Do-It-Yourself Geo Apps MOOC

Guided Exercise

Use JavaScript to Customize a Web App

Section 7 Exercise 1

08/2017

esri

THE SCIENCE OF WHERE™

Use JavaScript to Customize a Web App

Time to complete

Approximately 40-45 minutes.

Introduction

What is an API? API is short for Application Programming Interface and is a term that can take many forms. The ArcGIS API for JavaScript is a body of code and documentation that you can use to embed many functions into your web apps, including web maps from ArcGIS Online and mapping tools that perform querying, analysis, and routing. Most of the configurable apps we see in other sections of this course rely heavily on the ArcGIS API for JavaScript.

What is JavaScript? JavaScript is a programming language that is used with HTML and CSS to create web apps. HTML defines the content of web pages, CSS their layout and style, and JavaScript their behavior (for example, what happens when you click on a particular area of a web page).

We won't get into a lot of detail about HTML, CSS, and JavaScript in this exercise, but there are loads of resources available for those of you who are new to web development. Look at the Resources section at the end of the exercise if you would like to explore these topics more.

To introduce you to custom app development and the ArcGIS API for JavaScript, we are going to look at a sample web app, and add some simple HTML and JavaScript to change the displayed elements and behavior of the web page. Let's begin!

Let's Get Started!

First, you will need to find a sample app to use as the basis for your exploration.

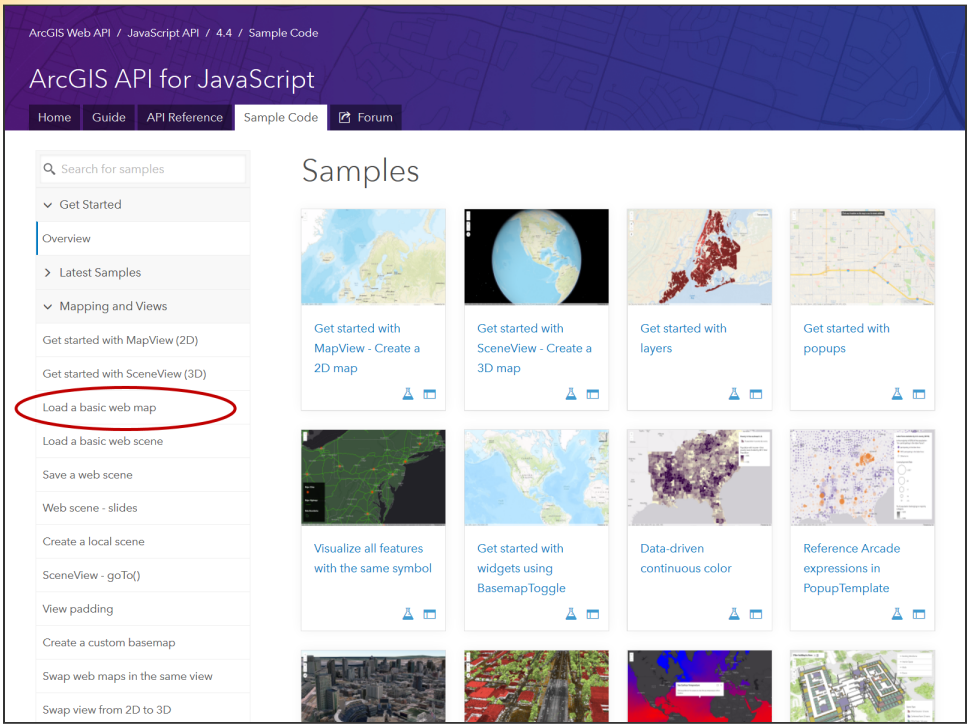
Step 1: Find a sample app

a. Open a new Internet browser tab or window and go to developers.arcgis.com.

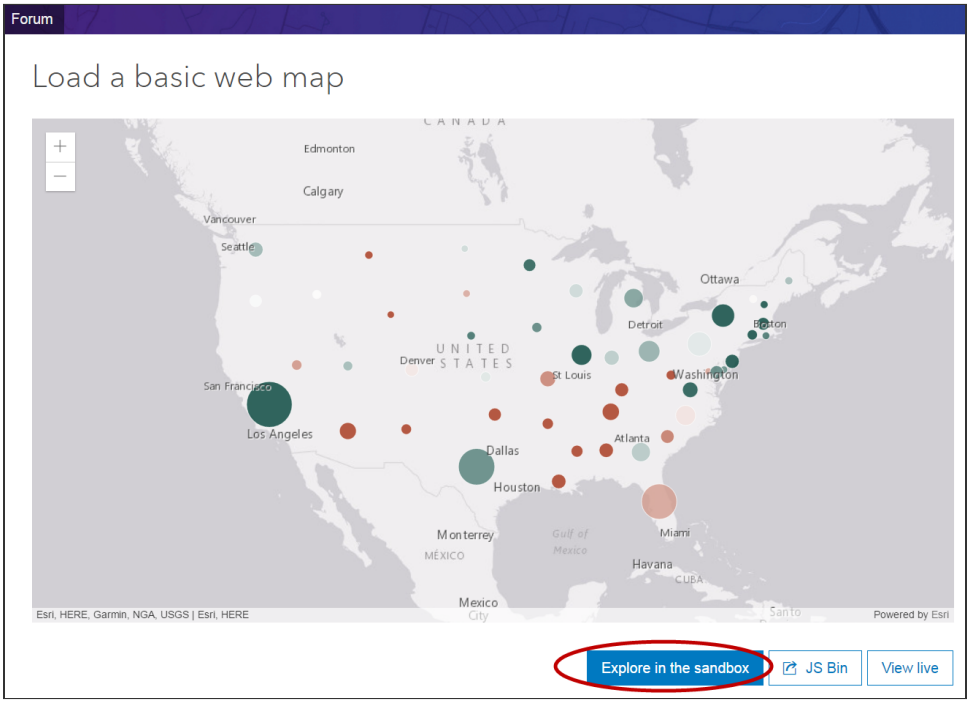
b. Below the landing image, click APIs & SDKs and, from the drop-down list, select JavaScript.

c. Click the Sample Code tab.

d. In the list of code samples organized by topic on the left side of the page, click Mapping and Views.



