Genesys Knowledge Center Developer's Guide

Knowledge Center 8.5.0

2/23/2022

Table of Contents

Genesys Knowledge Center Developer's Guide

> **Important**
>
> Genesys Knowledge Center is now available as a restricted offering. You must contact your Genesys representative to see if Genesys Knowledge Center is suitable for your environment and business needs. The documentation here anticipates a larger rollout of Genesys Knowledge Center in late 2015.

Welcome to the Genesys Knowledge Center 8.5.0 Developer's Guide. This document provides information about how you can integrate Knowledge Center for your website and environment. See the summary of chapters below:

Integration

Find out how Knowledge Center works with other Genesys components.

Integrating with Genesys Web Engagement

Using Knowledge On your Web Site

Learn how to add Knowledge Center to your web resources.

Simple Integration for Pre-Production Environment

Sample UI

Find out more information on the User Interface, including Knowledge Agent and Widgets.

Sample UI

Simple Integration for Pre-Production Environment

Overview

This chapter describes the integration steps that allows you to add Knowledge Center functionality to your site without modifying any memory. To configure the integration you need to use Proxy shipped with Genesys Web Engagement product.

The GWM Proxy is a development tool that you use to add new functionality to a website without directly modifying that site. Once you have configured this proxy, you can use the Genesys Knowledge Center Sample UI from any of your websites. In a few clicks, without modifying your website, the Knowledge Center Sample UI features shows up on a set of web pages, according to the rules and categories that you created. There are two proxy tools available in the Web Engagement installation, the Simple tool and the Advanced tool. Within this instruction you need to use the Advanced GWM Proxy. For more information regarding proxy please refer to the Genesys Web Engagement documentation.

> **Important**
>
> GWE Proxy provides support for easy integration into existing sites within the pre-production environment. It is not recommended to use it in a production environment. Please use similar tools available on the market.

Configuring the Advanced GWM Proxy

The Advanced GWM Proxy is based on the OWASP Zed Attack Proxy Project (ZAProxy). In addition to acting as a proxy, the Advanced GWM Proxy also provides a UI and validates vulnerabilities in your website at the same time.

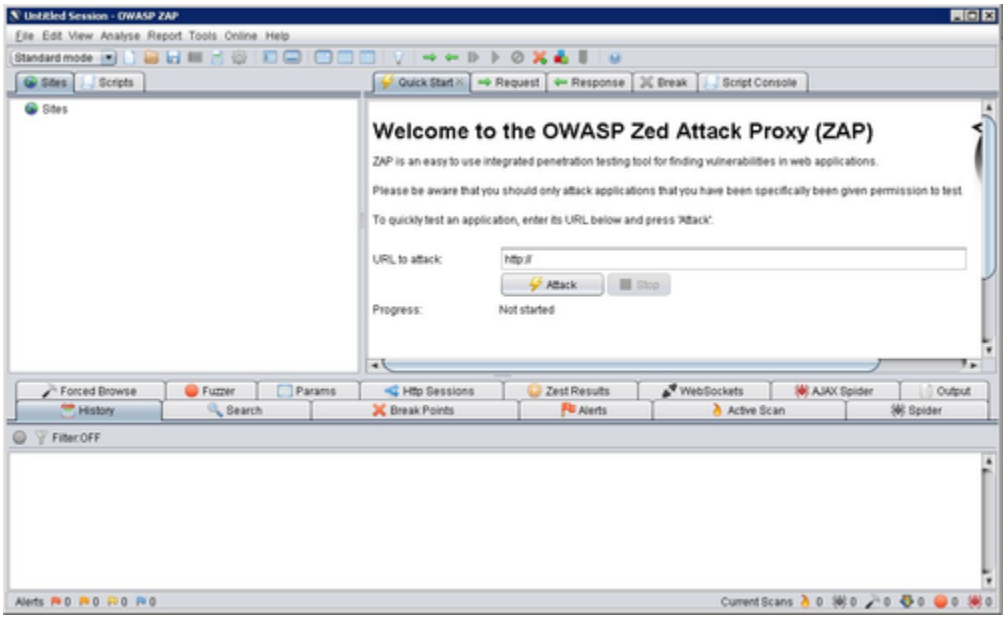
> **Important**
>
> The Advanced GWM Proxy requires JDK 1.7 or higher.

Before you start using the Advanced GWM Proxy, you need to carry out a few configuration procedures.

Starting the Proxy

Navigate to your Web Engagement installation directory and launch either *servers\proxy2\zap.bat* (on Windows) or *servers\proxy2\zap.sh* (on Linux).

The proxy starts.



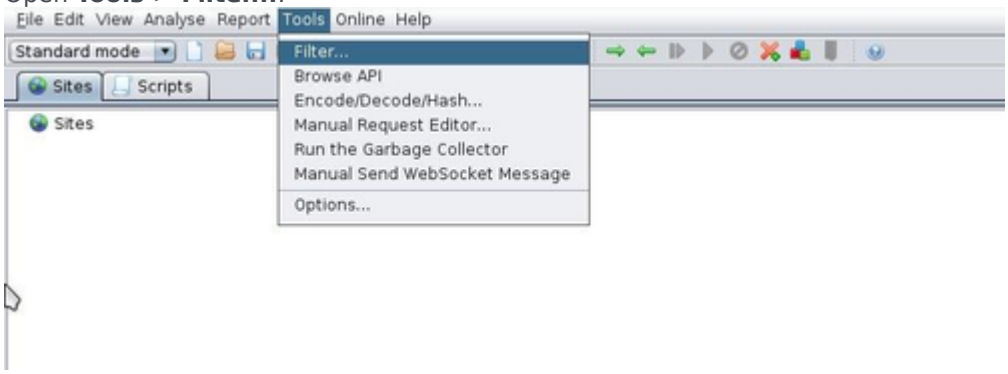
The Advanced GWM Proxy

Configuring the Proxy

Once the proxy is running, you can configure it using the GUI.

Start

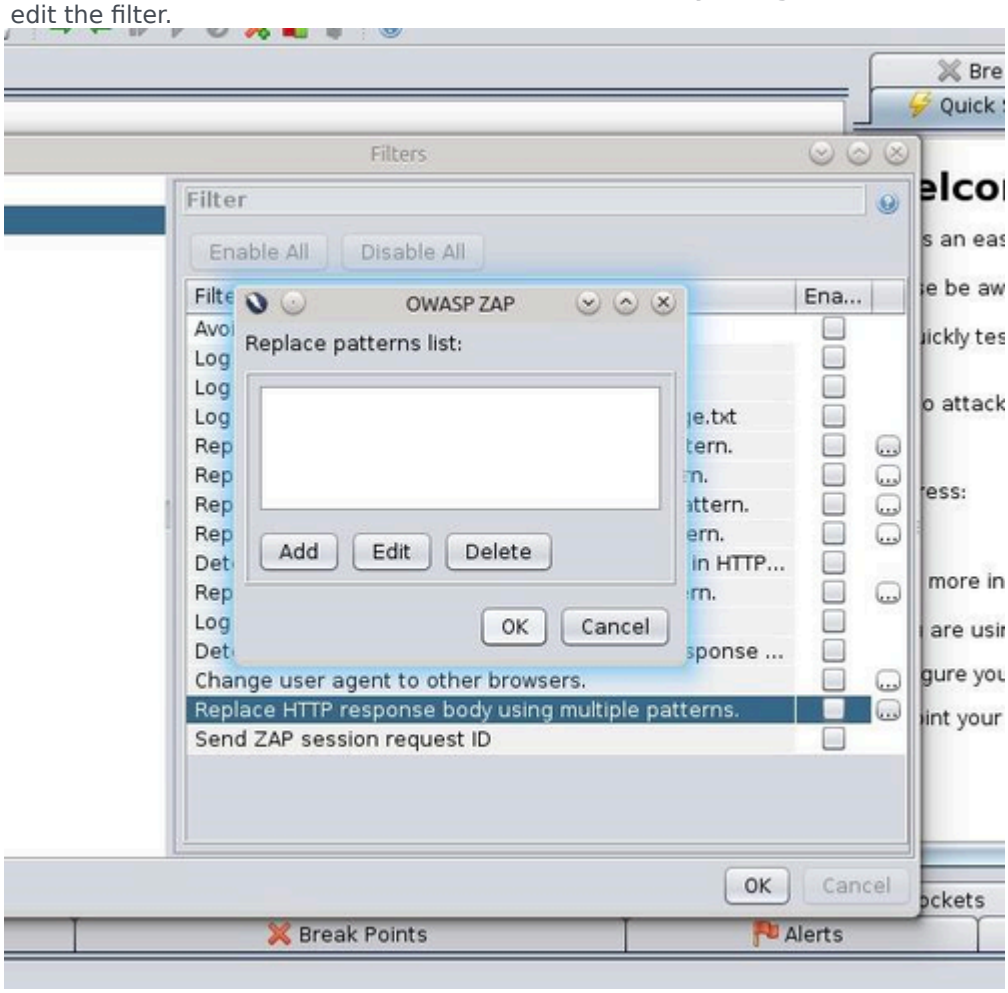
1. Open **Tools > Filter...**.



Configuring the Proxy Filter

Select the Filter menu item.

2. In the list of filters, select **Replace HTTP response body using multiple patterns** and click **...** to edit the filter.



List of Proxy Filters

Select the filter.

3. Click **Add**.
4. Enter **</head>** in the **Pattern:** field of the resulting dialog box.
5. Enter the following code in the **Replace with:** field.

