Genesys Knowledge Center Developer's Guide

UI Widgets

12/14/2025

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

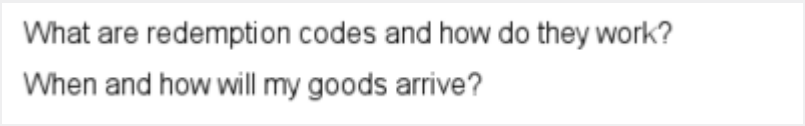
The Search widget displays the standard Search bar and has a custom placeholder and optional Clear button.

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') 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** (defult: []) 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**	 Search Widget Interface `new SearchWidget({ placeholder: 'What are you looking for?', showClearButton: true, searchButtonClickEventListeners: [function (element, options) { console.log('Search button clicked') }] }).render();`  Search Widget Interface `new SearchWidget({ placeholder: 'Type your question', showClearButton: false }).render();`

SearchWidget([options])	Example
Type: String User typed query string. • **clearButtonClickEventListeners** Type: Function(Element element)[] Functions to be called on Clear button click • **element** 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.

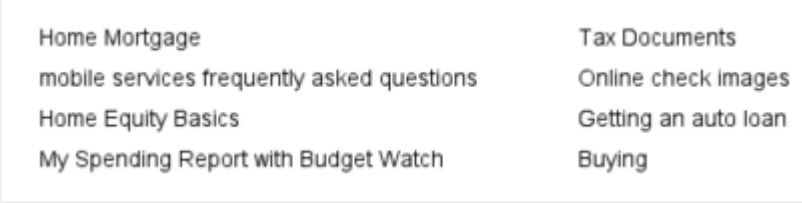
ResultWidget([options])	Example
Description: constructor for results widget. • **el** (default: '#_gk-_wd-_rs') Type: String Selector for DOM element in which the current widget will be inserted • **resultType** (default: 'SEARCH') Type: String enum ('SEARCH', 'TOP', 'BROWSE') Base type of the widget • **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' • **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")	 Result Widget Interface `new ResultWidget({` `  size: 2,` `  showAnswer: true,` `  charactersInAnswer: 250,` `  pagination: true,` `  highlighting: false,` `  moreLinkClickEventListeners: [function (element, options) {` `  console.log('Document selected')` `  }]` `}).fire().done(function(response, widget) {` `  widget.render()` `});`  Result Widget Interface

ResultWidget([options])	Example
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 • **showNoAnswerButton** Type: Boolean Whether the "No relevant results" button will be shown • **noAnswerButtonText** (default:"No relevant results") Type: String Localizable button label • **noAnswerClickedText'** (default: "Thank you for your feedback!") Type: String Localizable message after the "No relevant results" button has been clicked • **noMatchesText** (default: "No matches found.") Type: String Localizable message if search result contains zero documents • **documentClickURI** (default: function () { return 'javascript:;' }) Type: Function() URI after "more" button has been clicked • **documentClickListeners** 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)	`new ResultWidget({` `  size: 2,` `  showAnswer: false,` `  showCategories: false` `}).fire().done(function(response, widget) {` `  widget.render()` `});`

ResultWidget([options])	Example
• **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 • **noAnswerClickListeners** Type: Function(Element element)[] Functions to be called on "No relevant results" button click	
.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

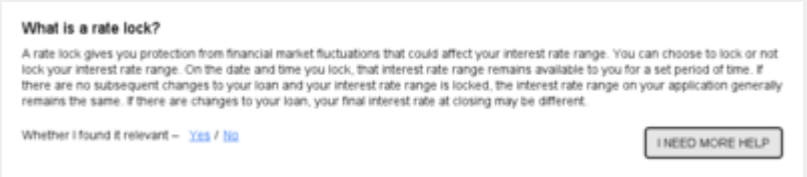
The Categories widget displays categories and is similar to the categories block in the Result widget or the Document widget however, in the Categories widget, only the categories are stored. The Categories widget is dependant on the **_gka.getCategories()** method in Knowledge Agent.