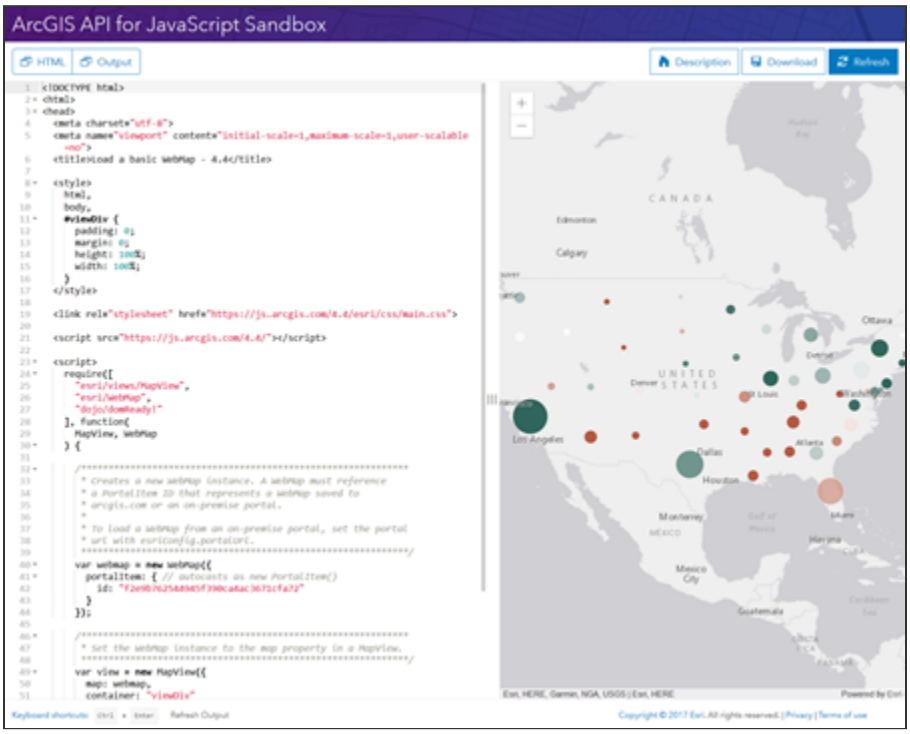
e Click Load a WebMap, and read the description below the map.



f Below the map, click Explore In The Sandbox.

You will see the HTML and JavaScript code in the editor pane on the left and the app preview on the right. After you have made edits to your code, you can click Refresh to refresh the preview.

The app previewed is pretty basic, it simply displays a web map.



This component was created in the <body> section of the web page.

Hint: Scroll down in the editor pane to view the <body> section.

```
<body>
  <div id="viewDiv"></div>
</body>
```

It's okay if you don't understand HTML. You just need to know the basics about tags. Tags are keywords surrounded by angle brackets, like this: <keyword>. They tell the browser what the content within them is and, in some cases (like with <body> and tags), how to display content. In this code, the tags and its attribute, id, is telling the browser to display the web map.

Step 2: Explore an HTML document

a At the top of the code document, you see the following components:

1. Metadata tags that help with web search.
2. The title of the web page, which is shown in the title frame of your app.
3. Internal CSS.
4. Links to external CSS stylesheets, or documents that have information used to style different elements in a web page. We are not going to cover CSS in depth, but if you are interested in learning more, you can start here.

```
<!DOCTYPE html>
<html>
<head>
1 <meta charset="utf-8">
  <meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no">
2 <title>Load a basic WebMap - 4.4</title>

3 <style>
    html,
    body,
    #viewDiv {
      padding: 0;
      margin: 0;
      height: 100%;
      width: 100%;
    }
  </style>

4 <link rel="stylesheet" href="https://js.arcgis.com/4.2/esri/css/main.css">

```

b Examine the <script> tags in the document.

```
  <script src="https://js.arcgis.com/4.4/"></script> 1

  <script>
    require([
      "esri/views/MapView",
2     "esri/WebMap",
      "dojo/domReady!"
    ], function(
      MapView, WebMap
    ) {

      /*************************************************************
       * Creates a new WebMap instance. A WebMap must reference
```

The first script tag in the graphic above is the API source code (the hosting location of the API), which you need to access the code in the ArcGIS API for JavaScript. The src attribute is the URL where this code is accessed.

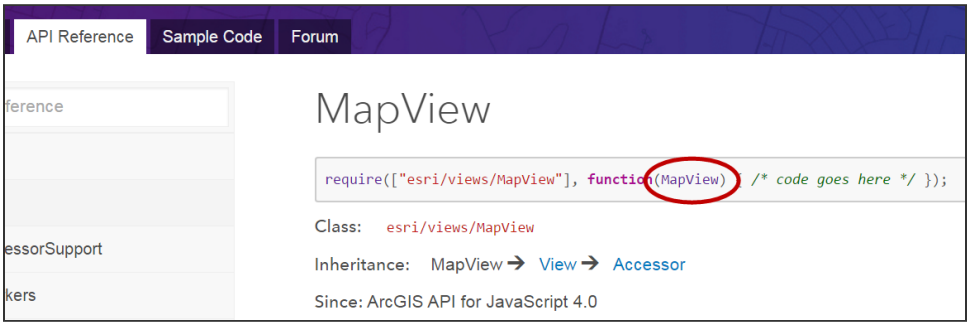
The second script tag lists specific modules individually. A module is just a bit of code that allows you to do a specific thing or set of things. For example, "esri/views/MapView" is a module that has code that you use to render web maps from ArcGIS Online in 2D. You can find a quick tutorial on how to use the MapView module here.

