**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 Github repository at the following link:

<https://github.com/ij8/Guru>

**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
* **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.

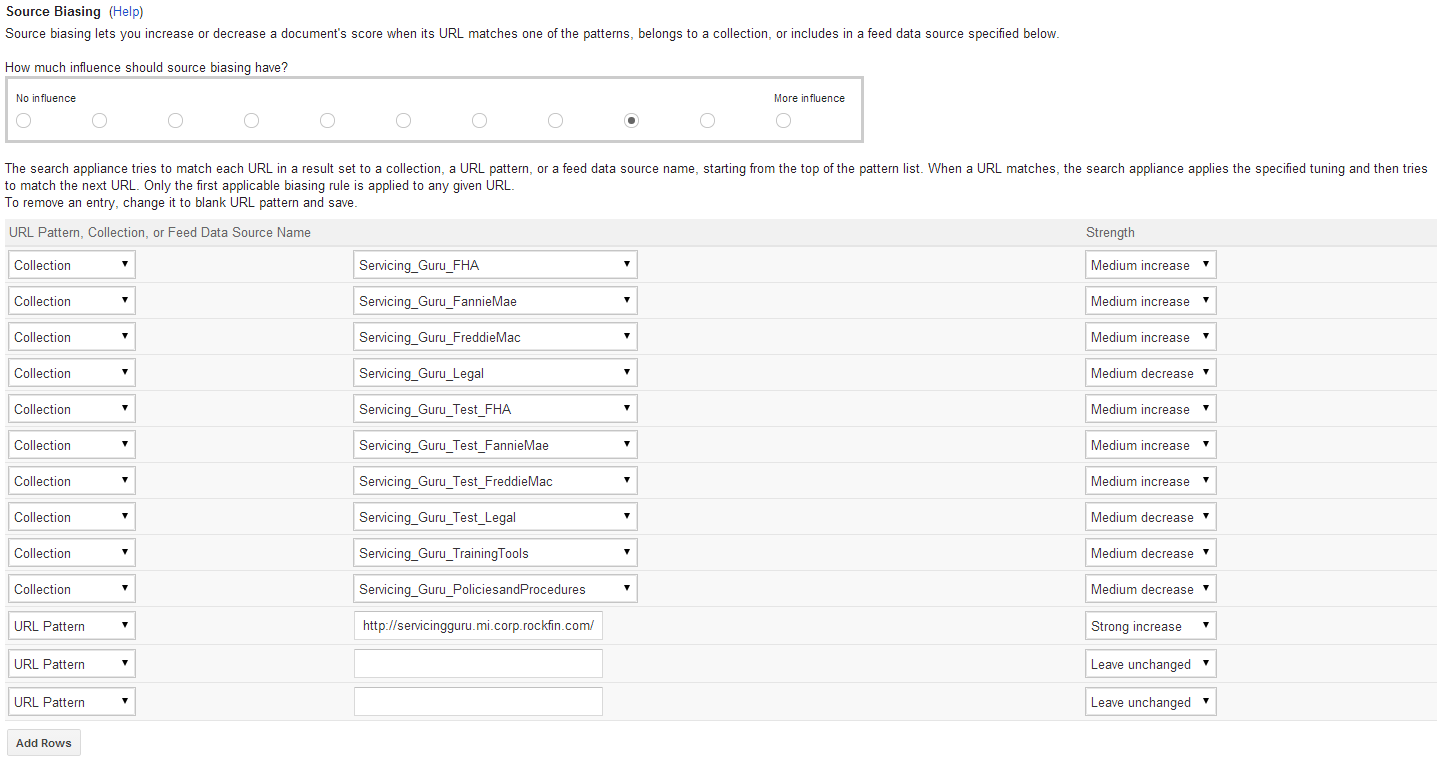


Figure 1: Source biasing interface

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.

