Michigan State University
LIBRARIES

LexisNexis WSK Implementation
Technical Overview

Purpose:

The purpose of this application is to consume the web service kit (WSK) provided by Lexis Nexis to provide MSU users with the ability to perform searches for research or data mining.

This initial release's goal is to implement a basic search incorporating the throttling controls associated with the limited license that MSU Libraries has with Lexis Nexis.

Components:

C# ASP.NET Web Site:
Allows users to queue search requests and download completed results.

C# Console Application:
Processes the search queue that is stored in the backend database.

MariaDB Backend Database:
Stores search status and lookup information such as the throttling limitations.

Overview:

The application has 3 main components: a web front end built in C# ASP.NET, a MariaDB backend database, and a C# console application.

The web front end allows users to enter search criteria and preview the first 10 results that the web service returns, queue searches that they want to retrieve the full result set for, and download completed search request results. If a search is too general, it will split the range in half and try to pull the first 10 results from the right half of the range, continuing to split that way until it gets down to a 1 day range before it returns that the search was too general. The screen will specify what range the result preview is from.

The database stores user's search requests with the current status, throttling limitations (such as the number of searches we can perform an hour, the hours we are allowed to run during, and the number of documents we can retrieve per search), as well as other application level values (like how many searches have been performed in the current hour or if the application is currently running).

The console application's primary function is to processes the search request queue. The processing steps are as follows:

Step 1: Determine if the process should be run (a connection with the application's database can be established, is within the valid run window, has remaining searches for the hour, and there are search requests in the queue).

Step 2: Loop through each of the search requests in the queue and call the search processor function for it.

Step 3: The search processor will attempt the call the web service for the search request for the entire date range specified, and if successful it will loop through the results based on the number of documents we can retrieve per search and download the results. It will save each document in html and txt format to the server. It will update the database after each iteration with the current index it is at. After each iteration it will also check that there are still remaining searches left for the current hour before continuing, otherwise it will stop.

Step 4: If the web service call for the entire date range fails, it will break the date range in half and try doing the left half of the range and then the right half. The function is written recursively so it will continue to split the date range until it gets to a range of 1 day before marking it as an invalid search for being too general. Once it finds a valid date range where the search is not too general, it will follow the same process described in step 3 with iterating through the results based on the current index (but will also store the current start and end date that it is working on for that index so it can pick up where it left off if it runs out of searches for the current hour).

Note: The reason we store the current index, start and end date is so the processor can pick up a search from the middle if it runs out of searches for that current hour and has to stop. For example if we have a search from 10/1/2015 – 10/20/2015 that was too general and it is currently on the 211th document in the range 10/17/2015 – 10/20/2015, it won't have to restart the processing from the beginning because it knows we have already downloaded the documents from 10/1/2015 – 10/16/2015 and the first 210 documents in the range of 10/17/2015 – 10/20/2015.

Step 5: Once complete with the search it will reverse the document order (since the sort order cannot be specified in the web service request), zip the results, and update the record for the search request in the database with the save location on the server and the percent complete.

Step 6: After all search requests have been processed or it ran out of searches for the current hour, the application will update the run status in the database to indicate that it is not running.

The console application also updates the available sources on a daily basis be performing the following steps:

Step 1: Verify that the process can establish a connection to the application's database.

Step 2: Recursively get all of the sources in a folder starting from the root folder when calling the web service function "BrowseSources". For each source record it finds in the folder it will add it to a list of source objects that contains the source name, number, and folder.

Step 3: Once all of the sources have been retrieved, it will soft-delete all of the active records in the search source table of our database and then bulk insert all of the new source records as the new active set.