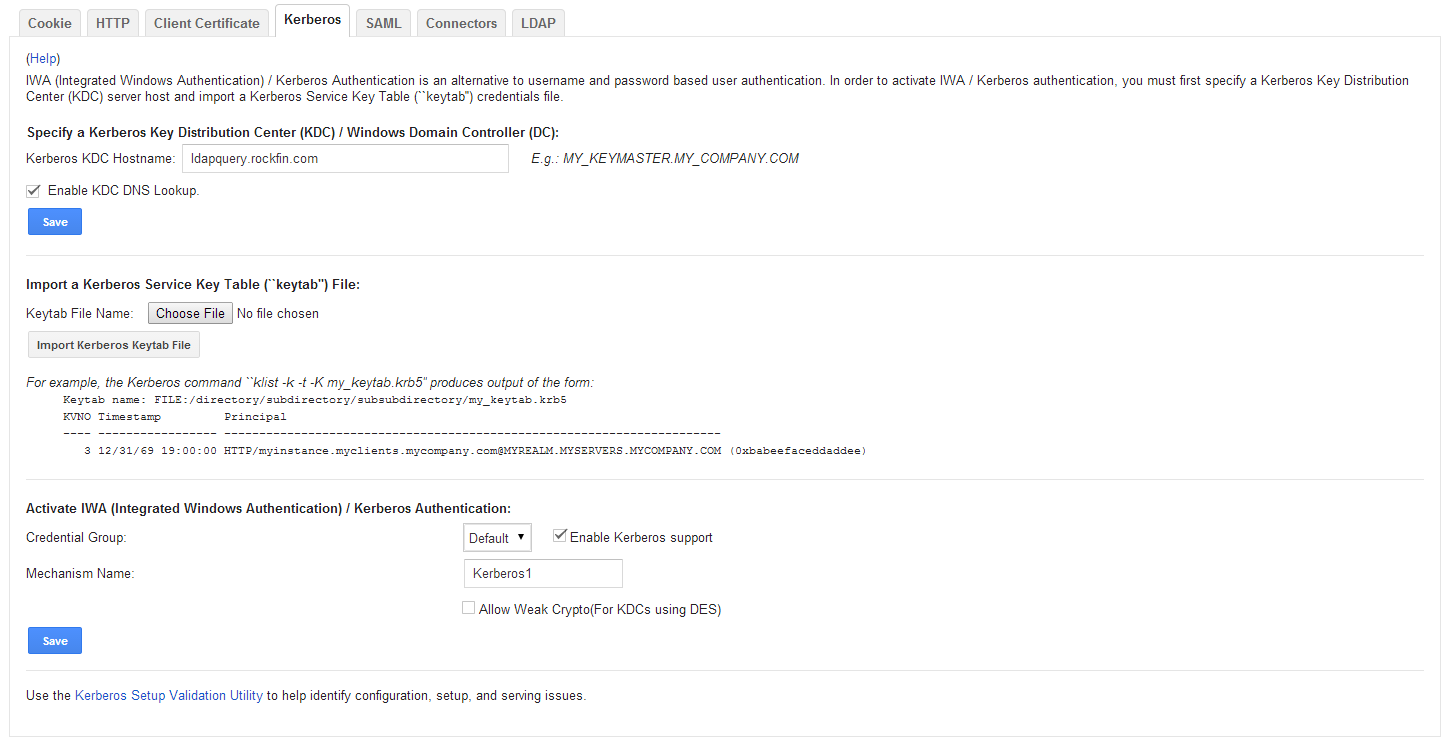


Figure 7: Kerberos configuration page

The Kerberos Setup Validation Utility can be used to help ensure proper setup and identify configuration, setup, and serving issues.

Once Kerberos-Based authentication system has properly been setup for the GSA, perimeter security can be enabled to ensure that the GSA doesn’t serve any results without user authentication. This setting can be enabled by going to **Search > Secure Search > Universal Login**, checking “Enable perimeter security,” and clicking “Save” to save the setting. This enables admin of the GSA to see user information (such as UserID), along with the query searched in the Servicing Logs, which are accessible by going to **Reports > Servicing Logs**. Search Logs can also be generated to see all the search queries for a collection within a specified time frame, along with the UserID information of the account that searched each query. Search Logs can be generated by going to **Reports > Search** Logs, and clicking “Generate Log” once a name and report timeframe have been defined. GSA can also generate reports containing statistics on searches by going to **Reports > Search Reports**, and generating a report for a specific collection within a specific timeframe.

To learn more about secure search, go to the links below:

General Security Configuration:

<https://support.google.com/gsa/answer/2766406?hl=en&ref_topic=2707841>

Kerberos-Based Authentication: <http://www.google.com/support/enterprise/static/gsa/docs/admin/72/gsa_doc_set/secure_search/secure_search_crwlsrv.html#kerberos_authentication>

**XSLT**

The frontend XSLT file is used in the GSA to set the look of the GURU results page. It was heavily modified in order to accommodate for the addition of new features and styling. Currently (as of 8/6/2014) the XSLT files for Origination/Servicing each have over 8600 lines of code and the QLMS XSLT has about 5100 lines of code. Because the frontend files are so large, they are difficult to navigate through. Thus, it is important to understand the general way in which key elements of the file are organized in order to know what portions of the code need to be modified for a given feature.

The following list outlines the more important portions of the XSLT and where they are (the line number references are from the current version of origination guru). More detail about each part of the file will come after this list.

* Variables (Lines 1 - 475)
* Header/Footer Templates (Lines 868 – 1143)
* Template “style2” CSS Style-Sheet (Line 1451 – 2651)
* Search Results (Lines 4379 – 4919)
* Search Box Input Form (Lines 5004 – 5234)
* Output Result Templates (Lines 5365 – 6387)
* Custom Sidebar Templates (Lines 6388 – 6900)
* Single Result Template (Lines 7084 – 7433)
* Single Keymatch Result Templates (Lines 7446 – 7642)

**XSLT: Variables**

The first 500 lines of code outline the global variables used in the XSLT file. These variables control many different aspects of the frontend, ranging from the color/size/styling of elements, to turning on and off features such as certain sidebar elements.