> *Hint: All the code you need to build the application is at the end of the tutorial. If you want to try the tutorial from start to finish, open up the sandbox with the finished code, delete the code in the editor pane, and follow the tutorial from Step 1. Click Refresh every time you want to preview the code you have written. Look to the complete code at the end of the tutorial for guidance if you get stuck.*

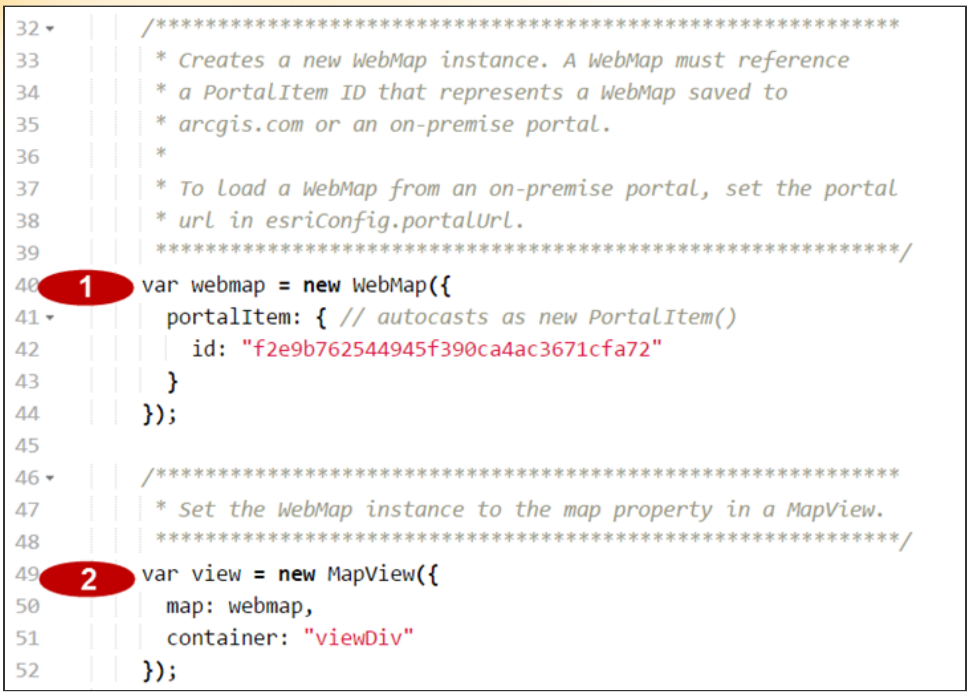
There is documentation on all of the modules in the API for JavaScript in the API Reference.

Most modules have an alias, or common name, that allows you to access the module quickly without typing in the full path. (For example, you can type "MapView" in your code rather than "esri/views/MapView".) These aliases are defined as arguments in the callback function that runs after all the source code from the JavaScript API has been downloaded. If you want a fuller explanation, check out this blog. This knowledge is not necessary to perform the exercise.

You can add any alias in the function, but it is a best practice to use the preferred aliases. You can find out the preferred alias for a module from the module's entry in the JavaScript API Reference. In the graphic below, the preferred alias for the MapView module is circled.



The code in the following graphic creates a WebMap instance from the ID of an existing web map in ArcGIS Online (1) and displays that web map in 2D (2). Does anything look familiar? The code uses the module aliases from the code above!




```
/*************************************************************
 * Creates a new WebMap instance. A WebMap must reference
 * a PortalItem ID that represents a WebMap saved to
 * arcgis.com or an on-premise portal.
 *
 * To load a WebMap from an on-premise portal, set the portal
 * url in esriConfig.portalUrl.
 *************************************************************/
var webmap = new WebMap({
  portalItem: { // autocasts as new PortalItem()
    id: "f2e9b762544945f390ca4ac3671cfa72"
  }
});

/*************************************************************
 * Set the WebMap instance to the map property in a MapView.
 *************************************************************/
var view = new MapView({
  map: webmap,
  container: "viewDiv"
});
```

Item 2 in the graphic above creates a variable, view, which uses the module MapView:

```
var view = new MapView({
  map: webmap,
  container: "viewDiv"
});
```

If "esri/views/MapView" and the alias MapView above had not been loaded, the map would not display!

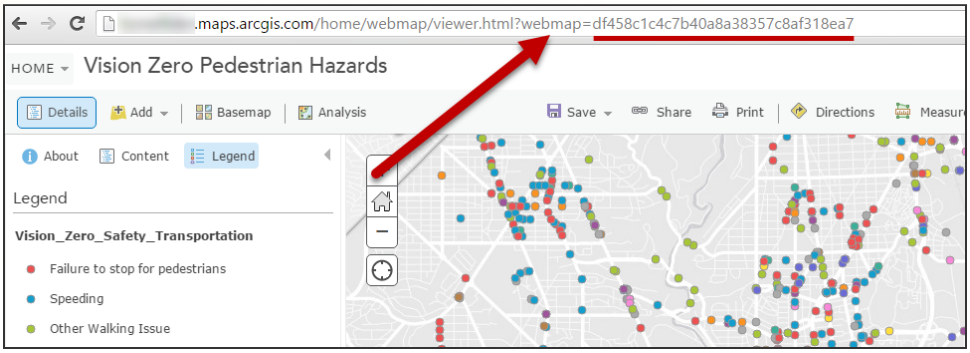
Again, you do not need to understand everything about the code. Just know that you need to load modules from the ArcGIS API for JavaScript in order to access that code and do things with it in your web app. If you want to know more, there are great HTML, CSS, and JavaScript tutorials out there, as well as specific tutorials for the ArcGIS API for JavaScript once you have mastered the basics.

Now, let's make some changes to the code to see some of the things it can do. First, you are going to change the web map in your app using the web map ID.

Step 3: Change the web map in your app

The web map ID is a unique set of characters stored in ArcGIS Online that is associated with a web map. When you are looking at a web map in the Map Viewer in ArcGIS Online, you can see the web map ID in the URL (as shown in the following graphic). To humans, this map is the

Vision Zero Pedestrian Hazards web map, but to computers, it is the df458c1c4c7b40a8a38357c8af318ea7 web map.



You are going to change the map from the Accidental Deaths map to the Vision Zero Pedestrian Hazards map using the web map ID.

a. In the editor pane, navigate to line 42.

