**Sherlock – Salesforce Global Search Utility**  
Daniel Llewellyn/Kenji776 (@Kenji776)

**Abstract:**

Sherlock is a Salesforce search replacement utility that can perform full text searches against any searchable object, including searching the content of uploaded content documents and chatter posts. The interface is provided via a jQuery plugin on a visualforce page. It is highly flexible, allowing for customized search result headers, an interface for selecting searched objects and numerous CSS hook to customize the look and feel. It also provides public methods and callbacks to easily extend the functionality.   
  
**Features:**

* Fast operation using JS remoting. No page reloads. Ajax style searching.
* Search any objects. Not limited to a single object type search
* Smart formatting adjusts search result display based on data available about the object
* Easy customization using callbacks and jQuery binds
* Flexible, can be modified to suit many different requirements
* Easy to use jQuery plugin based installation
* Efficient each search consumes only one database query
* Reliable return type makes processing search results easy
* CSS based design makes styling results quick and easy
* Structured namespaced code means smaller memory footprint and less chance of collisions
* Deployable package makes for easy install
* Over 90% code coverage with asserts provides assurance of functionality

**Installation:**

Simple install the unmanaged package from this link.

<https://login.salesforce.com/packaging/installPackage.apexp?p0=04tE0000000Qd9r>

Once the package is installed you will have a visualforce page that shows a sample invocation. In most cases simply customizing the sample page and creating a tab to access it are sufficient. You may optionally create additional visualforce pages and configure Sherlock on them as well.

**Usage/Configuration**

Due to its simple jQuery plugin based design, configuring Sherlock to meet your organizations needs is easy. Here is a sample invocation that passed in the default parameters:  
  


This invocation will call the Sherlock plugin on an element with the class searchBox. It will automatically create a search button and results container since none were provided. It will query account, intranet\_articles\_\_c and contacts by default on button click. It will re-label several objects and group accounts, intranet\_articles and contacts together. Only one instance of Sherlock is officially supported per page. You may have more, but some features may not work as expected.

**Sherlock Parameter Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Data Type | Default | Description |
| activeFilter | boolean | False | Should Sherlock search and filter as you type (true), or only after you click a search button (false). If you specify this as false and do not provide an id for the 'searchButton' param one will be created it will have the id of the search input with '\_submitButton' appended.  *Ex* 'search\_submitButton'. |
| linkTarget | string | ‘\_blank’ | The HTML target attribute to set on a clicked element. <http://www.w3schools.com/tags/att_a_target.asp> |
| searchButton | String (HTML element id) | null | The button that triggers searching if not using activeFilter. If none is provided one will be created with an ID of the search input with '\_submitButton' appended.  *Ex* 'search\_submitButton'. |
| contentTarget | String (HTML element id) | null | Where should the search results be written to? If this is left null, or has an invalid value, a results container will be created with an ID of the search input with '\_results' appended.  *Ex* 'search\_results'. |
| objectLabels | Object (key value pair) | FeedItem:Chatter  ContentVersion:Documents  User: People  Intranet\_Article\_\_c:Articles | Custom labels for the sObjects found. Simply provide the name of the sObject, including \_\_c for custom objects, and a label to use for that object. The label will be used as the header for that grouping of search results. By default Sherlock uses labels to match the challenge requirements.    You can combine various sObject results into a single category by providing the same label for multiple object types. An example of this is included shown above, where accounts, contacts and intranet\_articles\_\_c are all grouped together under the header 'articles' |
| categoryLimit | Integer | 20 | Limit of how many items per category/sObject type can be found. It would be wise to keep this somewhat low for better performance, especially when using the activeFilter param. This is per sObject type, not per category (if grouping multiple sObjects into one category) or for the entire query. |
| default Search | Array (array of sObject names) | Account Intranet\_Article\_\_c | What object types should be searched by default (aside from Chatter posts, documents and uses)? Invalid entries will be ignored. |
| onComplete | function | Empty function | Function to call after a search has completed and results have been drawn. Can be used to create additional binds on created elements or change styles on the fly. |