There are Global Style variables such as “global\_font” and “global\_link\_color” that control global styling and are referenced in the CSS style-sheet template later in the code.

There are variables for Result page components which can control what features are enabled in the frontend and how they may look. For instance, the “show\_top\_search\_box” variable can control whether the search box at the top of the result page is visible based on if the value of the variable is 0 or 1. In this portion of the variables section, there are also variables that have been created in order to turn new sidebar elements on and off. Examples of variables that have been added to the XSLT file are “show\_product\_up,” which determines if the product updates sidebar element is visible, and “hide\_topicnumbers” which determines if the topic numbers by the result titles are visible or hidden. Many of the variables in the XSLT file are Boolean variables so their values must be either 0 or 1 in order to work properly.

**XSLT: Header/Footer Templates**

In this area of the code, there are the global page header/footer templates, which are called “my\_page\_header,” “my\_page\_footer,” and “result\_page\_header,” along with variables for the separation bar (“sep\_bar”, separates the search bar from the body of results).

In the header/footer templates, it is possible to control the banner image used and the URL it is linked to by changing the href of the <a> tag and the src of the <img> tag. In the header for Guru, there is also a color-bar underneath the logo that can also be changed in the the “my\_page\_header” template. Pictures in the footer can be easily changed in the “my\_page\_footer” template.

**XSLT: CSS Style-Sheet Template**

This template provides the CSS styling for the XSLT. This portion of the document was modified in order to reface the look and feel of the frontend. Also, because a number of new features have been added to the frontend, it was necessary to add to this portion of the XSLT file to ensure that the new elements were styled properly. Styling for new sidebar elements, buttons, and results have been added. The CSS style sheet present in this portion of the XSLT file also includes the content of the Font Awesome CSS file so that icons from the Font Awesome library could be accessed. The majority of the newly added styling elements in the CSS are present at the top of this section of the code.

Some examples of new styles are “.dictionary-header,” “.form-container,” and “.sidebar-element.”

**XSLT: Search Results**

This portion of the XSLT code outlines templates to generate the html for the search results page. It contains JavaScript code to be generated with the html for tasks such as loading sidebar elements, checking if there are any results for a query, and posting dictionary results. Code is added here when there is a need for JavaScript to make an element of the frontend operate correctly.

For instance, the “readJson” function that makes the dictionary results work and the “readProductupJson” function that post product updates on a sidebar element are both functions that were written into this section of the XSLT.

**XSLT: Search Box Input Form**

This section of the XSLT file controls how the search form looks and functions. This was modified in order to update the styling of the search box and button Although for the majority of the document, tables were replaced with divs, the table for the search bar was kept in order to ensure that the elements of the search bar (such as the text box and the button) stayed together and maintained their formatting and placement on the page. Among the changes made to the search box input form were the styling of the text box and the styling and content of the button (“Search Guru” was replaced with a font awesome magnifying glass icon).

**XSLT: Output Result Templates**

In this section of the code, there are many templates that help with outputting results. The first template, which is called “results,” organizes the body of results, which is everything between the header and footer. This template calls the “main\_results” template, which was the most modified part of this section of the XSLT (Lines 5968 – 6387). The main results controls where the results appear within the page, where all the sidebars are located, where the navigation elements of the results are, and the presence of the bottom search box.

Many modifications were made to the main\_results template. Initially, there was no sidebar on the left side of the results page. The sidebar on the right side of the page was also not located in a location that would catch people’s attention as they performed searches, since it was small and on the rightmost side of the page. The template was modified so that, if new left sidebar elements were enabled, the page would automatically format to accommodate for the space need for the sidebar elements. The elements in the right sidebar were also reformatted and styled in order to bring them closer to the results so that they would be noticed. Thus, the main\_results template is very flexible and responsive to variables that enable and disable elements of the page (such as the sidebars, dictionary results, and keymatch results). The locations of some of the built in tools of the GSA, such as OneBox results and cluster results, were changed so that they could be present in sidebars.

**XSLT: Custom Sidebar Templates**

This section contains the templates for each custom sidebar we have created. Each of these is called within the main result generation template if its respective variable is set to 1. Most of them are governed by the “sidebar-element” class, meaning that to change the look of all sidebars you only need to edit one section of the css.

* Product Updates
  + This is the container into which the product updates information is loaded. For more information, see the section on product updates.
  + Variable: “show\_productup”
* Helpful Links
  + This is where we display the links that product lab wants users to see regardless of the search query
  + Not loaded from an external source, to change the links the xslt must be changes
  + Refer to the section on Helpful Links on more information about how to change/add links to this sidebar element.
  + Variable: “show\_helplinks”
* Dictionary Results
  + Here is the location into which to load the dictionary matches for the specific query
  + Does not appear unless there is a result to display
  + Variable: “show\_dictionary\_results”
* Relevant Training Materials
  + This is the container for our MBA/AmazeU OneBox modules to be loaded into
  + For more information, see the section on OneBox modules
  + Variable: “show\_mbaamaze”

**XSLT: Single Result Template**

This is the section in which the behavior of a single search result is determined. The first few chunks of code handle the URL of the result, taking various pieces of it, for example extracting the URL to display in the result. Then next part handles the result header. This header is blank unless the documents come from a specific source and we have not edited it.

The next chunk, “Result Title”, is the section that actually generates the result. Firstly, it handles the URL. The part of this that we edited is the section about adding the query to pass through and is based on the variable “pass\_query\_through\_url”. When that variable is set to 1, the GSA adds a query parameter into the URL, allowing the highlighting in Author-It to use the search term. For more information on highlighting, see the section on Highlighting Search Terms. Next is the Snippet Box, which holds the short summary of the link that the GSA generates. In that same area, a template called “reformat\_keyword” is called. This is a function that we added that removes the collection numbers from the front of the title of each result. The next few are display options essentially, Meta tags, URL, and miscellaneous all can be turned on and off with their respective variables and here is where the GSA displays or does not display them.