Note: Depending on your line spacing, your code may appear on a different line. Look for the web map ID, a unique identifier that contains numbers and letters (as shown in the following graphic).

```
40    var webmap = new WebMap({
41      portalItem: { // autocasts as new PortalItem()
42        id: "f2e9b762544945f390ca4ac3671cfa72"
43      }
44    });
```

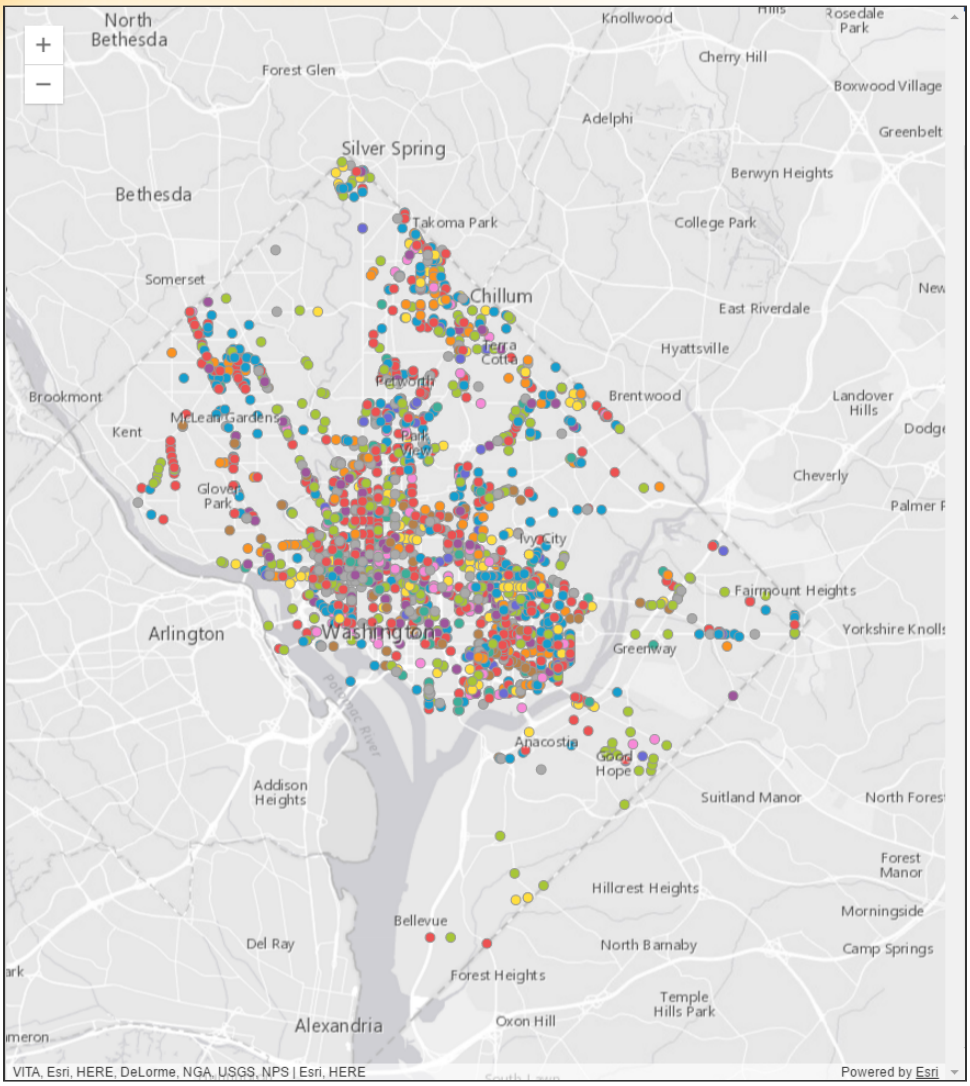
b. Delete f2e9b762544945f390ca4ac3671cfa72, and replace it with **df458c1c4c7b40a8a38357c8af318ea7**, as shown in the graphic below:

Note: Be sure to copy only the numbers and letters, not the comma.

```
40    var webmap = new WebMap({
41      portalItem: { // autocasts as new PortalItem()
42        id: "df458c1c4c7b40a8a38357c8af318ea7"
43      }
44    });
```

c. Click Refresh to see your preview update.

You have now added a new web map into your app. Your screen should resemble the following graphic:



Assigning a new web map ID for the WebMap instance has changed the web map that is being rendered in the app. Pretty cool, huh!?

Step 4: Explore the Search widget class

In this section, we are going to be talking a lot about classes and properties, which are often classes themselves. Remember those modules we loaded earlier? Each of those modules has a group of classes and subclasses within them. A class is a bit of code that returns something when you pass values to it.

We won't get into too much detail, but for illustrative purposes, let's look at the MapView code again.

```
46    /*************************************************************
47     * Set the WebMap instance to the map property in a MapView.
48     *************************************************************/
49    var view = new MapView({
50      map: webmap,
51      container: "viewDiv"
52    });
```

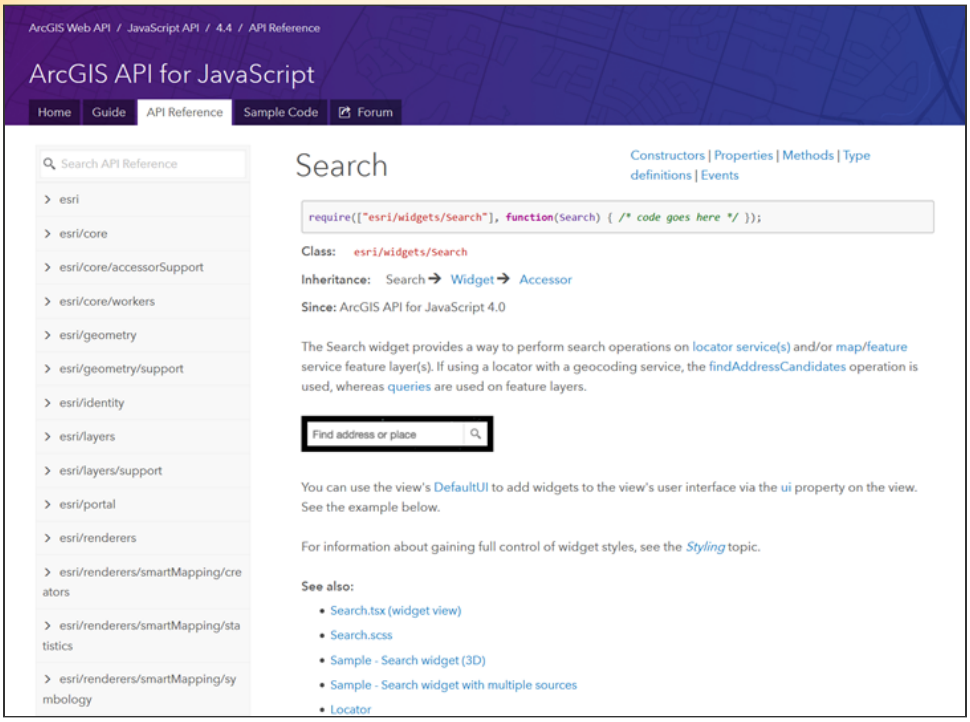
Here, we used the MapView class to display a web map in 2D. Here, a map property and a container property is passed to the view object.

