

Chapter 4: Working with Maps and Layers

Now that you have a taste of ArcGIS Server and the API for JavaScript it's time to actually get to work and learn how to build some great GIS web applications! The material in this chapter will introduce you to some of the fundamental concepts that define how you create a map and add information to the map in the form of layers.

We'll begin by exploring the basic steps that need to be performed for each application that uses the ArcGIS Server API for JavaScript. Every application that you create with the API for JavaScript will need to follow each of these steps. The creation of a map and adding layers to the map is part of this fundamental process of creating an application. You will learn how to create an instance of the Map class, add layers of data to the map, and display this information on a web page. When creating a Map you can supply various parameters that control the initial extent of the display, the spatial reference, levels of detail, and others. You'll be introduced to the various parameters that can be used when the map is initially created. Map is the most fundamental class in the API as it provides the canvas for your data layers and any subsequent activities that occur in your application. However, your map is useless until you add layers of data. There are several types of data layers that can be added to a map including tiled, dynamic, feature, and others. You'll learn more about these layers in this chapter.

Basic Steps for Creating an Application with the ArcGIS Server API for JavaScript

There are four steps that you will need to follow to add a map to your web application. These steps will always need to be performed if you intend to have a map as part of your application. I'll briefly highlight each of the steps and then we'll examine each in greater detail. In a nutshell the four steps are as follows:

1. Reference the ArcGIS Server API for JavaScript and CSS files
2. Create a container for the map
3. Create an initialization script that performs initial setup operations
4. Call `dojo.addOnLoad()`

That is a just a brief description of what needs to be done.

Step 1: Reference the ArcGIS Server API for JavaScript

The first thing you need to do in any application that you develop with the ArcGIS Server API for JavaScript is to add a reference to the API and associated Dojo style sheets.

The style sheet, as you learned in Chapter 3, is used primarily for the graphic elements of the application and any supporting Dojo widgets and is enclosed within a `<link>` tag as you can see in the code example below. Please note that you can place all the text you see in this code

example on a single line rather than dividing it into two lines. I've used two lines here to make it easier to read.

```
<link rel="stylesheet" type="text/css"
      href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.6/js/dojo/dijit/themes/tundra/tundra.css">
```

As you'll recall, the ArcGIS Server API for JavaScript is built directly on the Dojo JavaScript framework. Dojo comes with four pre-defined themes that control the look of user interface widgets that are added to your application. The four themes include claro, tundra, soria, and nihilo. In the code example above we are referencing the tundra theme.

```
<link rel="stylesheet" type="text/css"
      href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.6/js/dojo/dijit/themes/tundra/tundra.css">
```

Dojo.org provides a theme tester that you can use to get a feel for how each of the themes affect the display of Dojo widgets. The theme tester is located at:

<http://archive.dojotoolkit.org/nightly/dojotoolkit/dijit/themes/themeTester.html>