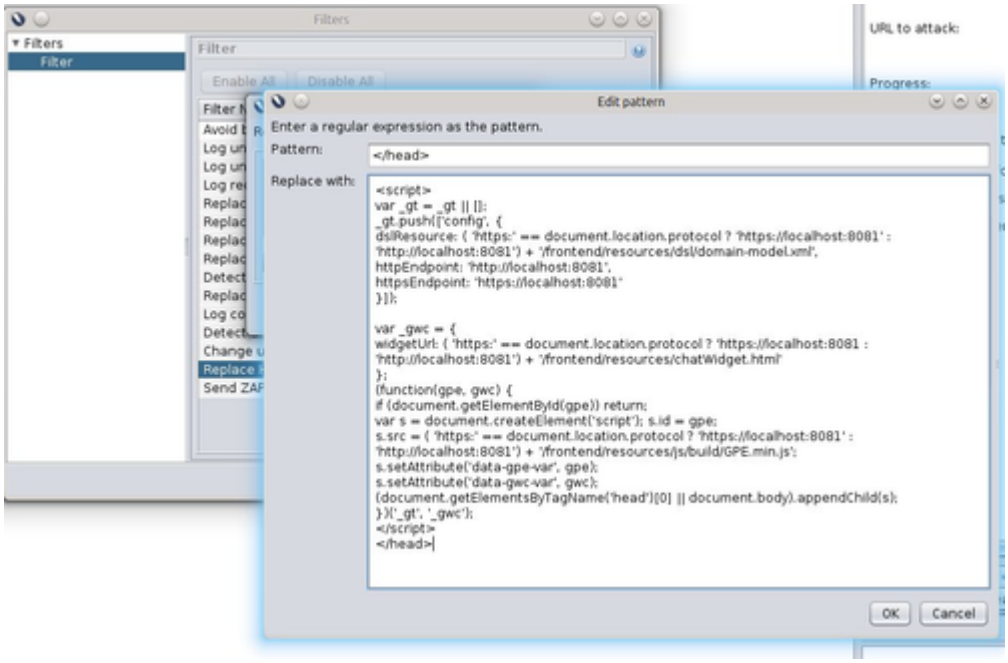
```
<meta charset="UTF-8">
 <meta content="IE=Edge" http-equiv="X-UA-Compatible">
 <title>Slide box</title>
 <style>
 .gkc-inject-body {
      position: fixed;
      top: 0;
      right: -1001px;
      width: 1000px;
      transition: right 300ms cubic-bezier(0.17, 0.04, 0.03, 0.94);
```

```
        overflow: hidden;
        float: right;
        border-left: 1px solid #eee;
        border-bottom: 1px solid #eee;
        background-color: #fff;
        height: 100%;
        z-index: 1000001;
    }
    .gkc-inject-body iframe {
        border: none;
        height: 100%;
        width: 100%;
    }
    #gkc-toggle {
        display: none;
    }
    #gkc-toggle + label:after {
        font-family: "Helvetica Neue", Helvetica, sans-serif;
        text-transform: uppercase;
        transform: rotate(90deg);
        transform-origin: right top 0;
        background-color: #31b0d5;
        color: #fff;
        margin-top: 150px;
        padding: 2px 10px;
        float: left;
        cursor: pointer;
        position: fixed;
        top: 0;
        right: 0;
        transition: right 300ms cubic-bezier(0.17, 0.04, 0.03, 0.94);
        z-index: 1000001;
    }
    #gkc-toggle + label:after {
        content:"Knowledge"
    }
    #gkc-toggle:checked ~ .gkc-inject-body {
        right: 0;
    }
    #gkc-toggle:checked + label:after {
        right: 1000px;
    }
    </style>
    </head>
```

6. Click **OK** to save the pattern.
7. Click **Add** again.
8. Enter **</body>** in the **Pattern:** field of the resulting dialog box.
9. Enter the following code in the **Replace with:** field, replacing *your.sample.ui.host:port* with the host and port where you have configured the Sample UI.

```
    <div class="gkc-container" >
        <div class="gkc-slide-box" >
            <input type="checkbox" name="toggle" id="gkc-toggle" />
            <label for="gkc-toggle"></label>
            <div class="gkc-inject-body" style="background-color: white">
                <iframe src="http://your.sample.ui.host:port/gks-sample-ui/" width="980"
height="800">
                    Sorry your browser does not support iFrames
                </iframe>
            </div>
```

```
    </div>
</div>
</body>
```



Entering a Filter Pattern

10. Click **OK** to save the pattern.

End

Configuring the URL Filter

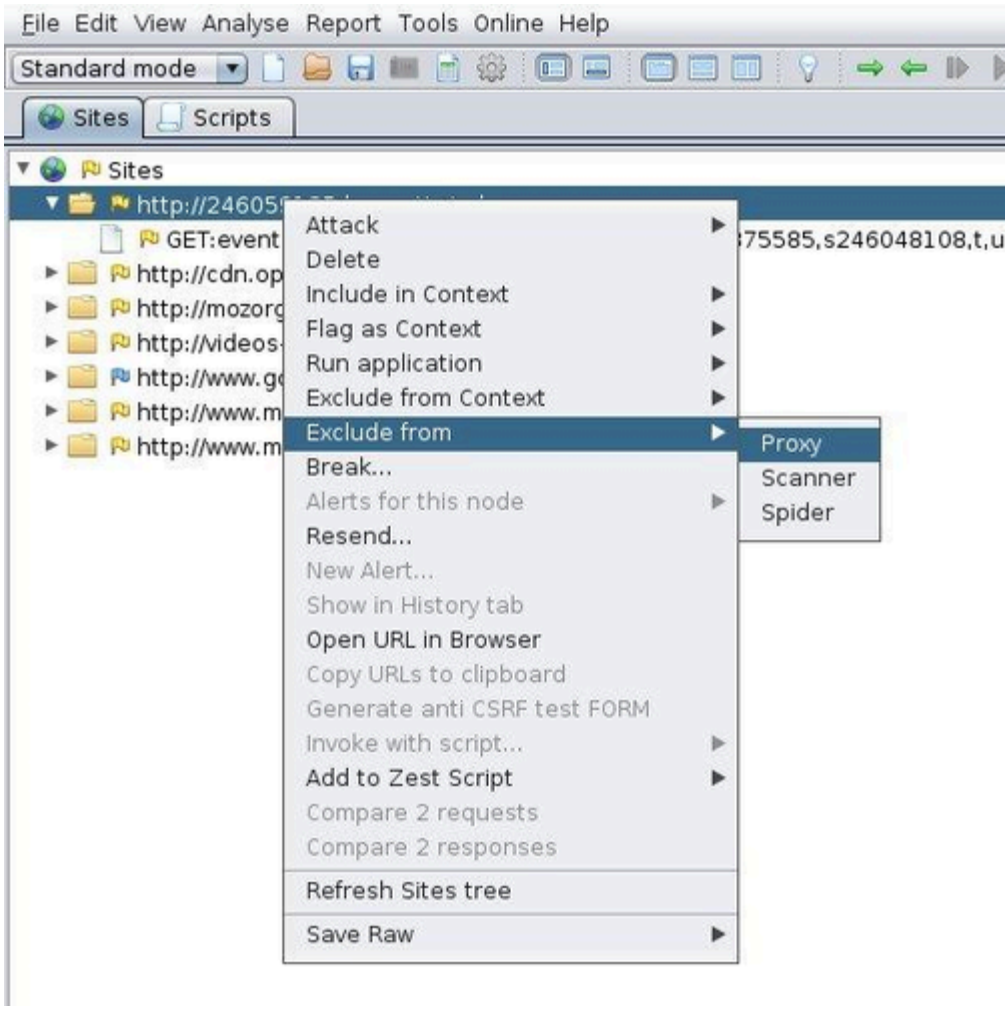
Complete this procedure to use the GUI to configure URLs that the proxy should ignore.

Start

You can exclude a site in one of two ways:

Using the Sites Tab

1. In the **Sites** tab, right-click on a site and select **Exclude from > Proxy**.



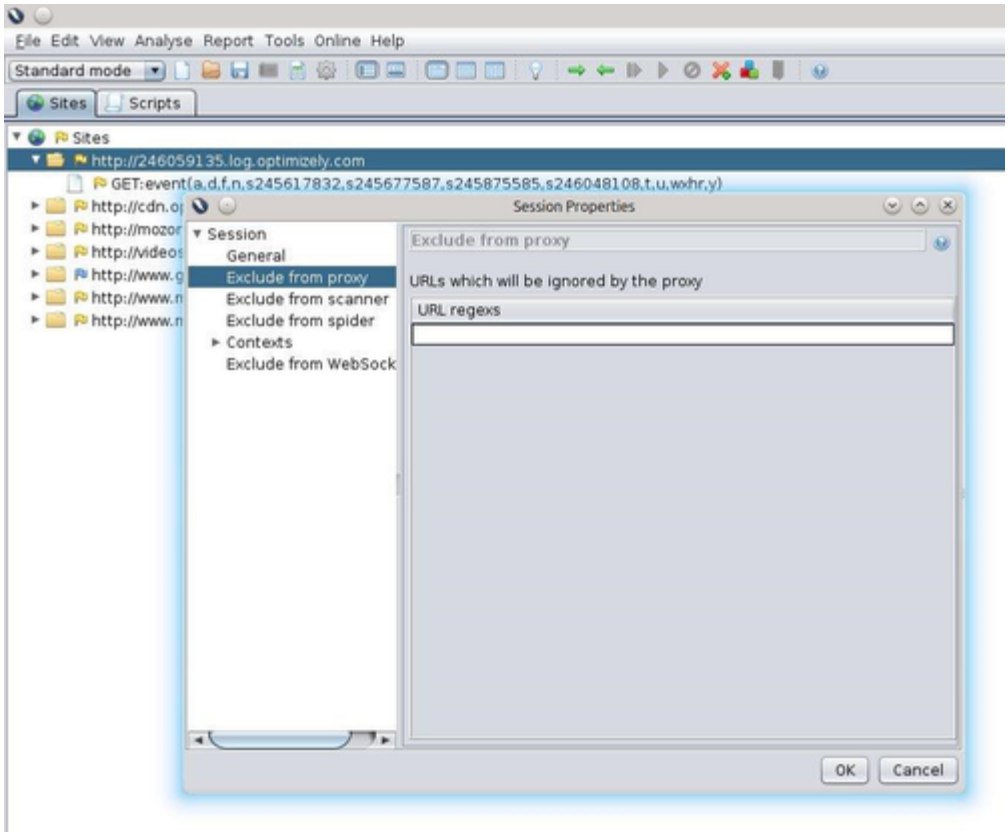
Excluding a Site from the Proxy Filter

2. Select a site to exclude.

Using the File Menu

1. Select **File > Properties**. In the **Session Properties** window, select **Exclude from proxy**, add your URL regular expression, and click **OK**. For example, to have the proxy include only the **google.com** website, use this regular expression:

```
^((?!google.com).)*$
```



Adding a Regular Expression for Ignoring Sites

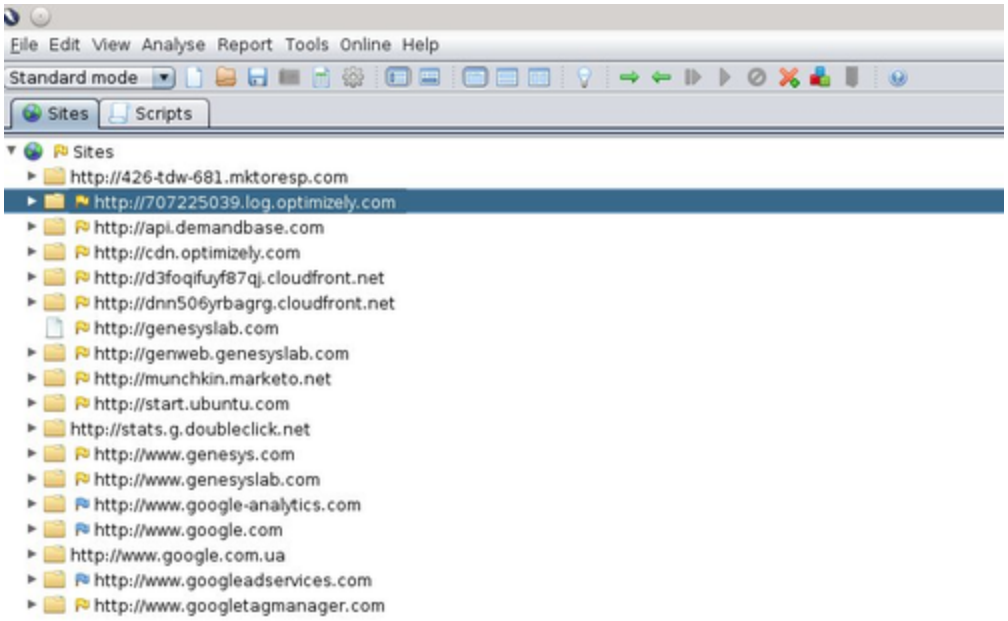
2. Enter a URL to exclude.

If you want the proxy to remember the excluded URLs beyond the current session, select **File > Persist session...** and select a file to save your session to.

End

Working with the Proxy

After you have configured the proxy, keep it open and then open up a web browser. Now you can browse through the web pages that are instrumented with the Knowledge Center Sample UI, and they will be displayed in the **Sites** tab of the proxy GUI, as shown here:



Browsing Your Proxy Sites

For more information about working with ZAProxy, see https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

Integrating with Genesys Web Engagement

Overview

When you integrate Knowledge Center with Genesys Web Engagement, you are giving your agents access to important proactive engagement capabilities. Knowledge Center (and the way you interact with) it allows you to better understand your customer needs and intentions. For example, monitoring customer activities with Knowledge Center on the corporate web site allows you to find the right moment to propose agent help when the customer appears to be lost. When such an interaction appears on an Agent workspace, all the customer requests and browsing history are made available. This is one of the many reasons why you might want to integrate Knowledge Center with Genesys Web Engagement in your environment.

Tight integration between Knowledge Center and Web Engagement allows you to monitor customer activities on your web site (both browsing and working with knowledge). It also defines customer behavior patterns and actions that should take place when patterns occur (including both immediate contact with an agent or postponed processing of the activity).

Here are some examples of the patterns you could look for and suggested reactions:

- Customer indicates that they cannot find the answer to the question. A suggested reaction for this event is the chat option with the agent (how to configure such integration is shown in the example below).
- A Premium customer has left negative feedback on one of the documents he viewed. A suggested reaction for this event is a follow-up call to maintain the relationship with the customer.
- While browsing throughout the site a customer has expressed interest in establishing a new service with the company. A suggested reaction for this event is to do a follow-up and check whether or not the customer has successfully set-up the new service and then send a note of thanks for being a loyal customer.

To integrate products in your environment you need to add Knowledge Center-specific events into the Web Engagement DSL file which describes business events for a given website. All other steps are standard for installation of Genesys Knowledge Center and Genesys Web Engagement.

Sample DSL

KnowledgeCenter.DSL provides a basic set of events that are used in your integration. Events are based on the Sample UI GUI shipped with the product.

DSL file contains following events:

- Open a category in browsing
- Viewing of search results

- Open document for viewing content
- Leaving positive and negative feedback
- Requesting additional help (no aster found)

Engaging chat with agent when no answer found

Follow the instructions below to configure this integration.

Start

1. Install and properly configure Genesys Web Engagement, using the GWE Deployment Guide.
2. Create a Knowledge Center application in GWE.
3. Create a DSL file that describes your site's business logic. You can either use the **Intool** provided with GWE or use the standard DSL for the Sample UI that is provided with Knowledge Center. Replace the standard GWE content by the new DSL that is included at *GWE root folder*\apps\gks_composer-project\WebEngagement_EngagementWidgets\dsl\domain-model.xml.

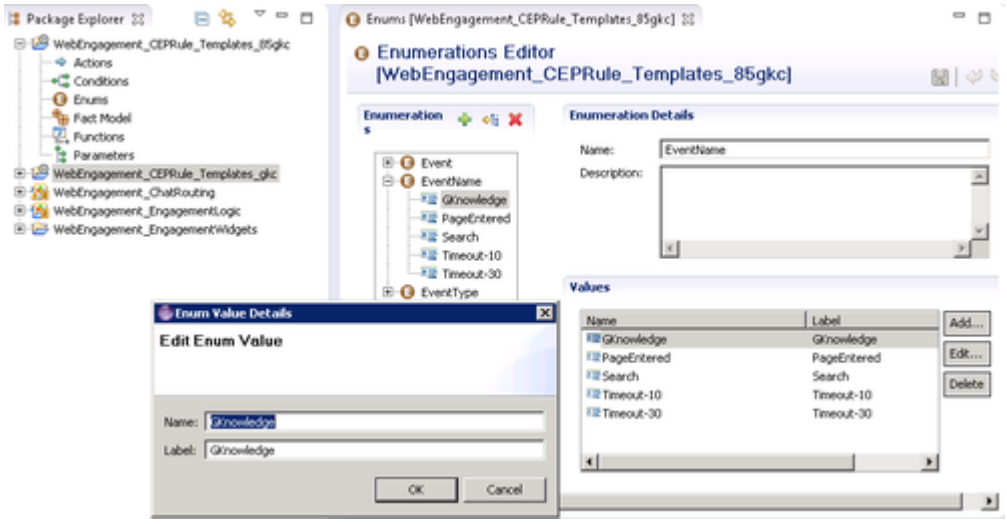
GWE Integration Tool

Here is a sample DSL file:

```
<?xml version="1.0" encoding="utf-8"?>
<properties>
     <events>
        <event id="NoRelevantResults" name="GKnowledge">
           <trigger name="NoResultsTrigger" element="DIV._gk-_wd-_rs-no-answer A"
action="click" url="" count="1"/>
           <val name="gks_question" value="$('#searchContent').val()"/>
           <val name="gks_kbId" value="'knowledgeFAQ'"/>
           <val name="gks_session" value="window.localStorage.getItem('sessionId')"/>
        </event>
     </events>
</properties>
```

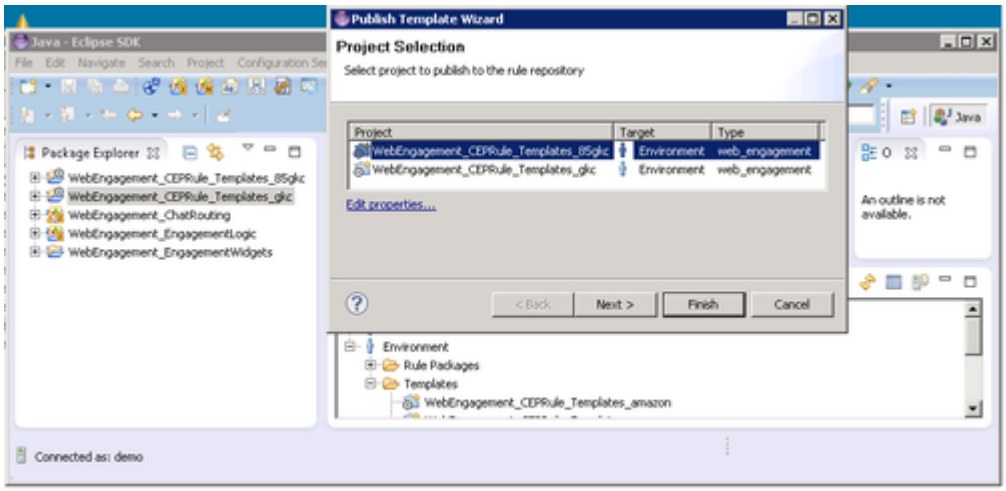
4. In Composer, modify the Web Engagement templates, which will be either **WebEngagement_CEPRule_Templates** (if you use GRAT 8.1.3) or **WebEngagement_CEPRule_Templates_85** (if you use GRAT 8.5).