Note: viewDiv is an ID selector that was defined earlier in the script that basically says the MapView will render in the entire browser space (height: 100%, width: 100%). View the graphic below to see how viewDiv is defined.

```
8   <style>
9     html,
10    body,
11    #viewDiv {
12      padding: 0;
13      margin: 0;
14      height: 100%;
15      width: 100%;
16    }
17  </style>
```

Let's take a look at another class.

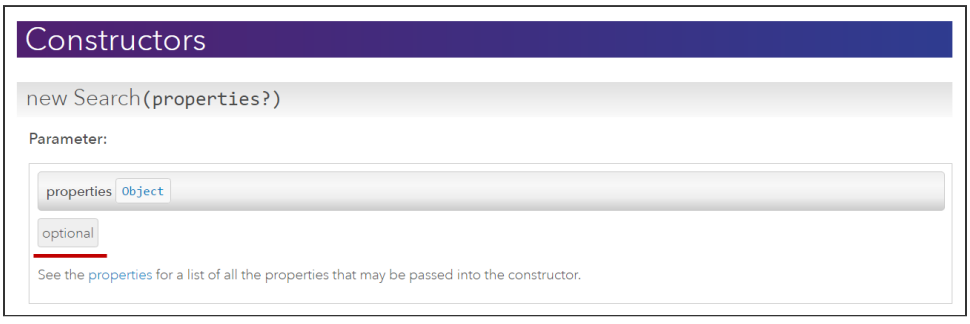
a. Open a new browser window or tab and look at the documentation for the Search widget in the API Reference.



The Search widget allows the user of our app to search for an address using a locator service like Esri's World Geocoding Service (the default) or for features using queries against a feature service (a slightly more advanced topic).

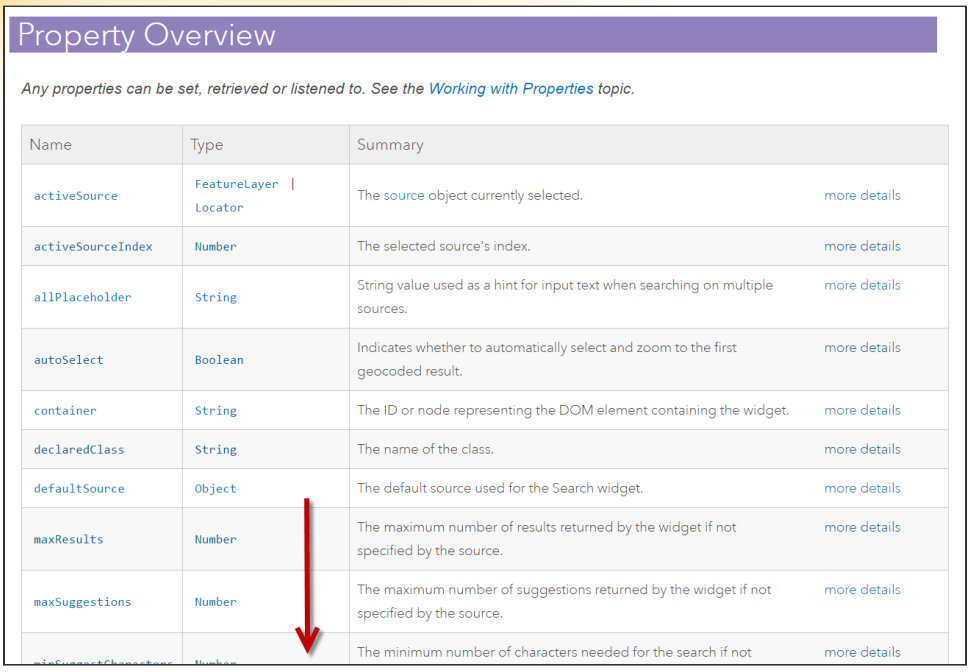
b. Scroll down to the Search class' Constructors section.

You will see that the class Search takes one parameter, the properties, and that it is optional.



Below the Constructors section for every class you will find the Property Overview, which lists all the properties that can be passed to the class referenced, and a Property Details section which provides more information and links to other resources for the properties listed in the Property Overview.

c. Scroll down through the Property Overview.



Property Overview

Any properties can be set, retrieved or listened to. See the Working with Properties topic.

Name	Type	Summary	
activeSource	FeatureLayer | Locator	The source object currently selected.	more details
activeSourceIndex	Number	The selected source's index.	more details
allPlaceholder	String	String value used as a hint for input text when searching on multiple sources.	more details
autoSelect	Boolean	Indicates whether to automatically select and zoom to the first geocoded result.	more details
container	String	The ID or node representing the DOM element containing the widget.	more details
declaredClass	String	The name of the class.	more details
defaultSource	Object	The default source used for the Search widget.	more details
maxResults	Number	The maximum number of results returned by the widget if not specified by the source.	more details
maxSuggestions	Number	The maximum number of suggestions returned by the widget if not specified by the source.	more details
minSuggestCharacters	Number	The minimum number of characters needed for the search if not	more details

You will see a long list of properties that can be passed to objects of the Search class, but rather than go through all of these, let's take a look at one from the Example code.

d Scroll up above the Constructors to the Example code.