As far as the style of the result, there are a few classes that are important. “result-item” is the class of the div that each result is generated into. The class “r” is the title of the page, which is also a link. However, “r” is replaced by “l” when the title is reformatted with “reformat keyword”. The snippet (the brief preview of the content) has the class “s”.

**XSLT: Single Keymatch Result Template**

This template handles the appearance of the keymatch results. Because of our implementation of keymatch (where it appears as another result, the only indication of any difference is the “suggested result” text in the upper right corner) instead of creating a separate css class for keymatch we just used the “result-item” class. The colors and upper right corner text can be edited by changing the variables keymatch\_bg\_color, keymatch\_text\_color and keymatch\_text. We also changed the formatting from a table to a div and spans so that we could more easily format the content within. Throughout the entire document we have made an effort to “flip the tables” by changing as many tables as possible to div and span tags instead.

Below in figure 8 is a fun example of flipping a table:

(\°o°)\ **┬─┬** 🡪 (╯°□°)╯︵ ┻━┻

Figure 8: Flipping Tables

**XSLT: Procedure for Modifying and Pushing**

Because of the differences in the links, collections, and frontend names between the test environment for GURU and the production environment of GURU, two different XSLT files were maintained for each frontend in order to keep track of what could go in the test environment and what should go in production. Whenever something in test was ready to go into production, the following procedure was used to update the production file:

1. Compare the current version of the production frontend to the new test frontend, and make the requisite changes.
2. Make sure that the key differences in link references, collection references, etc. between the test environment and the production environment are maintained.

Currently, the only differences between the test environment and the production environment are that certain features have been disabled because they are not fully developed, the href for the <a> tag in the “my\_page\_header” are different (in test, the banner image refer to gurutest and in prod, the banner image refers to guru), and the collection filter links templates are different (different frontend\_name,

Thus, in order to properly transition from test into production, the links in the “my\_page\_header” and “my\_page\_footer” templates need to be changed from “gurutest” links to “guru” links. The collection\_links template also needs to be modified when transitioning from test to production. The way the XSLT is currently written, this is a matter of uncommenting the version of the   
template with references to <http://gsa/search> and commenting the version of the template with references to <http://gsa/search>. The differences between the two versions are the collections and frontends that the filter links are connected to.

**Author-It: Editing Headers/Footers**

GURU’s content is housed in a content management site called Author-It. In order to match the look of the content pages with the results page, it is possible to change the header and footer of the content pages as well. This can be done using HTML and CSS. The two files necessary to make changes are called index.htm and webhelp.css and they can be found in each specific collection’s folder within [this](file:///I:\Capital%20Markets\Author-it%205\Data\Templates) folder, called templates. Currently, there is a different pair of index.htm and webhelp.css for each type of GURU (servicing, origination, QLMS portal) but within each of the three, the file is the same for every collection (with the exception of Legal in servicing). Images can be placed in either the main GURU images folder or in the individual collection folders.

There are two folders crucial to updating the Author-It templates. One is the above templates folder and the other is the [projects](file:///i:\projects\guru) folder. In order to permanently update the content pages, both places must be updated. To update the pages initially, the new index.htm and webhelp.css need to be copied into the appropriate projects folders. This, however, does not complete the process because once content is published or updated by Product Lab, the index.htm and webhelp.css are pulled from the appropriate templates folder. Therefore, to avoid reversion of the content pages upon publication, the templates must be updated as well.

**SharePoint Integration**

One of the features that we worked on implementing was integrating training materials into the GSA searches. The content that we wanted to access is housed in two separate SharePoint websites, the Mortgage Banker Academy and AmazeU. We created two OneBox modules, one for each site’s content. They can be turned on in the xslt code by setting the “show\_mbaamze” variable to 1 instead of 0. For information on editing the look of the training sidebar, see the section on OneBoxes.

Although the sidebar is in place, we currently do not have content to populate it with. In order to access the information, we need to be able to crawl SharePoint content. The GSA is capable of doing so, but the SharePoint team wants to make sure that the crawl won’t affect their production environment. Sam Kaufman has been working with Kristin Cameron from the SharePoint team in a test environment.

As far as implementing the feature once we have access to SharePoint’s content, it will be as simple as turning on the “show\_mbaamaze” variable in the xslt and, in the admin console, pointing the OneBoxes to the newly created collections of training materials. They are already configured to search a collection using the user’s query and display the results on the right side of the main results. In the image below in figure 9, the sidebar in question is circled in red.