Add new event names to the Enums. In the above example, we used an event name of *GKnowledge*.



Editing an Enum Value

5. Publish **CEPRule_Templates** to the GRS repository.

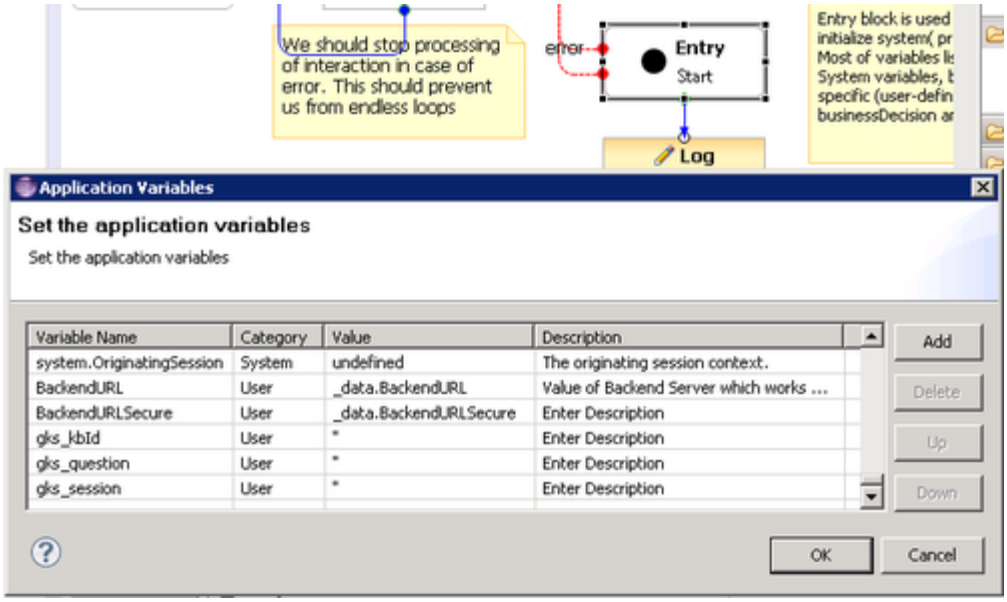


Publishing a Project

6. Create a business rule based on your custom DSL and on **CEPRule_Templates**. For example:

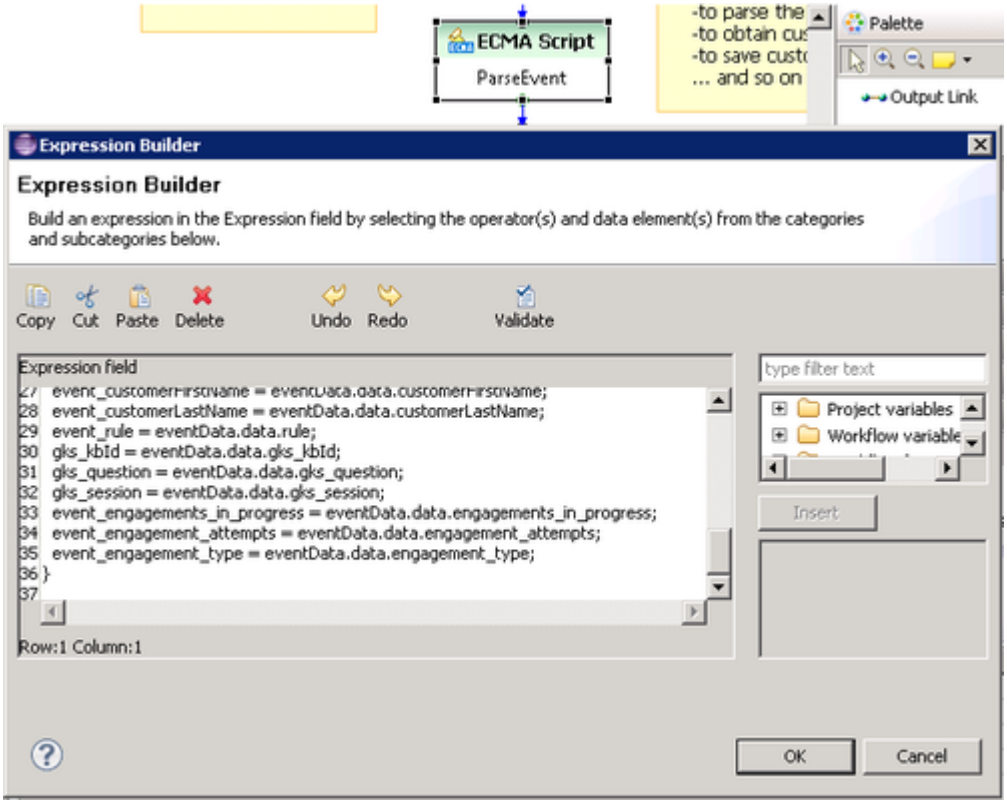
```
rule "Rule-100 No Relevant Results"
salience 100000
    agenda-group "level0"
    dialect "mvel"
    when
        $event1: Event(eval($event1.getName().equals('NoRelevantResults')))
    then
        sendEvent($event1, ed, drools);
end
```

7. Modify **default.workflow** in the **WebEngagement_EngagementLogic** Composer project.
   Add new user variables, **gks_kbId**, **gks_question**, and **gks_session**, to the **Entry (Start)** block:



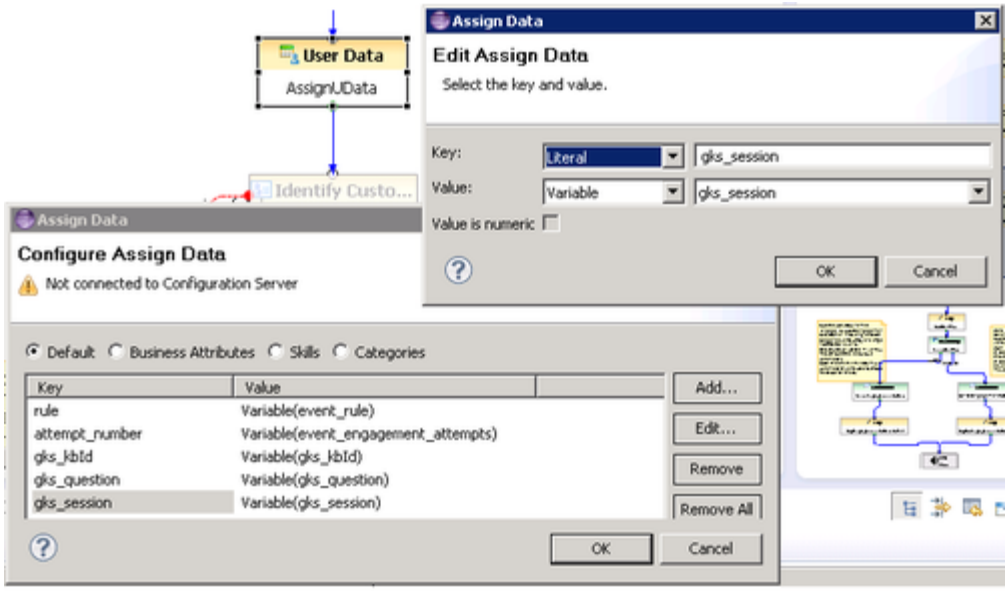
Adding New Variables

8. Add parsing for new variables to the **ECMA Script (ParseEvent)** block:



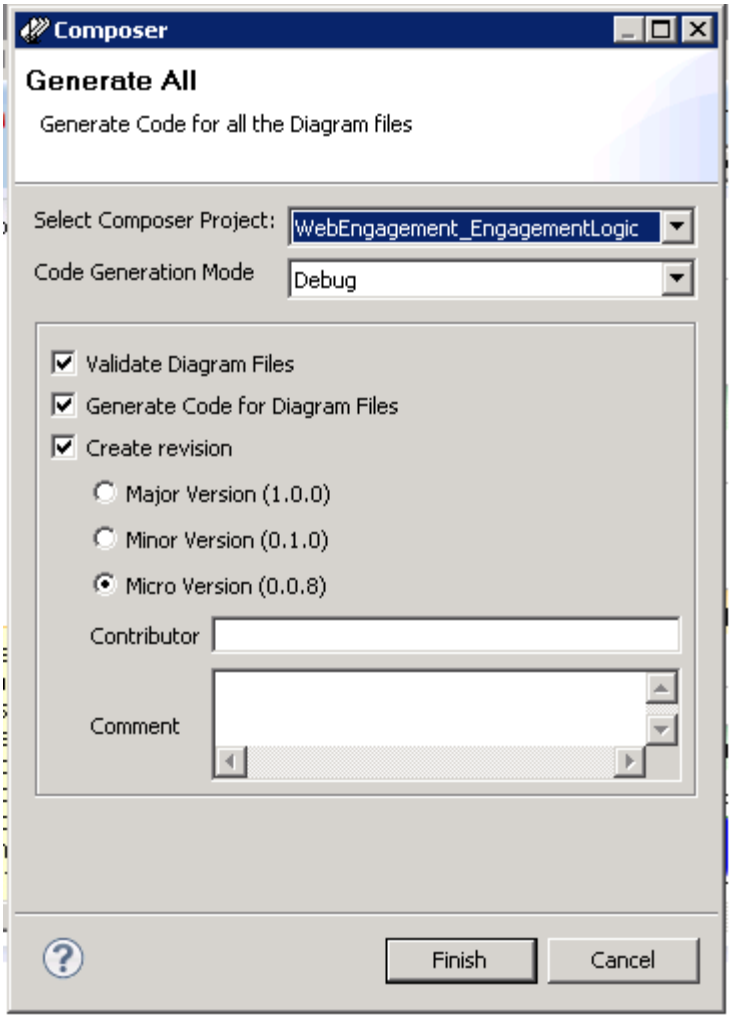
ECMA Script for Event Parsing

9. Add parsed data to the interaction in the **User Data (AssignUData)** block:



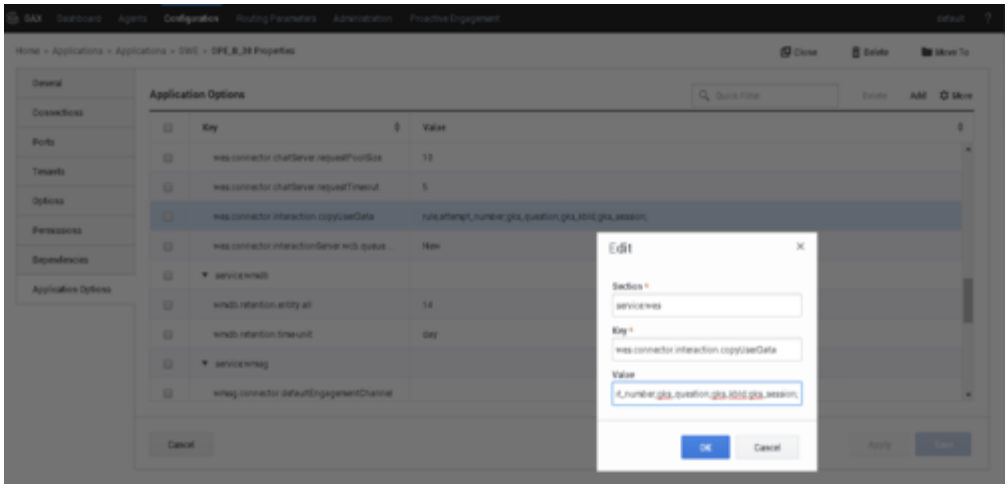
Add Parsed Data to Interaction

10. Save **default.workflow** and generate new SCXML strategies by clicking the **Generate All** button:



Generate SCXML Strategies

11. Build the Knowledge Center Server application (run **build gks**).
12. Deploy the Knowledge Center Server application (run **deploy gks**).
13. Modify the GWE backend Config Server application. Add new variables, **gks_question**, **gks_kbId**, and **gks_session**, to the **wes.connector.interaction.copyUserData** option.



Add Options to GWE Backend Server

14. Deploy the business rule created in Step 6, above, to GWE storage.
15. Run the GWE servers.

End

To allow GWE to access the Knowledge Center UI, you need to modify either your site or the Sample UI by adding a Web Monitoring Agent script similar to the following sample to the source code of your web UI application.

```
<script>
    var _gt = _gt || [];
    _gt.push(['config', {
       dslResource : ('https:' == document.location.protocol ? 'https://<host>:<port1>' :
'http://<host>:<port2>') + '/frontend/resources/dsl/domain-model.xml',
       httpEndpoint : 'http://<host>:<port2>',
       httpsEndpoint : 'https://<host>:<port1>'
    }]);

    var _genesys = {
       chat: {
          serverUrl: 'http://<host>:<port3>/backend/cometd',
          registration: true
       },
       debug: true
    };

    (function(d, s, id, o) {
        var fs = d.getElementsByTagName(s)[0], e;
        if (d.getElementById(id)) return;
        e = d.createElement(s); e.id = id; e.src = o.src;
        e.setAttribute('data-gcb-url', o.cbUrl);
        fs.parentNode.insertBefore(e, fs);
    }) (document, 'script', 'genesys-js', {
       src: "http://<host>:<port2>/frontend/resources/js/build/genesys.min.js",
    });
</script>
```

> **Important**
>
> To make the integration work, you need to run both the GWE backend and frontend servers.

For more detailed instructions, refer to the GWE documentation.

Sample UI

Overview

The Sample UI is based on backbone.js and divided into three parts:

- Knowledge Agent — low level mapper that covers Knowledge API and encapsulate Knowledge session management.

location: gks-sample-ui.war/modules/knowledge_agent/

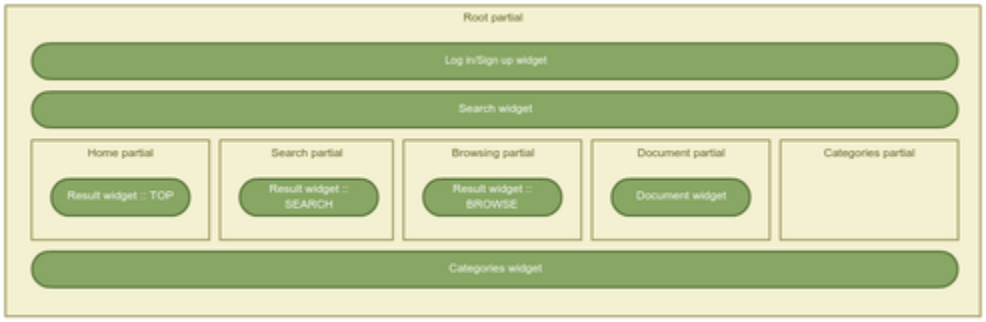
- UI Widgets — atomic modules that responsible for key UI elements (such as: search panel, search result view, document view).

location: gks-sample-ui.war/modules/widgets/

- The Sample-UI itself — combination of the first two and the logic of their interactions.

location: gks-sample-ui.war/

Templates Hierarchy



Templates hierarchy and available widgets

Page Descriptions

Name	Routing	Description
Home Page	#	The welcome page of the Sample UI. Displays list of Top questions and associated categories.

Name	Routing	Description
Search result page	*#category/:categoryId/search/:searchQuery*	The results of a search. Displays relevant documents based on search query and associated to them categories.
Browsing page	*#category/:categoryId/search/*	Categories browsing. Displays documents in a specific category as well as other documents associated to them in other categories.
Document page	*#kbId/:kbId/document/:documentId*	Full document info. Displays full content of the specific document. In case of click directly after search, this page also contains a voting area and a help button.
Categories page	*#categories*	A list of available categories. Displays all available categories and provides their browsing capability.

Knowledge Agent

Overview

Knowledge agent is an AMD-based module that can be used with RequireJS. It exports the _gka variable into the local context where it is accessed.

Configuration

_gka.initialize(options)	Examples
Description: Configure the Knowledge agent. • **Options** Type: PlainObject A set of key/value pairs that configure the Agent. • **host** Type: String A network host where Knowledge API is stored. • **kbId** Type: String Knowledgebase default identifier to be used. • **lang**(default: EMPTY) Type: String Default language to be used. • **subTenantId** (default: EMPTY) Type: String Sub tenant identifier	_gka.initialize({ host: 'http://localhost:8080', kbId: 'knowledge' })

Knowledge Agent API

Once configuration is complete, the Agent can receive data from the Knowledge API. The _gka variable contains the following interfaces:

Method	Description
Knowledge Base Operations	

Method	Description
.getKnowledgeBases()	Retrieves list of knowledge bases supported
.getKnowledgeBaseInfo()	Retrieves information about the particular knowledge base
.getCategories()	Retrieves list of categories
.getFullContent()	Retrieves full content of particular document
FAQ retrival	
.search()	Executes search for the answer for the given query
.suggestions()	Provides autocomplete functionality
.getSimilarContent()	Retrieves similar content for the document provided
.guessSpelling()	Guess spelling correction for the entered query
Feedback	
.noAnswer()	Marks query as the one that do not have valid answer in knowledge base
.like()	Records user rating for the document within query
.visit()	Registers viewing the document

Knowledge Base Operations

> **Important**
>
> For additional information, please refer to the Knowledge API page.