Example:

```
var searchWidget = new Search({
  view: view
});
// Adds the search widget below other elements in
// the top left corner of the view
view.ui.add(searchWidget, {
  position: "top-left",
  index: 2
});
```

Constructors

Here, searchWidget is only being passed one property, view, which sets the Search widget to a specific view. If you want more information about the view property you can look it up in the Property Overview or Property Details section of Search's entry in the API Reference.

We don't need to dig any deeper into the API reference for this exercise. Just know that there is a TON of information there, and with a little patience and experimentation you can learn to create some pretty amazing custom geo apps. Documentation is a developer's best friend!

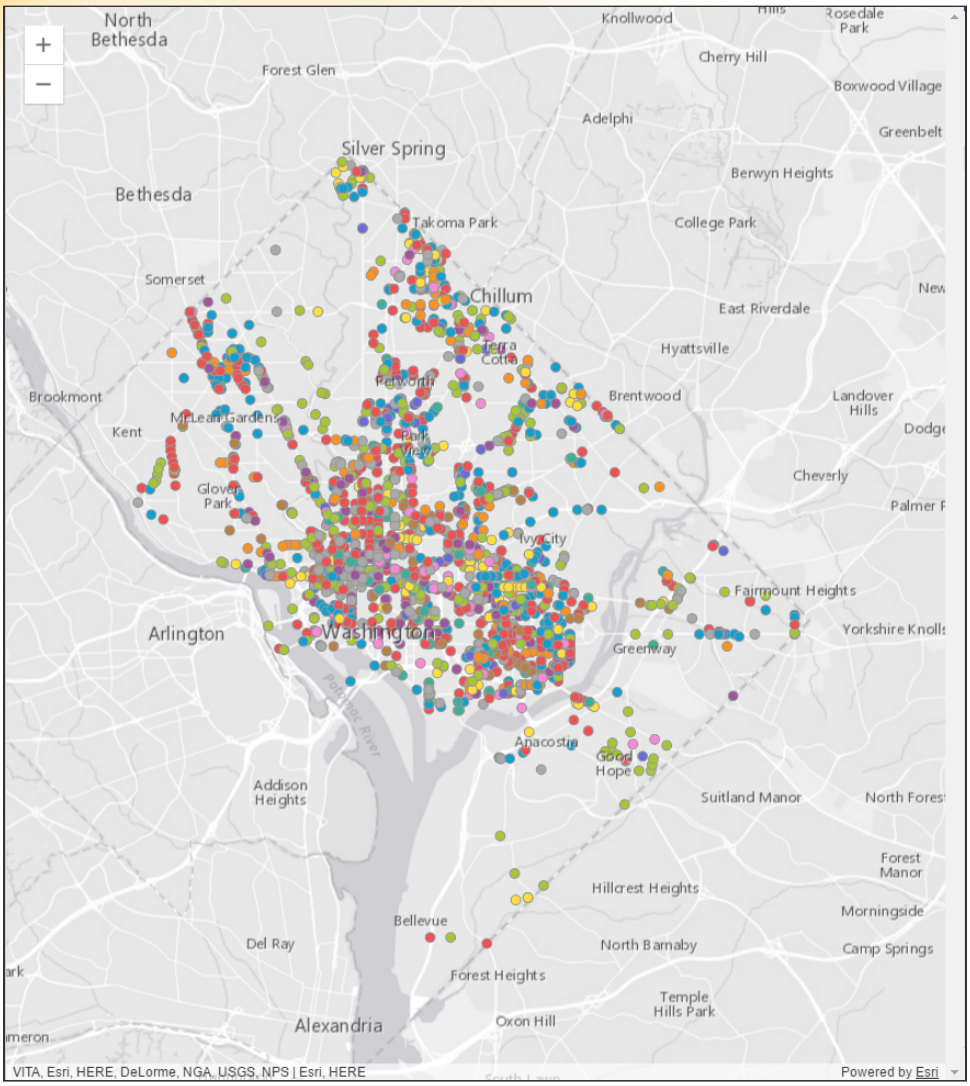
Let's add the Search widget to the Load a Basic WebMap sample app we were working with earlier using the example code from the API Reference.

Step 5: Add the Search widget to the Load a Basic WebMap sample

a. Copy all of the code from the Example.

b. In another tab, open the Load a Basic WebMap sample you were working on previously.

Note: Make sure you have switched out the web map ID's so that the Washington D.C. Vision Zero Pedestrian Hazards layer is displayed in the app preview, as shown in the graphic below.



c In the editor pane, click after the semi-colon on line 52 and, on your keyboard, press Enter (Return for Apple computers).

```
46      /*************************************************************
47       * Set the WebMap instance to the map property in a MapView.
48       *************************************************************/
49      var view = new MapView({
50        map: webmap,
51        container: "viewDiv"
52      });
53    });
54  </script>
55 </head>
```

d. Paste the code copied from the Example on to line 53 in the editor pane.

```
46      /*************************************************************
47       * Set the WebMap instance to the map property in a MapView.
48       *************************************************************/
49      var view = new MapView({
50        map: webmap,
51        container: "viewDiv"
52      });
53
54    });
55  </script>
56 </head>
```

Your code should look something like the graphic below:

```
46      /*************************************************************
47       * Set the WebMap instance to the map property in a MapView.
48       *************************************************************/
49      var view = new MapView({
50        map: webmap,
51        container: "viewDiv"
52      });
53      var searchWidget = new Search({
54    view: view
55  });
56  // Adds the search widget below other elements in
57  // the top left corner of the view
58  view.ui.add(searchWidget, {
59    position: "top-left",
60    index: 2
61  });
62    });
63  </script>
64 </head>
```

e. Highlight the code from lines 54 to 61, as shown in the graphic below:

```
46      /*************************************************************
47       * Set the WebMap instance to the map property in a MapView.
48       *************************************************************/
49      var view = new MapView({
50        map: webmap,
51        container: "viewDiv"
52      });
53      var searchWidget = new Search({
54    view: view
55  });
56  // Adds the search widget below other elements in
57  // the top left corner of the view
58  view.ui.add(searchWidget, {
59    position: "top-left",
60    index: 2
61  });
62      });
63    </script>
64  </head>
```

f On your keyboard, press Tab three times so that the code you pasted is in proper alignment with the rest of the code sample, as shown in the graphic below.

```
46      /*************************************************************
47       * Set the WebMap instance to the map property in a MapView.
48       *************************************************************/
49      var view = new MapView({
50        map: webmap,
51        container: "viewDiv"
52      });
53      var searchWidget = new Search({
54        view: view
55      });
56      // Adds the search widget below other elements in
57      // the top left corner of the view
58      view.ui.add(searchWidget, {
59        position: "top-left",
60        index: 2
61      });
```