**Sherlock CSS Classes**

|  |  |
| --- | --- |
| **Selector** | **Description** |
| .searchPageHeader | The header at the top of the search page. |
| .searchResultContainer | The container div which houses all the search results. |
| . searchResultCategory | A container for all the search results inside for a single category. |
| .searchCategoryHeader | The title at used for the category header. |
| .searchResultList | The unordered list element which contains the search result list item links. |
| .searchResultList li / .searchResultListItem | A single list item in the unordered list. They can be accessed via the child selector (searchResultList li) or directly via .searchResultListItem. Useful for binding jQuery events. |
| .searchResultPhotoContainer | A div containing the thumbnail photo for a search result. |
| .searchResultSpacer | A spacer between the category header (.searchCategoryHeader) and the list of items (.searchResultList). |
| .searchResultClear | A utility class that clears all floated divs. Used for any search result with a thumbnail photo to prevent overlaps. |
| .searchButton | Class applied to the automatically generated search button when not using activeFiltering and no button is provided during invocation. |
| .objectSelect | A single checkbox for selecting which object(s) to search against. |
| .searchResultItemPhoto | The IMG element used to display the thumbnail photo in the search results. |
| .searchResultItemTitle | The title portion of the search results. |
| .searchResultLink | The link element for the for search results. These links are inside of spans that have thesearchResultItemTitle class. |
| .searchResultItemDescription | The description portion of the search results. |
| .searchResultItemSize | The size in kilobytes portion of the search results. |

**Sample Search Result Output**

This is a sample of the structure of the output generated by Sherlock for listing search results. The root element is the div specified by the contentTarget attribute, or the div created if one was not provided. Anything inside {} indicates a variable that would be replaced with pertinent data during runtime.

  
This illustrates the basic structure of search results. Of course there would normally be many more entries, based on how many search results and categories there were.

**Public Method overview**

Sherlock exposes several of its own methods to help you further extends its functionality or customize it to your needs. Before using any of these methods you must first invoke the Sherlock plugin.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Arguments** | **Returns** | **Description** |
| $.fn.sherlock.runSearch | String - searchterm  Function - callback | Object – An object with property for each category. Each property contains an array of search result hits. | Performs a search using the given search term and the data provided at invocation. |
| $.fn.sherlock.drawSearchResultTable | Object - An object with property for each category. Each property contains an array of search result hits.  Function - callback | A reference to the created table if a callback function is provided. Otherwise null. | Takes a list of search result data generated by runSearch and creates a search result table using the data provided an invocation. |
| $.fn.sherlock.getSearchResults | null | Object – an object containing all the search results of the last search | Returns all the search results of the last search, keyed by record ids. This object does not have the same structure as the search results, but does contain all the same information. Useful for getting additional information about a search result record. |

**Customizing Returned Data**

By default Sherlock will allow you to query for any searchable sObject in your org. However, beyond the pre-configured types of FeedItem, User, and ContentVersion the results returned are pretty plain. They simple just show the name of the object that matched the search and allow you to click it as a link. If you want further control over search results you may modify the Apex class. All search hits generate a searchResult object in which you may set the thumbnail photo, title, description, link, content size, type, record id, and include a copy of the sObject. To do so just create a branch in the else/if condition logic in the inner loop for that object type you wish to customize the results for. There are several samples of doing this in the Apex class that should be able to show you how it’s done.

**Customizing the Interface by Using callbacks**

By using the onComplete callback function you can also attach custom jQuery events to created elements in the table. The onComplete function will return a reference to the created search results which can be modified to your liking. You could combine this with the $.fn.sherlock.getSearchResults function to create nearly any kind of additional functionality you might need from your search interface.

There you have it. Hopefully you have as much fun using and working with Sherlock as I did writing it, or at the very least don’t hate it :P

-Dan/Kenji