_gka.getKnowledgeBases(): Promise	**Example**

Description: Retrieves information about the particular knowledge base.	_gkn.getKnowledgeBases().done(function(response, status) {
	`console.log(response)`
	}).fail(function(error, status) {
	`console.log(error)`
	});

Response	

• **knowledgebases**
Type: PlainObject[]
List of supported knowledgebases
• **name**
Type: String
Name of knowledgebase
	{
	`"knowledgebases": [{`
	`"name": "knowledgefaq",`
	`"languages": [`
	`"en"`
	`]`
	`}, {`
	`"name": "the_bank",`
	`"languages": [`
	`"en"`

_gka.getKnowledgeBases(): Promise	**Example**
• **languages** Type: String[] List of supported languages	] }] }

_gka.getKnowledgeBaseInfo([options]): Promise	**Example**

Description: Retrieves information about the knowledge base.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
• **kbId** (default: stored _gka.kbId)
Type: String
Particular knowledge base identifier.
	_gkn.getKnowledgeBaseInfo().done(function(response, status) {
	console.log(response)
	}).fail(function(error, status) {
	console.log(error)
	});

Response	

• **languages**
Type: String[]
List of supported languages for given knowledgebase
	{
	languages: [
	'en',
	'fr',
	'de'
	]
	}

_gka.getCategories(): Promise	**Example**

Description: Retrieves list of categories.	_gkn.getCategories().done(function(response, status) {
	console.log(response)
	}).fail(function(error, status) {
	console.log(error)
	});

Response	
• **categories**	{

_gka.getCategories(): Promise	**Example**

Type: PlainObject[]
List of supported categories
• **id**
Type: String
Category identifier
• **name**
Type: String
Category name
• **count**
Type: Number
Total count of documents in category
	<pre>"categories": [{
	"id": "Home Mortgage",
	"name": "Home Mortgage",
	"count": 78
	}, {
	"id": "Home Equity Basics",
	"name": "Home Equity Basics",
	"count": 78
	}]
	
	}</pre>

_gka.getFullContent(options): Promise	**Example**

Description: Retrieves full content of particular document.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
• **docId**
Type: String
Particular document identifier.
	<pre>_gkn.getFullContent({
	docId: 'knowledge'
	}).done(function(response, status) {
	console.log(response)
	}).fail(function(error, status) {
	console.log(error)
	});</pre>

Response	

• **id**
Type: String
• **language**
Type: String
• **typeName**
Type: String
• **kbId**
Type: String
• **categories**
	<pre>{
	"id":"knowledge",
	"language":"en",
	"typeName":"qna_document_en",
	"kbId":"knowledge",
	"categories":[
	"Booking trips"
	],
	"created":1402573911163,
	"modified":1402573911163,
	"snippet":"",
	"fields":{
	"id":"knowledge",
	"created":1402573911163,</pre>

_gka.getFullContent(options): Promise	Example
Type: String[] • **created** Type: Number • **modified** Type: Number • **snippet** Type: String • **fields** • **id** Type: String • **created** Type: Number • **answer** Type: String • **categories** Type: String[] • **question** Type: String • **modified** Type: Number	"answer":"Every travel option we offer", "categories":["Booking trips"], "question":"What's the difference between", "modified":1402573911163 } }

FAQ retrival

_gka.search([options]): Promise	Example

Description: Executes search for the answer for the given query.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
• **from** (default: 0)
Type: Number
Pagination offset.
• **size** (default: 10)
Type: Number
Pagination page size
	_gkn.search({
	from: 0,
	size: 10,
	query: ,
	categories: []
	}).done(function(response, status) {
	console.log(response)
	}).fail(function(error, status) {
	console.log(error)
	});

_gka.search([options]): Promise	Example
• **query** (default: *) Type: String User typed query string • **categories** (default: []) Type: String[] List of categories that is used as a context for the current query • **spellcheck** (default: 'AUTO') Type String enum ("YES", "NO", "AUTO") Spell checking strategy.	
Response	

• **count**
Type: Number
• **page**
Type: PlainObject
• **from**
Type: Number
• **size**
Type: Number
• **documents**
Type: PlainObject[]
• **id**
Type: String
• **language**
Type: String
• **typeName**
Type: String
• **kbId**
Type: String
• **categories**
Type: String[]
• **created**
Type: Number
	{
	"count": 78,
	"page": {
	"from": 0,
	"size": 10
	},
	"documents": [
	{
	"id": "knowledge",
	"language": "en",
	"typeName": "qna_document_en",
	"kbId": "knowledge",
	"categories": [
	"Home Equity Basics",
	"Home Mortgage"
	],
	"created": 1404823845989,
	"modified": 1404823845989,
	"snippet": "What is the difference\n",
	"score": 4.20981,
	"fields": {
	"question": "What is the difference",
	"answer": "Locking ensures that you"
	},
	"morelikethis": [
	
	],
	"confidence": 1.0
	},
	{
	"id": "knowledge",
	"language": "en",
	"typeName": "qna_document_en",

_gka.search([options]): Promise	Example

• **modified**
Type: Number
• **snippet**
Type: String
• **score**
Type: Number
• **fields**
Type: PlainObject
  • **question**
Type: String
  • **answer**
Type: String
• **morelikethis**
Type: String[]
• **confidence**
Type: Number
• **categories**
Type: PlainObject[]
  • **id**
Type: String
  • **name**
Type: String
  • **count**
Type: String
	<pre>"kbId": "knowledge",
	"categories": [
	"Home Equity Basics",
	"Home Mortgage"
	],
	"created": 1404823845989,
	"modified": 1404823845989,
	"snippet": "How long do I have to pay",
	"score": 4.20981,
	"fields": {
	"question": "How long do I have",
	"answer": "If you obtained your"
	},
	"morelikethis": [
	
	],
	"confidence": 1.0
	}
	],
	"categories": [
	{
	"id": "Home Equity Basics",
	"name": "Home Equity Basics",
	"count": 78
	},
	{
	"id": "Home Equity Rates & Services",
	"name": "Home Equity Rates & Services",
	"count": 2
	}
	]
	
	}</pre>

_gka.suggestions(options): Promise	Example

Description: Provides autocomplete functionality.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
  • **query**
Type: String
User typed query string.
	<pre>_gkn.suggestions({
	query: 'ipad'
	}).done(function(response, status) {
	console.log(response)
	}).fail(function(error, status) {
	console.log(error)
	});</pre>

Response	

_gka.suggestions(options): Promise	Example

• **suggestions**
Type: String[]
	{
	"suggestions":[
	"What else do I need to know at check-in",
	"What is a Non-Sufficient Funds fee",
	"What's a 401(k) plan?\n",
	]
	}

_gka.getSimilarContent(options): Promise	Example

Description: Retrieves similar content for the document provided.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
• **docId**
Type: String
Particular document identifier.
	_gkn.getSimilarContent({
	docId: "knowledge"
	}).done(function(response, status) {
	console.log(response)
	}).fail(function(error, status) {
	console.log(error)
	});

Response

- **count**

  Type: Number
- **suggestions**
  Type: String[]
- **page**
  Type: PalinObject
  - **from**
    Type: Number
  - **size**
    Type: Number
- **documents**
  Type: PlainObject[]
  - **id**
    Type: String
  - **language**
    Type: String

_gka.getSimilarContent(options): Promise	Example
• **typeName** Type: String • **kbId** Type: String • **categories** Type: String[] • **created** Type: Number • **modified** Type: Number • **snippet** Type: String • **score** Type: Number • **fields** Type: PlainObject • **question** Type: String **answer** Type: String • **morelikethis** Type: String[] • **confidence** Type: Number	

_gka.guessSpelling([options]): Promise	Example

Description: Guess spelling correction for the entered query.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
• **number** (default: 1)
Type: Number
Number of suggestions to be proposed.
	_gkn.getSimilarContent({
	`number: 10`
	}).done(function(response, status) {
	`console.log(response)`
	}).fail(function(error, status) {
	`console.log(error)`
	});

_gka.guessSpelling([options]): Promise	Example
Response	

• **suggestions**
Type: PlainObject[]
  • **word**
Type: String
  • **suggestions**
Type: String[]	

Feedback

_gka.noAnswer(options): Promise	Example

Description: Marks query as the one that do not have valid answer in knowledge base.
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
  • **from**
Type: Number
Pagination offset.
  • **size**
Type: Number
Pagination page size
  • **query**
Type: String
User typed query string
  • **categories**
Type: String[]
List of categories that is used as a context for the current query
  • **spellcheck**
Type String enum ("YES", "NO", "AUTO")
Spell checking
	_gkn.noAnswer ({
	`from: 0,`
	`size: 10,`
	`query: ,`
	`categories: []`
	}).fail(function(error, status) {
	`console.log(error)`
	})

_gka.noAnswer(options): Promise	Example
strategy	

_gka.visit(options): Promise	Example

Description: Registers viewing the document.
 	
• **options**
Type: PlainObject
A set of key/value pairs that contains arguments for the RESTful API.
  • **docId**
Type: String
Particular document identifier.
	_gkn.visit({
	  docId: "knowledge",
	}).fail(function(error, status) {
	  console.log(error)
	});

Promise Object

The object returned by all methods of _gka variable implements the Promise interface, giving them all the properties, methods, and behaviour of a Promise (see jQuery Deferred for more information). It represents a value that may not be available yet.

promise.done(doneCallbacks [, doneCallbacks]): Promise	Example

Description: Add handlers to be called when the Deferred object is resolved.
 	
• **doneCallbacks**
Type: Function(response, status)
A function, or array of functions, that are called when the Deferred is resolved.
 	