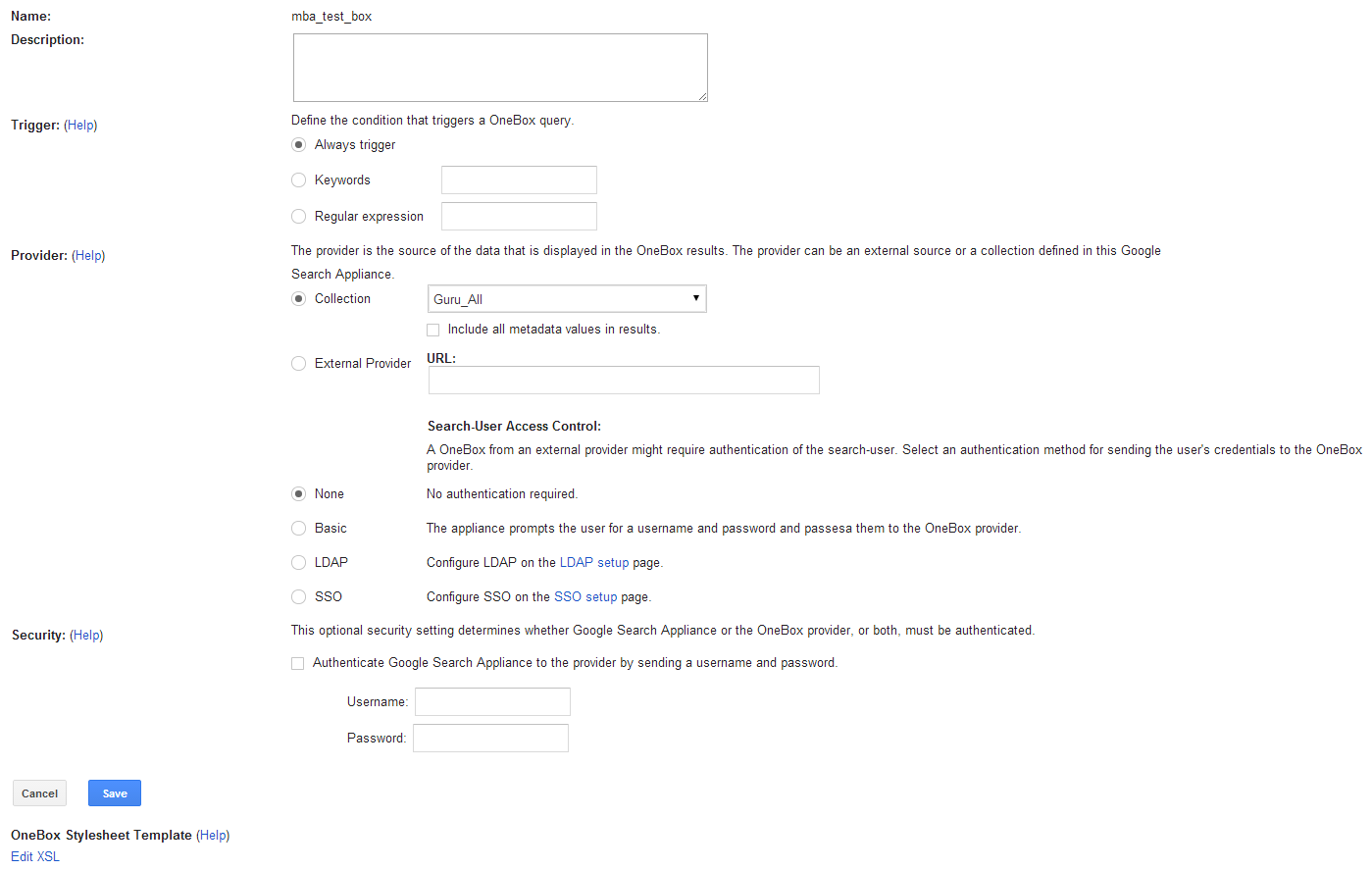


Figure 2: OneBox Module configuration page

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.

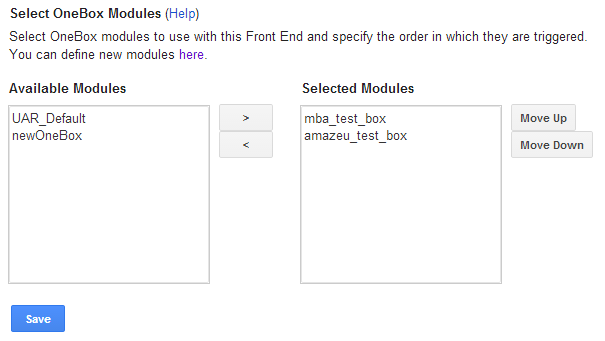


Figure 3: OneBox Module selection page for a Frontend

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.