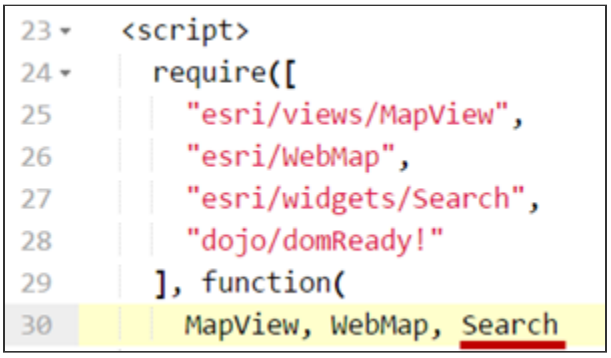
At this point you have created an object of the class Search, passed a view property which is set to the MapView object instantiated on line 49, and added the Search widget to the MapView in the top left corner on lines 58 and 59. In order for the Search widget to be added though, you need to load the Search module.

g. In the editor pane, click just to the right of the comma on line 26 and, on your keyboard, press Enter (Return on Apple computers).

h. On line 27, type **"esri/widgets/Search",** as shown in the graphic below. Remember to add a comma at the end.

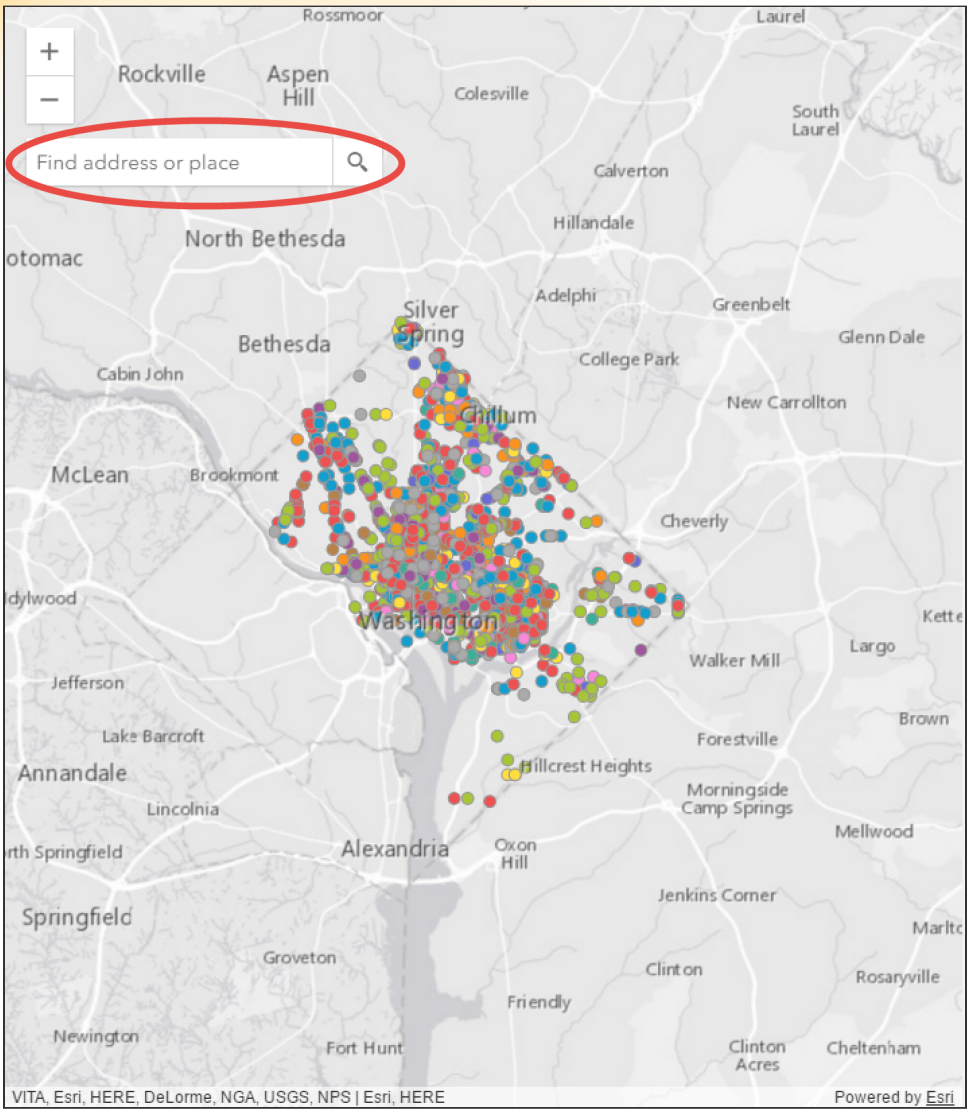
Note: Type the entry as indicated above, instead of copying and pasting.

i. On line 30, add a comma after WebMap and type **Search**, as shown in the graphic below.



j. Click Refresh.