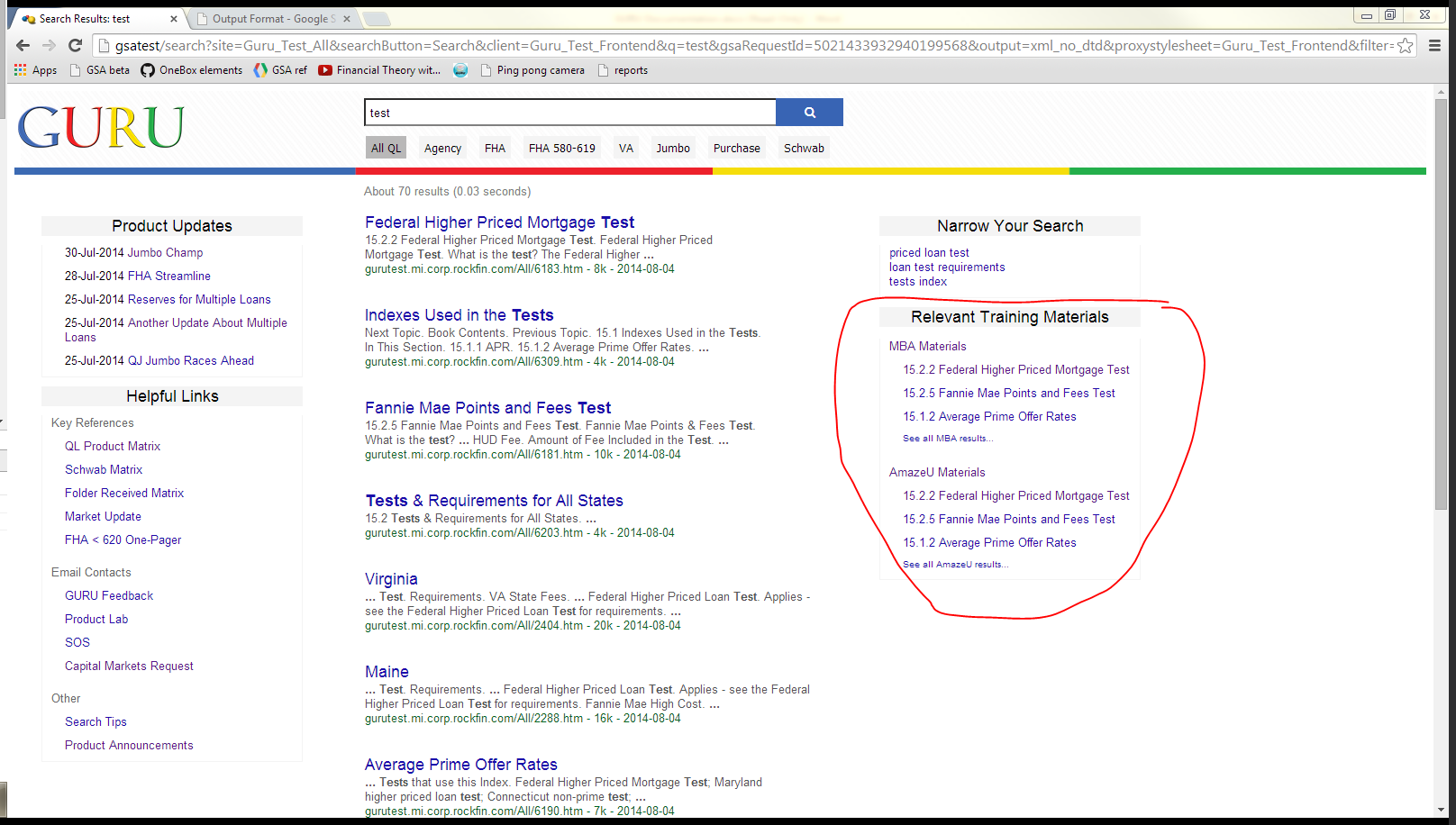


Figure 9: Framework built for MBA/AmazeU Training Materials Sidebar.

Currently, product updates are configured so that the 5 most recent updates are posted on the sidebar. However, in the future, the goal is to have the option to search through product updates to find those that are most relevant. In order to make this happen, it seems that the sharepoint documents that comprise the product updates need to be crawled and a OneBox module needs to be set up in order to have the product updates visible in a sidebar element. s

**Single Sign-on**

In order to collect more accurate search data from the GSA we wanted to be able to find out not only what is being searched but also by whom. The solution to this is enabling single sign-on user authentication in the GSA admin console. When implemented correctly, this allows the GSA to connect to Quicken’s authentication program and it will include usernames in the search logs and serving logs beside the queries.

Although this feature is currently turned on and functioning in the guru test environment, there are issues with the setup of Kerberos authentication in production. Thus, it is necessary to look at the GSA documentation online for Kerberos set up for the GSA in order to ensure that it is configured properly. In the paragraph below, the general process for setting up Kerberos will be outlined.

Single sign-on can be enabled in the admin console by following the path **Search > Secure Search > Universal Login Auth Mechanisms > Kerberos.** Quicken uses a Kerberos authentication system. Once the authentication is turned on, in order to see the usernames you must go to **Search > Secure Search > Access Control** and check the “Record user identity in Search Logs and Serving Logs” box. Now, the GSA will include username data if available. Lastly, to make the data available, go to **Search > Secure Search > Universal Login** and check the “Enable perimeter security” box. This tells the GSA to get the login information every time a search is made, not just when private content is involved.

For more information on how to integrate Kerberos with GSA, refer to the resources in this document for the Secure Search section or contact Sam Kaufman.

**Author-It Javascript: Highlighting Search Terms**

The file responsible for inserting the content into each Author-It topic is called webhelp.js. This file, like the index.htm and webhelp.css files, is kept in the [templates](file:///I:\Capital%20Markets\Author-it%205\Data\Templates) folder. Each collection (e.g. FHA, Jumbo) has its own javascript file and, unlike the html and css files, each is unique. Therefore, it is especially important to back up each file before editing.

The only change we made to the js is in test currently, not yet pushed to prod. We implemented a highlighting capability so that the content page can accept an argument passed through by gsa in the url. There are four functions involved in this process.

1. fixurl()
   1. This function facilitates the existence of an argument in the url. It takes in the address of the page and then, if an argument exists, sets the “url\_noparam” variable to the url without the argument, allowing the other sections of js to function normally.
   2. Additionally, this function extracts the argument and stores it in the “quer” variable for use by the other highlighting functions.
2. splitContent(string)
   1. splitContent() takes in html code as a string and separates it into an array
   2. Each entry is a string, either an html tag (“<something>”) or the text between tags (“some other thing”). This allows the highlight to avoid inserting tags within tags.
3. highlightArray(string[])
   1. This one takes in the array created by splitContent() and then considers each entry, then outputs html code as a string.
   2. If the entry contains the “<” symbol, it is added to the string without any modification. Additionally, if the tag is either “script” or “style” (both of which have text that is not tags but is also not to be highlighted), it and the next two entries (assumed to be the actual script or style and the end tag) are added to the string without modification.
   3. If the entry is just text, it is sent to the highlighter() function
4. highlighter(string)
   1. highlighter() takes in a string and returns a string with added html span tags around the query
   2. all occurances of the “quer” variable are surrounded with <span class=”highlighted”>quer</span> tags. This class can be edited in the corresponding webhelp.css file.