• **doneCallbacks**
Type: Function(response, status)
Optional additional functions, or arrays
	promise.done(function(response, status) {
	  alert('ajax call succeeded');
	  console.log(response);
	  console.log(status)
	});

promise.done(doneCallbacks [, doneCallbacks]): Promise	Example
of functions, that are called when the Deferred is resolved.	

promise.fail(failCallbacks [, failCallbacks])): Promise	Example

Description: Add handlers to be called when the Deferred object is rejected.
• **doneCallbacks** Type: Function(error, status) A function, or array of functions, that are called when the Deferred is rejected.
• **doneCallbacks** Type: Function(error, status) Optional additional functions, or arrays of functions, that are called when the Deferred is rejected.
	promise.done(function() {
	alert('ajax call succeeded');
	}).fail(function(error, status) {
	alert('ajax call failed');
	console.log(error);
	console.log(status)
	});

promise.always(alwaysCallbacks [, alwaysCallbacks]): Promise	Example

Description: Add handlers to be called when the Deferred object is either resolved or rejected.
• **doneCallbacks** Type: Function(response|error, status) A function, or array of functions, that is called when the Deferred is resolved or rejected.
• **doneCallbacks** Type:
	promise.always(function(responseOrError, status) {
	alert('ajax call completed with success or error callback arguments');
	console.log(responseOrError);
	console.log(status)
	});

promise.always(alwaysCallbacks [, alwaysCallbacks]): Promise	Example
Function(response|error, status) Optional additional functions, or arrays of functions, that are called when the Deferred is resolved or rejected.	

promise.state(): String

Description: Determine the current state of a linked Deferred object.

The .state() method returns a string representing the current state of the Deferred object. The Deferred object can be in one of three states:

- **pending**: The Deferred object is not yet in a completed state (neither "rejected" nor "resolved").
- **resolved**: The Deferred object is in the resolved state, meaning that the doneCallbacks have been called (or are in the process of being called).
- **rejected**: The Deferred object is in the rejected state, meaning that the failCallbacks have been called (or are in the process of being called).

UI Widgets

Overview

The Sample UI is based on independent and easily configurable components. Each component is a **View** in term of Backbone.js and may depend on Knowledge Agent and other 3rd party libraries. All the dependencies are managed by RequireJS in AMD-style.

When using widgets, the path to their file must be written using the **.define** function, which exports a constructor function to the current context. The last step is to create object **(new Constructor(options))** based on this constructor and call **.render()** method. Some widgets (in particular : Categories, Result and Document widgets) fetch data from the server and, in this case, **.fire()** method must be called before **.render()** method. Furthermore, each widget has public object settings, with stored widget configuration, based on constructor arguments.

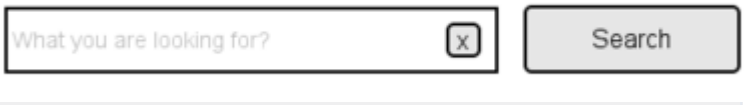
You can find a Basic API on the Backbone documentation page.

Components

Search Widget

The Search widget displays the standard Search bar and has a custom placeholder and optional Clear button.

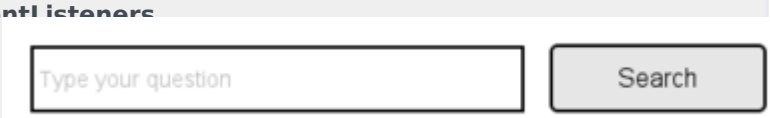
SearchWidget([options])	**Example**

Description: constructor for search widget.
• **el** (default: '#_gk-_wd-_sr')
Type: String
Selector for DOM element in which the current widget will be inserted
• **placeholder** (default: *)*
Type: String
Placeholder for search input
• **buttonText** (default: 'Search')
	
	Placeholder Example 1
	new SearchWidget({
	placeholder: 'What are you looking for?',
	showClearButton: true,
	searchButtonClickEventListeners: [function (element, options) {
	console.log('Search button clicked')
	}]
	}).render();

SearchWidget([options])	Example

Type: String
The text in search button

- **showClearButton** (default: true)
  Type: Boolean
  Whether to show or not Clear button
- **query** (default: *)
  Type: String
  Typed text for search input
- **categories** (deafult: [])
  Type: String[]
  Context categories for autocomplete
- **searchButtonClickEventListeners**
  Type: Function(Element element, PlainObject options)[]
  Functions to be called on Search button click
  - **element**
    Type: Element
    An element in the Document Object Model (DOM)
  - **options**
    Type: PlainObject
    A set of key/value pairs that contains data for listeners.
    - **query**
      Type: String
      User typed query string.
- **clearButtonClickEventListeners**
  Type: Function(Element element)[]
  Functions to be called on Clear button click
  - **element**



Placeholder Example 2

```
new SearchWidget({
   placeholder: 'Type your question',
   showClearButton: false
}).render();
```

SearchWidget([options])	Example
Type: Element An element in the Document Object Model (DOM)	
.render(): SearchWidget	
Description: renders the view template and updates this.el with the new HTML.	

Result Widget

The Result widget displays the search results and has optional pagination and optional embedded blocks of categories. this widget is dependent on the **_gka.search(**) method in Knowledge Agent.

ResultWidget([options])

Description: constructor for results widget.

- **el** (default: '#_gk-_wd-_rs')
  Type: String
  Selector for DOM element in which the current widget will be inserted
- **resultType** (default: 'SEARCH')
  Type: String enum ('SEARCH', 'TOP', 'BROWSE')
  Base type of the widget
- **size** (default: 10)
  Type: Number
  Number of documents in result list
- **showAnswer** (default: true)
  Type: Boolean
  Whether to show or not answers in document
- **charactersInAnswer** (default: 250)
  Type: Number
  Number of character in answer before "more" link appears
- **pagination** (default: true)
  Type: Boolean
  Whether to show or not pagination panel. Works only with resultType='BROWSE'
- **highlighting** (default: true)
  Type: Boolean
  Whether to highlight or not typed blocks
- **filters**
  Type: PlainObject[]
  Array of custom filters for search.
  - **fieldId**
    Type: String
    Identifier of the field which need to be filtered (segment equal "premium")
  - **operation**
    Type: String enum ("equal", "gt", "lt", "range", "regexp", "enum")

ResultWidget([options])

Expression operator (segment equal "premium")

- **value**
  Type: not defined
  Value for expression (segment equal "premium")

- **moreLinkText** (default: 'more')
  Type: String
  Allows to override default text in "more" link
- **moreLinkClickListeners**
  Type: Function(Element element, PlainObject options)[]
  Functions to be called on More link click
  - **element**
    Type: Element
    An element in the Document Object Model (DOM)
  - **options**
    Type: PlainObject
    A set of key/value pairs that contains data for listeners.
    - **id**
      Type: Number
      Document identifier
      question
      Type: String
      Question in document
      answer
      Type: String
      Answer in document

.fire(): Promise

Description: fetch data from the server according to passed options.

- **query** (default: *)*
  Type: String
  User typed query string
- **categories** (default: [])
  Type: String[]
  List of categories that is used as a context for the current query
- **filters** (default: [])
  Type: String[]
  Filters that will be passed. Works anly with resultType='SEARCH'

.render(): ResultWidget

Description: renders the view template from fetched data and updates this.el with the new HTML

Categories Widget

The Categories widget displays categories and is similar to the categories block in the Result widget

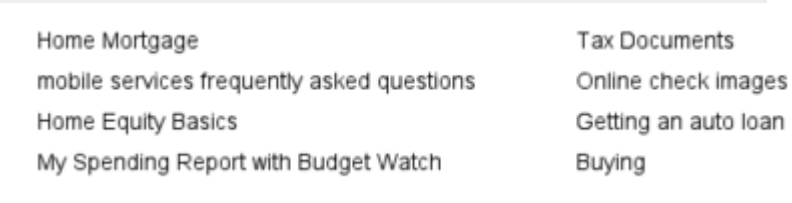
or the Document widget however, in the Categories widget, only the categories are stored. The Categories widget is dependant on the **_gka.getCategories()** method in Knowledge Agent.

CategoriesWidget([options])

Description:
constructor for categories widget.

- **el** (default: '#_gk-_wd-_cat')
  Type: String
  Selector for DOM element in which the current widget will be inserted
- **numberOfColumns** (default: 3)
  Type: Number
  Number of columns in panel categories
- **filteredCategories** (default: [])
  Type: String[]
  Array of id's for categories which should be rendered. Empty array means that all of fetched categories will be rendered.
- **categoryClickEventListeners**
  Type: Function(Element element)[]
  Functions to be called after particular category selected
  - **element**
    Type: Element
    An element in the Document Object Model (DOM)
  - **options**
    Type: PlainObject
    A set of key/value pairs that contains data for listeners.
    - **id**
      Type: Number
      Category identifier

Example



Categories Example

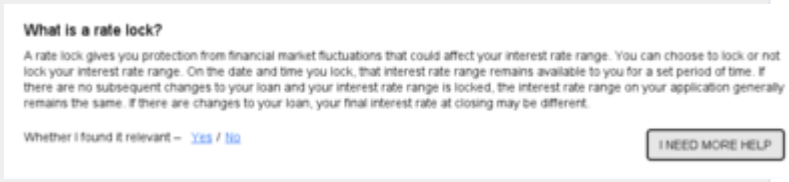
```
new CategoriesWidget({

   numberOfColumns: 2,
   categoryClickEventListeners: [function (element,
options) {
       console.log(options.id)
   }]

}).fire().done(function(response, widget) {

   widget.render()

});
```

CategoriesWidget([options])	Example
• **name** Type: String Category name	
.fire(): Promise	
Description: fetch data from the server.	
.render(): CategoriesWidget	
Description: renders the view template from fetched data and updates this.el with the new HTML.	