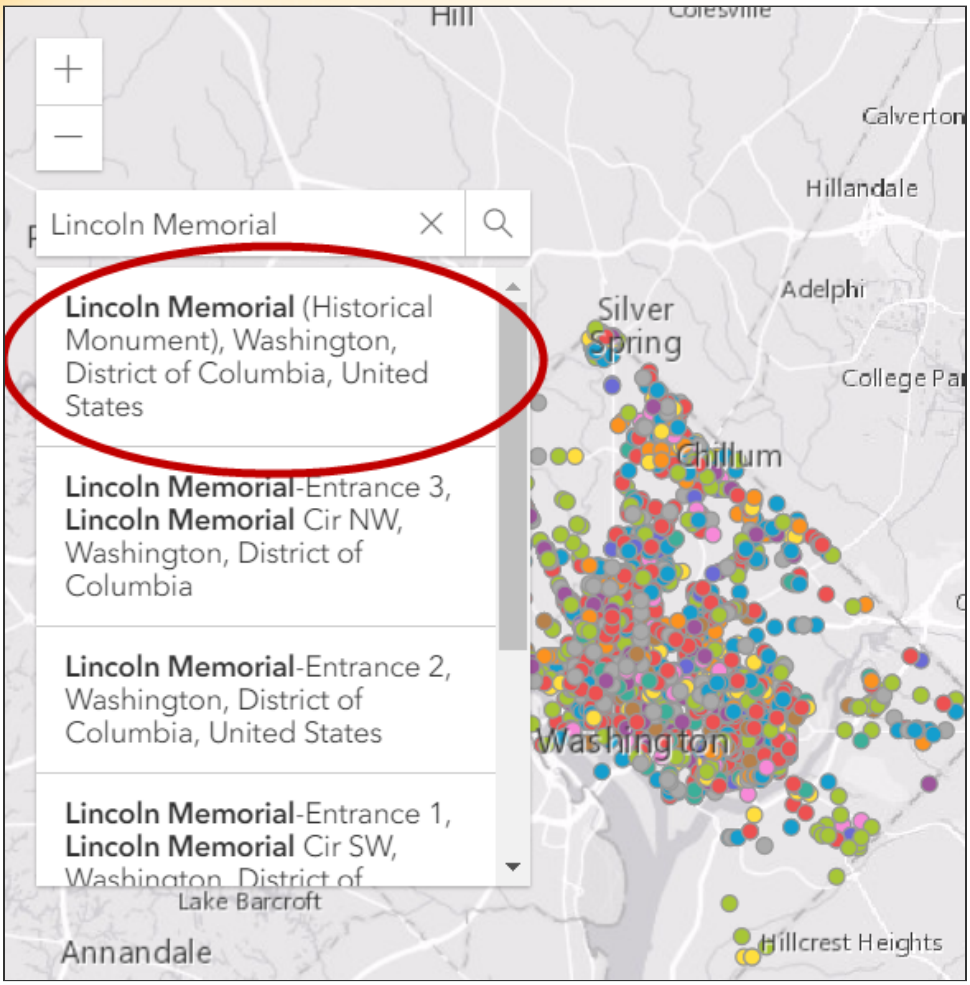
In the preview window, you should see the Search widget appear in the top-left corner of the app, as shown in the graphic below.



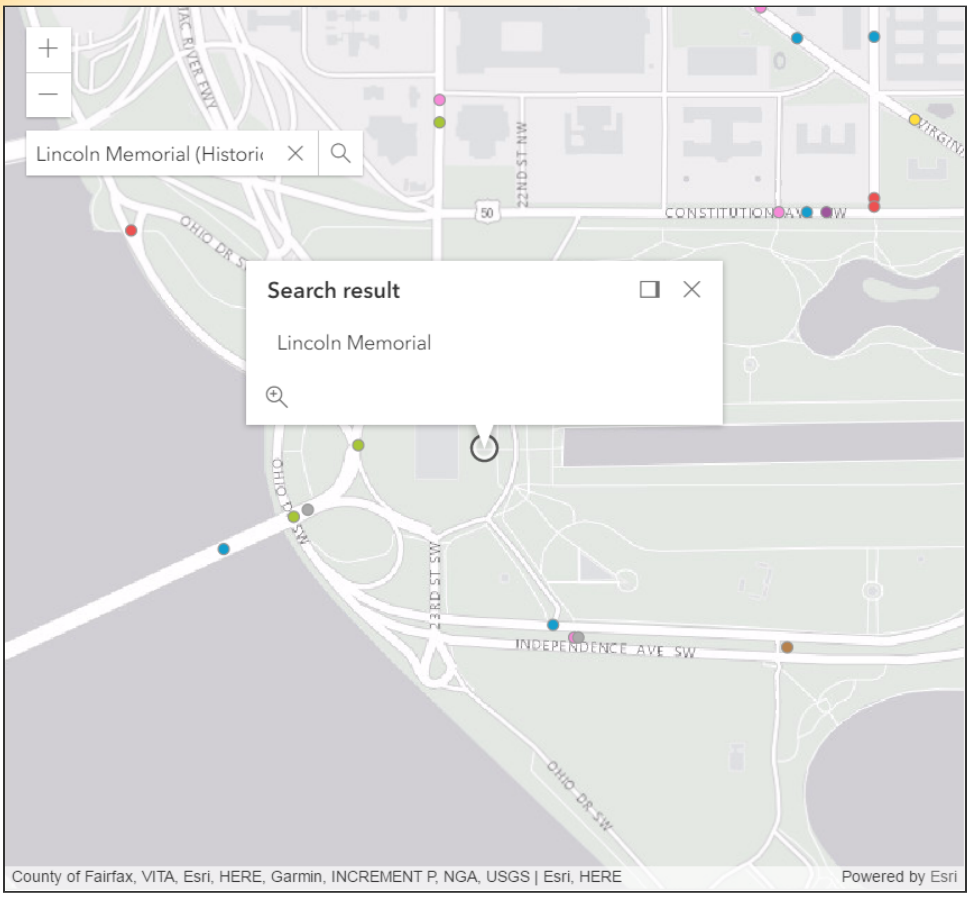
Let's test the Search widget out.

k. Copy and paste (or type) the text **Lincoln Memorial** in the Search widget, and then click the magnifying glass icon.

l. From the results drop-down list, choose the first result.



Your final result should look like the following graphic:



Conclusion

Congratulations! You just customized a web app while learning a thing or two about the API Reference along the way. We are the first to admit that developing is not always intuitive, but it is not magic either. There are many resources out there. With a little time and dedication, you can create powerful geo apps, customized to your tastes and needs!