CategoriesWidget([options])	Example
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. • **categoryClickURI** (default: function () {return 'javascript:;'}) Type: Function() URI after category has been selected (clicked) • **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 • **name** Type: String Category name	 Categories Example `new CategoriesWidget({ numberOfColumns: 2, categoryClickEventListeners: [function (element, options) { console.log(options.id) }] }).fire().done(function(response, widget) { widget.render() });`

CategoriesWidget([options])	Example
• **categoriesNotFoundText** (default: "No linked categories found") Type: String Localizable message if no linked categories found	
.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 • **commentTrigger** (default: 'negative') Type: String enum ('negative', 'positive', both) When comment block should appear (for example when 'negative', the comment block will be shown just after a negative vote) • **feedback** Type: PlainObject • **modified** (default: 'Modified') Type: String Localizable message for modification date tooltip • **views** (default: 'Views') Type: String Localizable message for views number tooltip • **rating** (default: 'Rating') Type: String Localizable message for rating tooltip • **question** (default: *Was this helpful?*) Type: String Localizable message • **defaultAnswer** (default: Thank you for your vote.') Type: String Localizable message • **noCommentAnswer** (default: 'Thank you for your vote.')	 Document Widget 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()` `})`  Document Widget Example 2

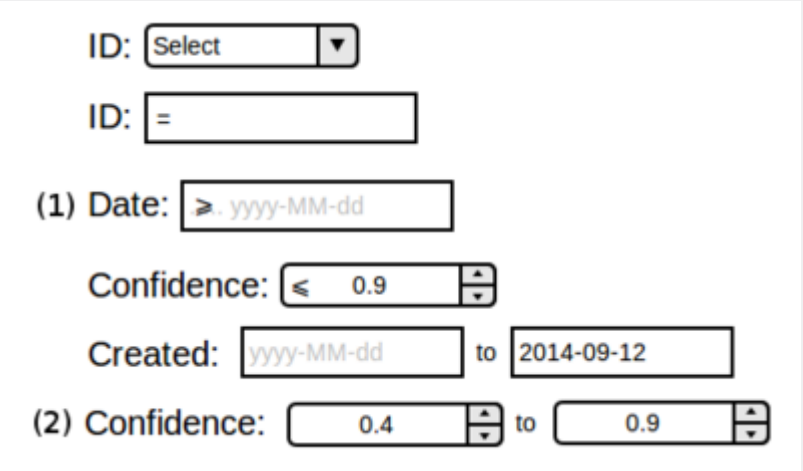
DocumentWidget([options])	Example
Type: String Localizable message • **submitAnswer** (default: 'Thanks, your feedback has been submitted.') Type: String Localizable message • **commentPlaceholder** (default: "Why wasn't this helpful?') Type: String Localizable message • **buttons** Type: PlainObject • **yes** (default: 'Yes') Type: String Localizable message • **no** (default 'No') Type: String Localizable message • **submit** (default: 'Submit') Type: String Localizable message • **noComment** (default: 'No Comment') Type: String Localizable message • **showHelpButton** (default: true) Type: Boolean Whether or not to display help button • **helpButtonText** (default: 'I need more help.') Type: String Text in help button	new DocumentWidget({ documentId: 'knowledgefaq_4', feedbackType: 'NONE', showHelpButton: false }).fire({ kbId: 'knowledgefaq', docId: 'knowledgefaq_66' }).done(function(response, widget) { widget.render() }) [[File:Document3.png|thumb|center|400px| Document Widget 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
• **showHelpAttachments** (default: true) Type: Boolean Whether or not to display document attachments • **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** Type: Element An element in the Document Object Model (DOM) • **options** Type: PlainObject	

DocumentWidget([options])	Example
A set of key/value pairs that contains data for listeners. • **document** Type: PlainObject Current document • **localLinkClickEventListeners** Type: Function(Element element)[] Functions to be called after help local link in the document clicked • **element** Type: Element An element in the Document Object Model (DOM)	
.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 • **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 Widget 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({ ```

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

"DROPDOWN" 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 <option> obtains selected attribute.

```
    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
"BETWEEN" 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.