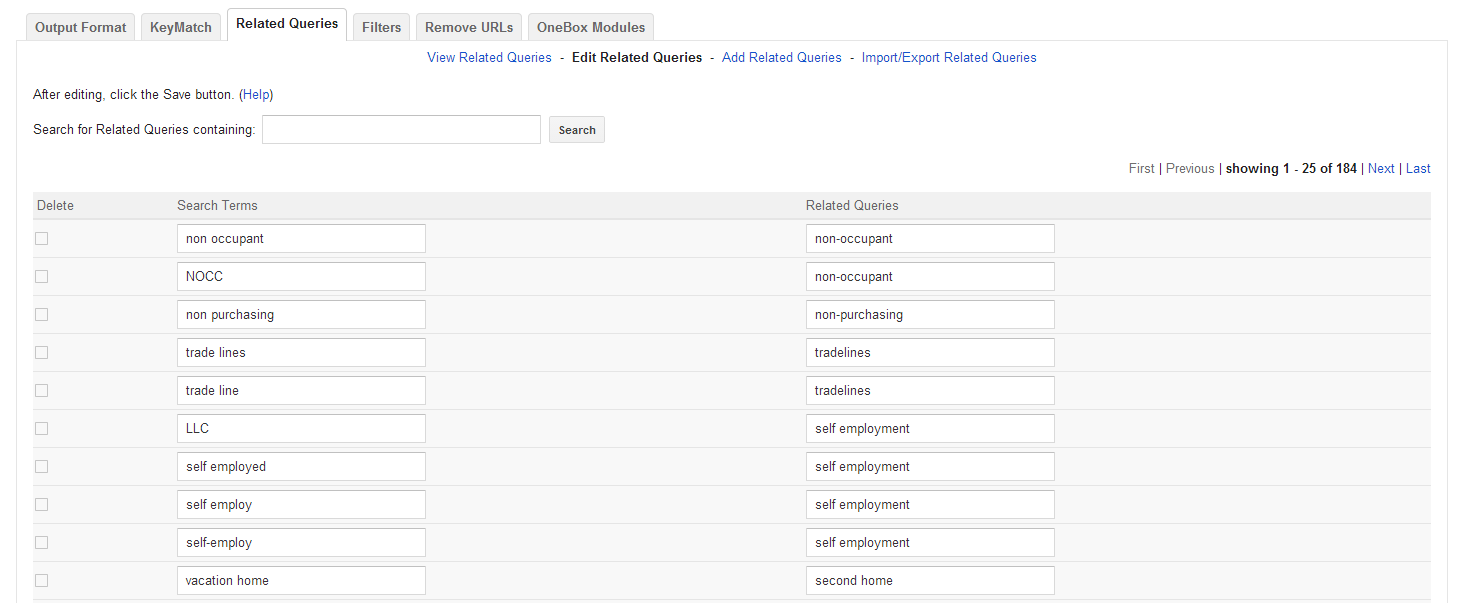


Figure 4: Admin console Related Queries editing interface

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.

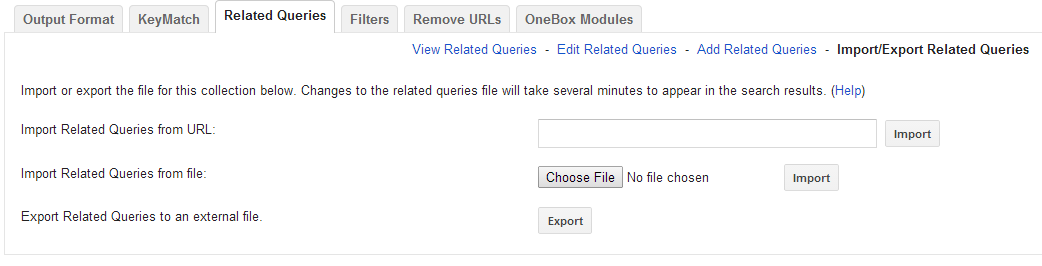


Figure 5: Related Queries file import/export interface

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:

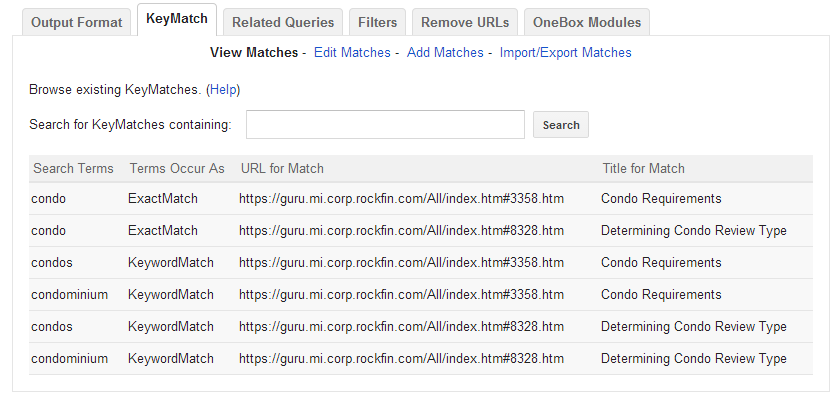


Figure 6: KeyMatch configuration page screenshot

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.

