Front-end Implementation of a Code Widget

Implement JavaScript

JavaScript Entry Point

This is the JavaScript file that is defined by the *AbstractCodeTemplate.getJavascriptMainPath().* This is the first JavaScript file that will be loaded by Yellowfin when the code widget is loaded onto a piece of content, and Yellowfin will pass a number of options to it.

Java

Define the my_widget.js file and return it as the *mainJavascriptPath()*

```
setupResources() {
addResource(“my_widget.js”, “text/javascript”);
}

getMainJavascriptPath() {
        return “my_widget.js”;
}
```

JavaScript

```
my_widget.js
define(function() {

        class myWidget {
                constructor(options) {
                        //This will be called by YF when the code widget is rendered.
}
}

return myWidget;
});
```

If there are dependencies, you can define them in an array, which will then be passed into the defined function. A file’s position in the dependency array will match its position as a passed argument.

```
my_widget.js
define([‘my_example_dependency.js’], function(MyExampleDependencyObject) {
        class myWidget {
                constructor(options) {
                        //This will be called by YF when the code widget is rendered.
}
}

return myWidget;
});
```

Options

The options object passed into the constructor will contain:

- APIs
- Element
- ResourceLoader
- Messenger

Item	Description	JavaScript example
API	An object that will contain the following: • DashboardAPI • WidgetAPI • CanvasAPI • Filters	*constructor(options) {* *options.apis.dashboards.name;* *options.apis.filters;* *}*
Element	This is the DOM element that Yellowfin has generated for the code widget to be rendered into. Any HTML you generate should be appended to this element.	*constructor(options) {* *options.element.innerHTML = 'Hello World';* *}*
ResourceLoader	This is a helper class that can be used to load extra resources from the code widget definition. This allows you to load other items from your code widget definition or any third-party libraries you may need after the initial code widget load. Note: Any resources you define in AbstractCodeTemplate.setupResources that aren't defined as isLibrary will be loaded when the CodeWidget is initialised. This is done to reduce the number of requests that are made when loading a code widget.	*options.resourceLoader.load ('my_second_file.js', function (MySecondFile) {* *//Do something with MySecondFile* *));*
Messenger	This is an object that contains some helpful functions and flags to get the status of the canvas and save options. For example: *edit* A flag to define if the canvas that this code widget is included on, is in edit mode. This can be used to create a custom interface for your widget. *getOptionValue(optionName)* Returns a value that has been defined for this widget. *setOptionValue(optionName, optionValue)* Saves the passed *optionValue* against the passed *optionName*, which can be retrieved later. This can be used for custom setup. If this is called while the canvas is not in edit mode, the call will be ignored.	***edit*** *if(options.messenger.edit) {* *//Custom Edit Code* *} else {* *//Published Code* *}* ***getOptionValue(optionName)*** *console.log(options.messenger.getOptionValue('myOption'));* ***setOptionValue(optionName, optionValue)*** *options.messenger.setOptionValue ('myOption', 'myOptionValue');* *console.log(options.messenger.getOptionValue('myOption')); //Value will be 'myOptionValue'*

Other JS files

Any JS file loaded by the code widget can also load multiple JavaScript files as dependencies for itself. This can be used to include third-party JS libraries, and for creating logically-organized modules of your own code that the code widget can use.

For example, if you've written a filter list widget which allows a user to filter by free text, an image or radio buttons, you could store the logic for each of these elements in their own separate files. The JS entry point can then determine which to use and load the relevant file.

Yellowfin will only automatically call the main JS file, so if you have more than one, remember to explicitly call them.

Java

```
addResource("my_second_file.js", "text/javascript");
```

JavaScript

```
my_widget.js
define([‘my_second_file.js’], function(SecondFile) {

});

my_second_file.js
define(function() {
        return class SecondFile {
                //My Class
        }
});
```


To avoid conflicts with Yellowfin and other code widgets, we recommend that you create a class or object to be returned from these files. However, if required, they can simply be util files that attach functionality to the window without returning anything. Using them in this way should be approached with caution and thoroughly tested for conflicts. If you wish to use those files in this manner you should set the *isLibrary* fl ag to true when registering the resource in the setupResources function.

Third-party JS libraries

If your code widget requires a third-party library, it can also be bundled within your code widget. Use the isLibrary flag when registering the resource in the AbstractCodeTemplate implementation:

Java

```
addResource(new Resource(“jquery.js”, “text/javascript”, true));
```

JavaScript

```
define([‘jquery.js’], function($) {

});
```

Implement HTML

You can include HTML in your code widgets. This will be delivered to the JavaScript function as a string.

Java

```
addResource(“my_html.html”, “text/html”);
```

JavaScript

```
define([‘my_html.html’], function(MyHTML) {
        console.log(MyHTML); //HTML String
});
```

Implement CSS

You can include CSS in your code widgets. This will be appended to the page the first time a code widget is rendered.

Java

```
addResource(“my_css.css”, “text/css”);
```

JavaScript

```
define([‘resource/my_css.css’], function(css) {
        //The CSS variable will be undefined, however you should still include it in your function header to ensure other inclusions aren’t ignored
}
```

Next step

Now that you've written the back-end code and the front-end code, you can optionally customize the Code Widget Properties panel. If you don't need to customize the panel, you can jump straight to bundling the code into a jar file, then upload it to Yellowfin.

To further assist you during code widget development, click on the code widget reference page for samples, API links and detailed descriptions.

Front-end Implementation of a Code Widget#top