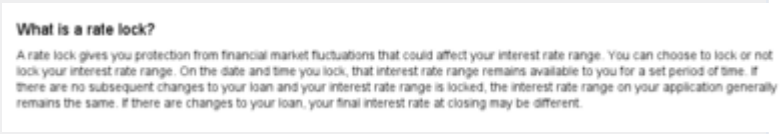
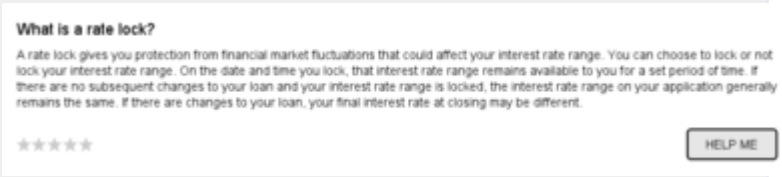
Document Widget

The Document widget displays documents and can have an optional feedback block, a help button, and an embedded block of categories that are associated to the document. The Document widget is dependant on **_gka.getFullContent()** and **_gka.like()** methods in Knowledge Agent.

DocumentWidget([options])	Example

Description: constructor for document widget.
• **el** (default: '#_gk-_wd-_doc')
Type: String
Selector for DOM element in which the current widget will be inserted
• **feedbackType** (default: 'BINARY')
Type: String enum ('NONE', "BINARY")
Style of rendering the feedback block
• **feedbackText** (default: *Whether I found it relevant')*
Type: String
Text in feedback block
• **feedbackAnswer** (default: *Thank you for your answer!*)
Type: String
Text that appears after successful feedback action
	
	Document Example 1
	
	new DocumentWidget({
	`documentId: 'knowledgefaq_4',`
	`feedbackType: 'BINARY',`
	`feedbackText: 'Whether I found it relevant',`
	`showHelpButton: true,`
	`helpButtonText: 'I need more help',`
	`feedbackSelectEventListeners: [function (element, options) {`
	`console.log(options.feedbackType);`
	`console.log(options.rate);`
	`}],`
	`helpButtonClickEventListeners: [function (element, options) {`
	`console.log(options.document.id);`
	`}]`
	}).fire({
	`kbId: 'knowledgefaq',`
	`docId: 'knowledgefaq_66'`
	}).done(function(response, widget) {
	`widget.render()`

DocumentWidget([options])	Example

• **showHelpButton** (default: true) Type: Boolean Whether to show or not help button
• **helpButtonText** (default: 'I need more help') Type: String Text in help button
• **feedbackClickListeners** Type: Function(Element element, options)[] Functions to be called after feedback selected
  • **element** Type: Element An element in the Document Object Model (DOM)
  • **options** Type: PlainObject A set of key/value pairs that contains data for listeners.
  • **document** Type: PlainObject Current document
  • **rate** Type: Number Chosen rate
  • **feedbackType** Type: String Current feedbackType
• **helpButtonClickListeners** Type: Function(Element element, options)[] Functions to be called after help button clicked
  • **element**
	})
	 
	Document Example 2
	new DocumentWidget({
	documentId: 'knowledgefaq_4',
	feedbackType: 'NONE',
	showHelpButton: false
	}).fire({
	kbId: 'knowledgefaq',
	docId: 'knowledgefaq_66'
	}).done(function(response, widget) {
	widget.render()
	})
	 
	Document Example 3
	new DocumentWidget({
	documentId: 'knowledgefaq_4',
	feedbackType: 'RATING',
	feedbackText: ,
	showHelpButton: true,
	helpButtonText: 'Help me'
	}).fire({
	kbId: 'knowledgefaq',
	docId: 'knowledgefaq_66'
	}).done(function(response, widget) {
	widget.render()
	})

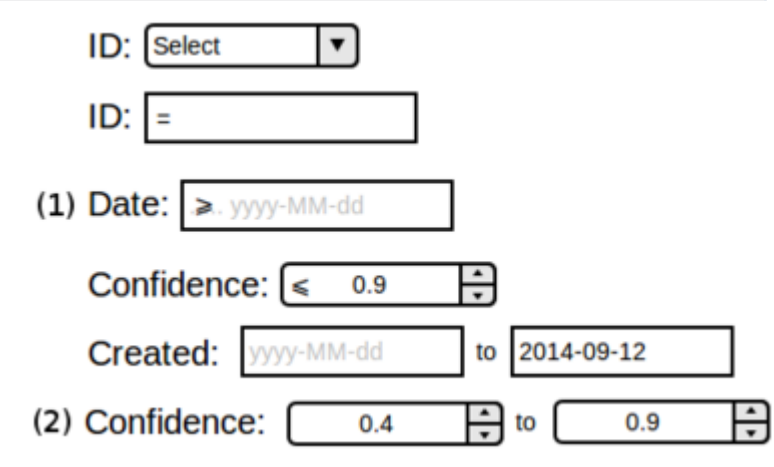
DocumentWidget([options])	Example
Type: Element An element in the Document Object Model (DOM) • **options** Type: PlainObject A set of key/value pairs that contains data for listeners. • **document** Type: PlainObject Current document	
.fire(options): Promise	
Description: fetch data from the server according to passed options. • **kbId** Type: String Knowledge base identifier • **docId** Type: String Document identifier	
.render(): DocumentWidget	
Description: renders the view template from fetched data and updates this.el with the new HTML.	

Filter Widget

The filter widget displays one single filter item depending on the passed configuration options and can be configured for visualization (timepicker, string input, number input).

FilterWidget(options)	Example
Description: constructor for filter widget. • **el** (default: '#_gk-_wd-_fl') Type: String Selector for DOM element in which the current widget will be inserted	

FilterWidget(options)	Example

• **type** (default: 'INPUT')
Type: String enum ('INPUT', 'DROPDOWN', 'BETWEEN')
Basic filter type
• **field** (default: 'id')
Type: String
Document field on which the result will be filtered
• **fieldAlias** (default: field)
Type: String
The text that will be shown near the input(s)
• **options**
Type: PlainObject
A set of key/value pairs which contain additional configuration of the widget
Widgets with different type expect different set
• **valueChangeEventListeners** (default: [])
Type: Function(Element element, options)[]
Functions to be called after value changed
  • **element**
Type: Element
An element in the Document Object Model (DOM)
  • **options**
Type: PlainObject
A set of key/value pairs that contains data for listeners
	
	Filter Widgets Example
	
	// Example (1) var filterWidget = new FilterWidget({
	  type: 'INPUT',
	  field: 'created',
	  fieldAlias: 'Date',
	  options: {
	  inputType: 'DATE',
	  operator: 'GREATER_EQUAL',
	  },
	  valueChangeEventListeners: [function (element) {
	  console.log('date changed')
	  }]
	})
	//Example (2) var filter2 = new FilterWidget({
	  type: 'BETWEEN',
	  field: 'confidence',
	  fieldAlias: 'Confidence',
	  options: {
	  separator: 'to',
	  inputType: 'NUMERIC',
	  from: 0.4,
	  to: 0.9
	  },
	  valueChangeEventListeners: [function (element, options) {
	  console.log('one of two values changed')
	  }]
	})

FilterWidget(options)	Example
"INPUT" type options (default) Rendering based on HTML tag: <input type="*text* | *number*"> • **inputType** (default: 'STRING') Type: String enum ('STRING', 'NUMERIC', 'DATE') Determines which basic type wiil be used for <input> • **operator** (default: 'EQUAL') Type: String enum ('LESS', 'LESS_EQUAL', 'GREATER', 'GREATER_EQUAL', 'EQUAL') • **value** (default: *)* Type: String	
"BETWEEN" type options Rendering based on HTML tag: <select> • **header** (default: 'Select') Type: String First, default, empty-behaviour <option> • **values** (default: []) Type: String[] Other, non-empty <options>'s • **value** (default: *)* Type: String Current value. If value in an instance of values, particular	

FilterWidget(options)	Example
"BETWEEN" type options <option> obtains selected attribute.	
"DROPDOWN" type options Rendering based on two of the following HTML tags: <input type="*text* | *number*"> • **inputType** (default: 'DATE') Type: String enum ('NUMERIC', 'DATE') Determines which basic type wiil be used for <input> • **separator** (deafult: 'to') Type: String The text separator between two <input>'s • **from** (default: *)* Type: String | Number First value • **to** (default: *)* Type: String | Number Second value	
.render(): FilterWidget	
Description: renders the view template and updates this.el with the new HTML.	
.filterJSON(): Filter	
Description: returns compiled filter object related to current widget that can be used in /search API.	

CSS Customization

All widgets have some basic CSS defined and built-in, however Styles can be managed though the browser debugger or easily overridden in a custom CSS file. HTML tags separation is based on classes and all classes used in the Sample UI are divided into different namespaces. For example, widget-classes have a **_gks-_wg-** prefix. Non-widget classes only use the **_gks-** prefix.

Filters

The Result widget supports extensions with custom filters and has a filters property, which is an array of standard objects.

Default filters are configurable along with other Knowledge Base information however only one default filter can be configured per language. Filters can be based on the basic fields of the knowledge article and custom fields (properly defined custom fields only).

All filter criteria is applied using AND logic (for example, createddate > 20140101 00:00:00 AND segment = "premium").

> ### Important
> The Result widget only supports the standard syntax and does not know which filters are enabled in the Knowledge Base.