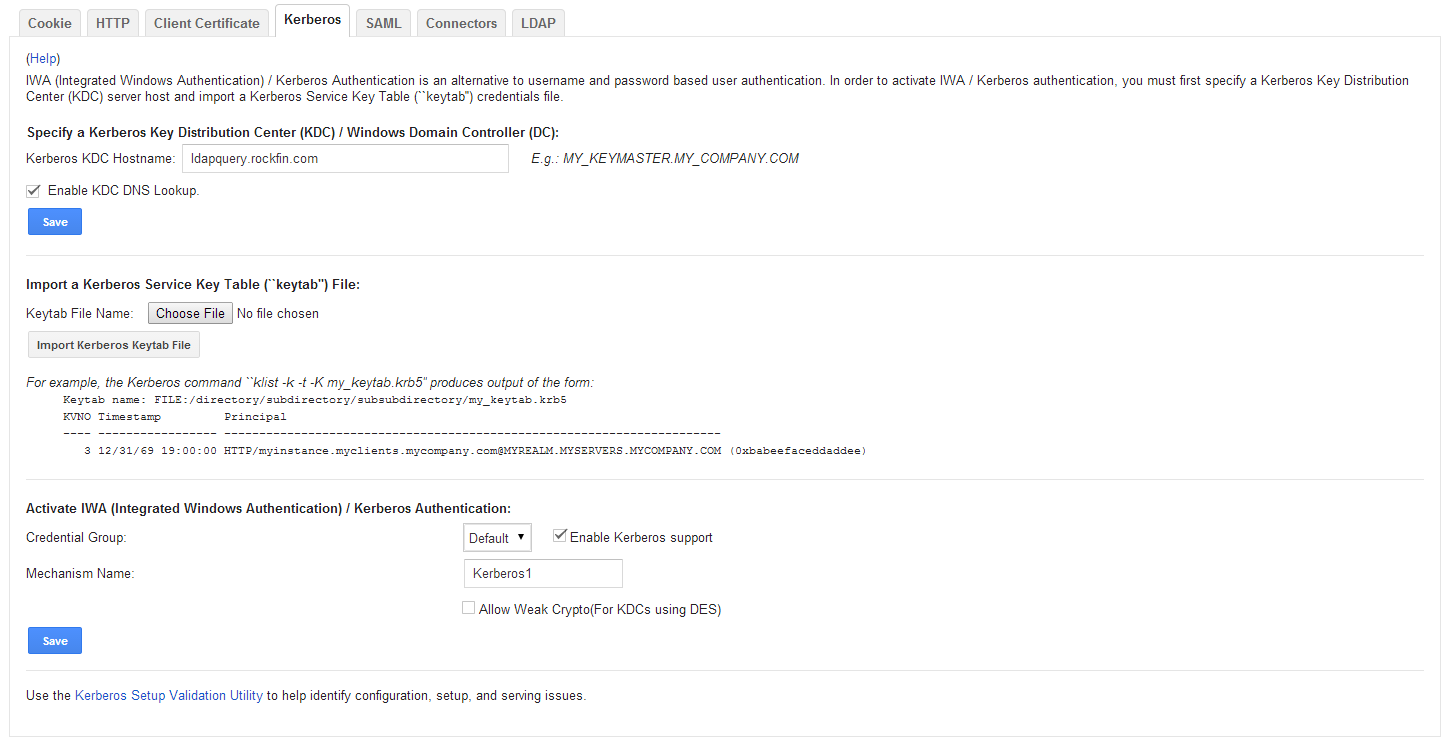


Figure 7: Kerberos configuration page

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 XSLT files are currently organized as follows (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)
  + 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.
* User Added Results Templates (Lines 477 – 866)
  + Not 100% sure about how these function, but they were not tampered with. Judging from the comments before each of the templates, they have to do with UAR or User Added Results.
* Header/Footer Templates (Lines 868 – 1143)
  + Only modified the global page header/footer templates which are called “my\_page\_header,” “my\_page\_footer,” and “result\_page\_header”
  + Variables for the separation bar (sep\_bar, separates the search bar from the body of results) are also declared in this area of the code.
* Suggest Service/Search Within Results input page/”Front door” search input page (Lines 1145 - 1397)
  + Did not modify
* Empty Result Set (Lines 1400 – 1448)
  + Called later in the code to display a message for the case when there are no results for a query. Did not modify.
* Template “style2” CSS Style-Sheet (Line 1451 – 2651)
  + This template provides the CSS styling for the XSLT.
* URL Variables/Search Parameters/Page Identifier/Cached Page/Advanced Search Page/Google Apps search results/ (Lines 2652 - 4377)
  + Did not modify much, if at all.
* Search Results

**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.

**Author-It Javascript**

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);”