In order for the content to load properly, fixurl() must be called before the js accesses the url of the page and each instance of window.location.href or document.location.href must be replaced with url\_noparam. This way, the other functions will always receive a “clean” url. The hightlighting function must be placed before the content is loaded into the page. This occurs in this snippet:

“Ext.get(‘start-panel’).update(response.responseText, true);”

In order for the highlighted text to appear, response.responseText must be replaced by the output of the highlightArray() function. This can be achieved by replacing the line with:

“Ext.get(‘start-panel’).update(highlightArray(splitContent(response.responseText)), true);”

In order to pass the query from the GURU results page to the Author-It page, a query needs to be passed through the URL. In order to make sure that clicking on a result link passes the query through the URL, this capability must be enabled in the frontend XSLT file by setting the value of the variable “pass\_query\_through\_url” to 1. This is currently disabled due to issues with the webhelp.js not being completely ready.

**Product Updates**

The product updates sidebar requires several steps to work properly. Currently, it is turned off since the correct framework has not yet been implemented on the server side in order to accommodate the JSON file which needs to be in place in order for the product updates sidebar to get its information. Thus, the following steps must be taken in order to get the product updates sidebar fully functional, since the framework is already there:

1. Export the Product & Guideline Announcements from the <http://quniverse/teams/capitalmarkets/Pages/Product%20Announcements.aspx> page to Excel (.xlsx). See figure 10 to see how to export to excel from the announcement site.

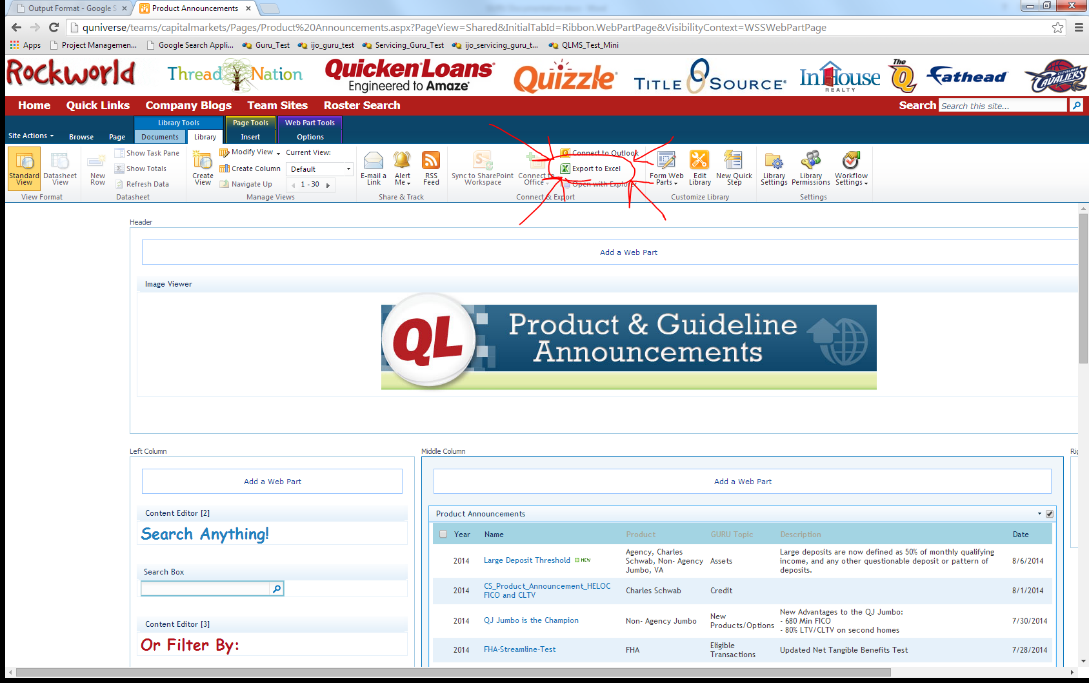


Figure 10: Location of the “Export to Excel” Button

1. Use the “ExcelToJson.jar” application to convert the Excel file to a JSON file. Below in figure 11 is an image of the application interface

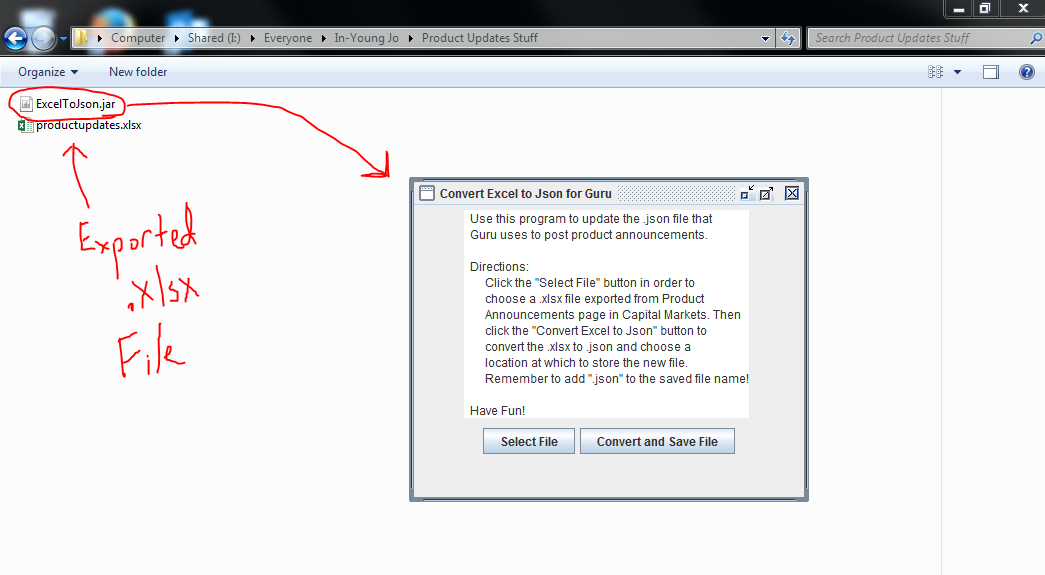


Figure 11: View of the ExcelToJson.jar interface

* 1. Click “Select File” to select the excel document exported from Sharepoint. Look at Figure 12 below to see how to use the file selector that comes up after pressing the “Select File” button:

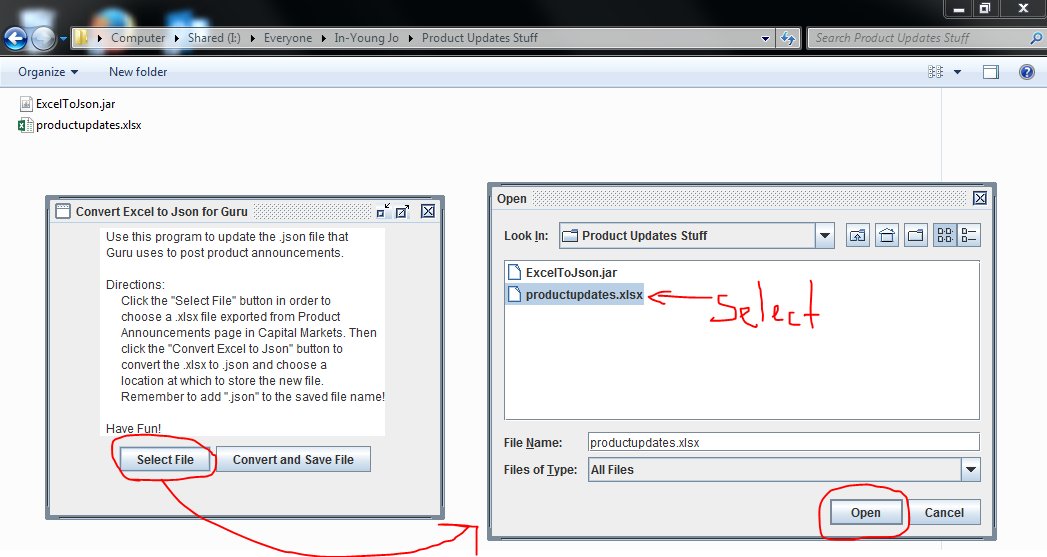


Figure 12: File chooser for ExcelToJson.jar

After choosing the exported .xlsx file, the selected file will appear on the original application window, as seen in figure 13 below:

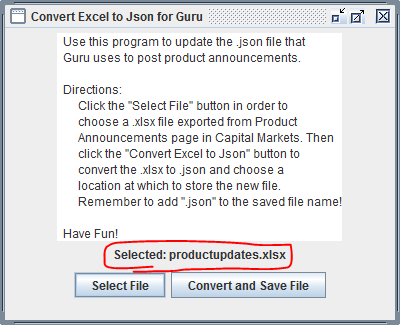


Figure 13: Updated window

* 1. Click “Covert and Save File” to select a location for the output JSON file. **Make sure to name the file “productupdates.json”** Look at figure 14 below to see how to use the file saving window that comes up after pressing the “Convert and Save File” button:

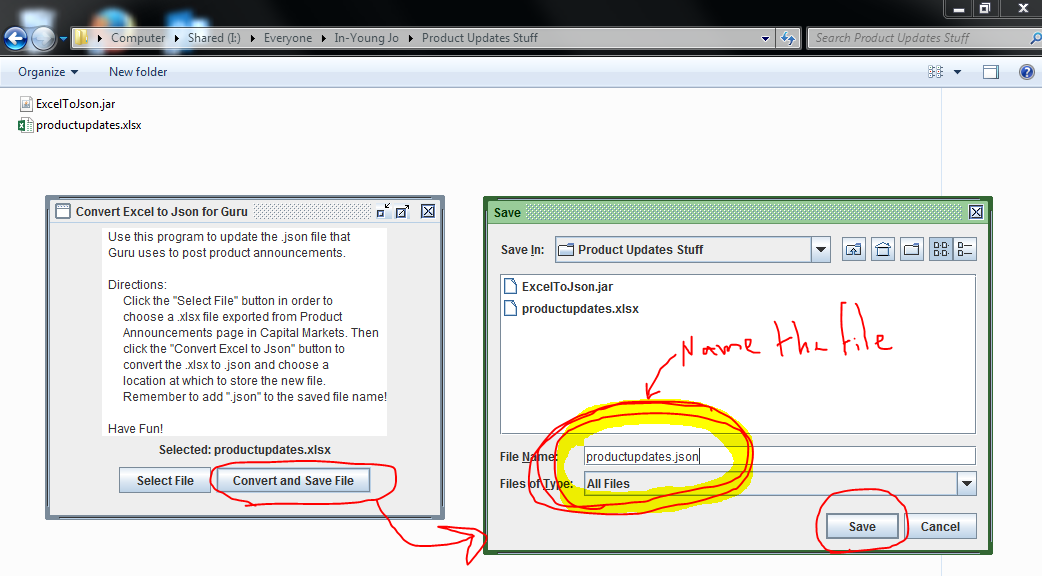


Figure 14: File saving for ExcelToJson.jar

After clicking the “Save” button on the file saving window, the productupdates.json file should appear in the specified directory, as shown in Figure 15 below:

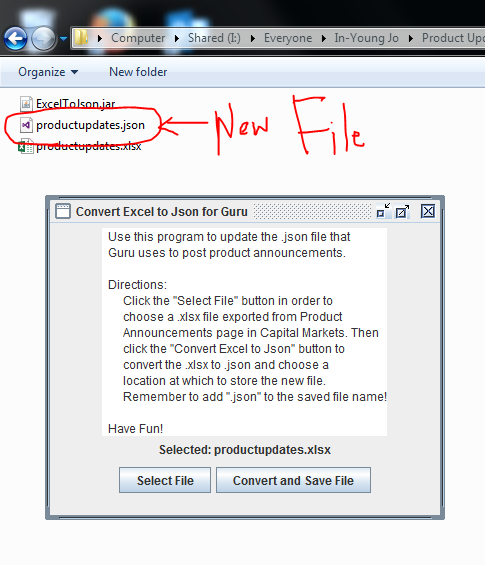


Figure 15: Directory containing output productupdates.json file

1. Fill in the “info” tags for each of the first 5 updates within the json file that will be visible within the product updates sidebar. Fill in the empty quotes by the “info” tags with the phrase that should appear for the link.

Note: Since Charles Schwab product updates are not to be posted, there is no need to fill in the info for them, as they are automatically excluded from the list of product updates by the javascript.

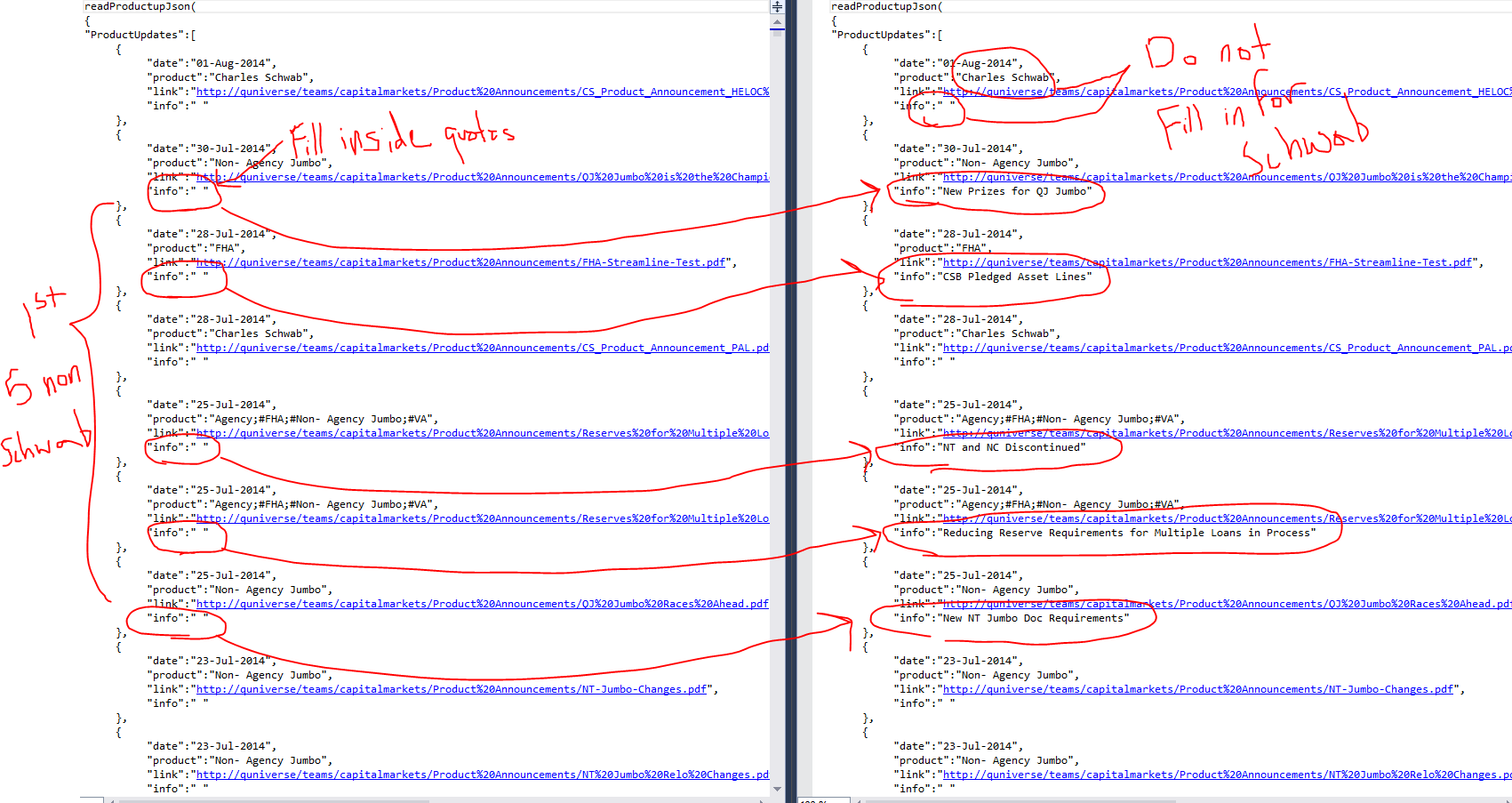


Figure 16: Fill in the space between quotes by the “info” tags for non-schwab updates

1. Make sure that the productupdates.json file is moved onto the server.
2. Make sure that within the frontend XSLT file that the variable “show\_produtup” is set to 1 and that within the JavaScript section of the XSLT, “productupScript.src = 'LINK';” where link is the URL of the productupdates.json file. See figures 17 and 18 below to see what in the XSLT file needs to be changed in order to make the product updates sidebar in GURU to work:

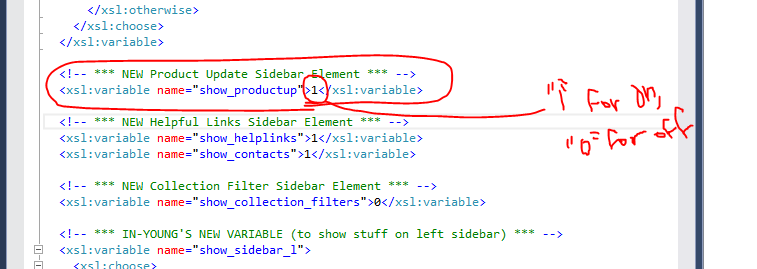


Figure 17: Set “show\_productup” to 1 to enable sidebar

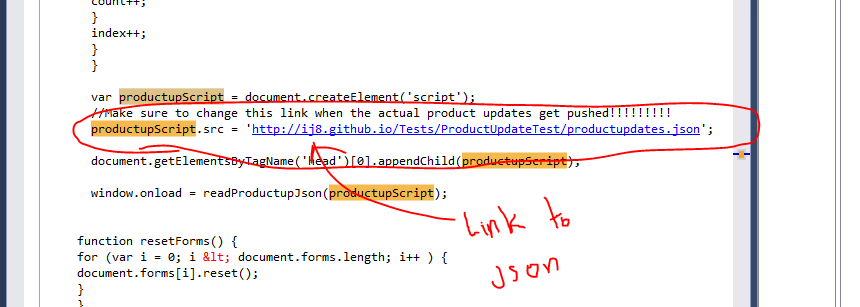


Figure 18: Make sure the link is correct

If you decide to turn on the product updates (they are currently off in production), you only really need to perform this last step once.

If the steps outlined are followed, the product updates sidebar should come up in the frontend with updated information. The first four steps of this process must be performed every time a new announcement is made in order for the product update sidebar to show the most recent information.

It’s really not hard to do this! Once you go through turning on the product updates sidebar once and establish a location on the server to put the productupdates.json file, it can easily become a routine of updating the product announcements page.

In the future, the “readProductupJson” JavaScript function should be applied to the Capital Markets Page (<http://quniverse/teams/capitalmarkets/Pages/default.aspx>) so that the Product Updates bar in this page, which is shown below, can be updated at the same time as the GURU Product Updates sidebar element.



Figure 19: The “readProductupJson” could be applied to the Product Updates and Schwab Updates elements so that they update simultaneously with the updates bar in GURU when the productupdates.json file is updated.

**Helpful Links**

The way things are currently set up, one must edit the “helplinks\_sidebar” template in order to change links in the sidebar (editing/adding/removing them). In order to see how the helplinks\_sidebar template is organized, refer to figure 20 below.



Figure 20: “helplinks\_sidebar” template organization

When adding a link to the sidebar, follow the steps below:

1. Find which category the link falls under
   1. There are 3 categories, denoted by the <div> tags of the class “sb-l-lbl” surrounding them. The categories are “Key References,” “Email Contacts,” and “Other.”
2. Under the proper category, create a <div> of class “sb-l-res” and add an <a> tag that is formatted in the same way as all the other <a> tags in this template
   1. In essence, just copy one of the other divs of the sb-l-res class and paste it under the right category. Change the link and label for the link in the <a> tag to the desired link and label.

**Dictionary Results**

Dictionary results are a new feature of the GURU that posts the definitions to certain search queries if they are defined within a dictionary JSON file stored on the server. The display of dictionary results can be turned on and off by setting the “show\_dictionary\_results” variable in the XSLT to 1 or 0.

The dictionary JSON file, which is named “GuruDictionary.json,” is stored at the following URL:

<https://guru.mi.corp.rockfin.com/dictionary/GuruDictionary.json>

Here is a very short example of the GuruDictionary.json file with just 2 terms in it:

readJson(

{

"GuruDictionary": [

{

"term": "Bankruptcy",

"definition": "Bankruptcy (BK) is defined as the client's application for court ordered release or restructuring of debt."

},

{

"term": "Pre-Paid Tuition Plan",

"definition": "Plan where participants lock in future tuition fees at today's prices."

}

]

}

)

In order to add a term/definition, copy and paste the formatting from a previous term/definition and replace the contents within the quotation marks by the “term” and “definition” tags with the desired contents. Remember to include the open and closed curly brackets ( “{“ and “}” ) that surround the term/definition pairing. Here is an example of the GuruDictinary.json file after a term has been added:

readJson(

{

"GuruDictionary": [

{

"term": "Bankruptcy",

"definition": "Bankruptcy (BK) is defined as the client's application for court ordered release or restructuring of debt."

},

{

"term": "Pre-Paid Tuition Plan",

"definition": "Plan where participants lock in future tuition fees at today's prices."

},

{

"term": "New Term",

"definition": "Definition of the New Term"

}

]

}

)

Note that when the new term was added to the end of the document a comma was added after the closing bracket of the previous term, while the last term does not have a comma. This is very important to maintaining the validity of the JSON file. In order to check if the contents of the JSON file are valid, copy everything BETWEEN the open parenthesis on the first line and closed parenthesis on the last line (make sure not to include the first line and last line) and paste the text into a JSON validator, which can be found online. One example of a JSON validator is <http://jsonlint.com/> and a screenshot of its usage is shown in figure 21 below.



Figure 21: Using a JSON validator to check the GuruDictionary.json file

When the desired terms have been edited/added to the JSON file, reupload the GuruDictionary.json file to the server (make sure it is named that!) and the new/updated dictionary result should come up instantaneously when searched on GURU.

There is JavaScript code in the frontend XSLT file to make dictionary results work. The function is called “readJson” and it makes a dictionary box containing a term and its definition visible if it matches the query that is searched.

Right now, the dictionary result posting functionality in GURU works and more terms/definitions just need to be added to the GuruDictionary.json file (there are currently only 10 terms in the document).

After learning about the ability to export sharepoint lists to excel, a process similar to that used to convert the product updates excel files to JSON can be used in conjunction with the process of posting normal dictionary results in order to provide results stemming from Quicktionary. In order to learn more about the ExcelToJson conversion process that is employed for product updates, refer to the “Product Updates” section within the “Loose Ends” of this document.