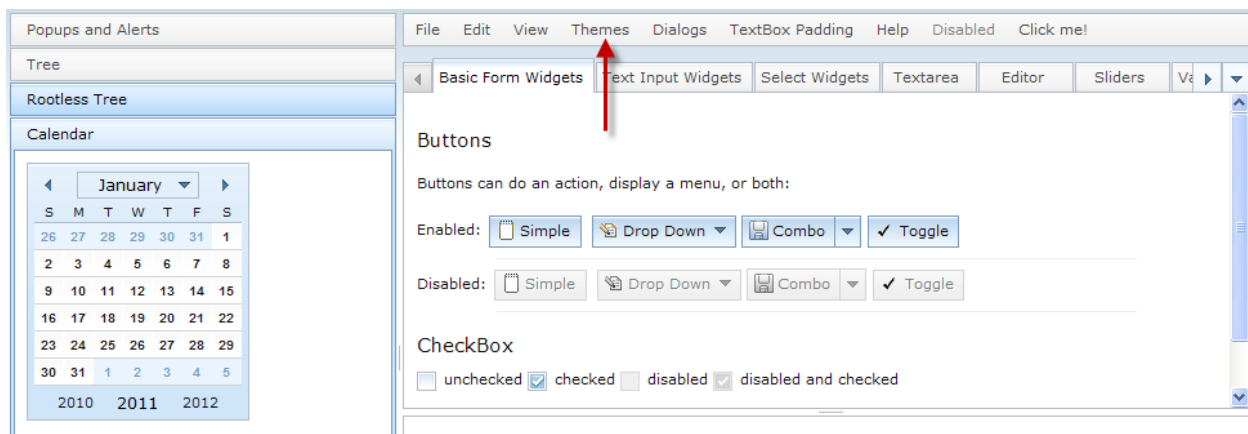


Figure 20: Dojo Theme Tester

You will also need to reference the ArcGIS Server API for JavaScript from within a <script> tag. Any <script> tags need to be defined inside the <head> tag of your web page. You must add a reference to the API for JavaScript along with a reference to the Dojo CSS theme for every mapping application that you build with the ArcGIS Server API for JavaScript. The reference to the API is absolutely necessary as it provides a link to all the objects provided by the API. In order to use any of the objects in the API you must provide this reference. The code example below shows the reference to the API along with the <link> element which references the style sheet. As of this writing in early 2012, the current version of the AGIS API for JavaScript is 2.6. The version is referenced at the end of the URL provided in the 'src' attribute. This version will change over time as new releases are made available. You can obtain information on the current version release at <http://help.arcgis.com/en/webapi/javascript/arcgis/>.

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Adding JavaScript API</title>

  <!-- include a link tag to include a dojo theme (tundra/soria/nihilo) -->
  <link rel="stylesheet" type="text/css"
        href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.6/js/dojo/dijit/themes/tundra/tundra.css">

  <!-- include a script tag with src url to api hosted on ArcGIS Online -->
  <script type="text/javascript" src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.6"></script>
</head>
<body>

</body>
</html>

```

References inside the <head> tag

Reference the style sheet

Reference the API for JavaScript

Step 2: Create a container for your map

The map needs to be displayed somewhere on a web page, and the best way to accomplish this is through an HTML <div> tag which acts as a container for the map. The <div> tag needs to be placed somewhere inside the <body> tag. Please also note that you need to use the ID attribute to assign a unique name to your map container as it will be referenced elsewhere in your code. Notice in the code example below that an ID of “map” has been assigned to this container. This ID will be used later in your JavaScript functions to reference the map. Also notice that we are referencing the “claro” Dojo theme. Since the reference is placed inside the <body> tag all widgets will have this theme applied.

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Adding a Tiled Layer</title>

  <link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.6/js/d
  <script type="text/javascript" src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.6"></script>
</head>
<body class="tundra">
  <div id="map" style="width:1024px; height:512px; border:1px solid #000;"></div>
</body>
</html>

```

Reference the Dojo CSS theme

Give Map an ID

Step 3: Create an initialization script that creates an instance of Map

In this step you’ll get your first look at a JavaScript function. We examined functions briefly in Chapter 3, but we’ll go into more detail now. JavaScript functions are blocks of code that are executed as the result of some type of event that has occurred in the application. This could be something as simple as a map click or dragging the mouse in a panning action. These are known as user generated events. Application generated events differ in that they are triggered by the application. The initialization script is executed as the result of an application event that fires after all the elements of an HTML page have loaded when initially displayed in a browser.

In this step you will create an initialization function that creates the map, adds layers, and

performs any other startup routines. You will use the ID that you assigned to your <div> tag in this step to place the newly created map inside the <div> container.

In the code example below you will see a code block that demonstrates a typical initialization script. Obviously before you can do anything you must create a map object. This is accomplished through a series of steps all of which are wrapped inside a JavaScript <script> tag.

Inside this <script> tag is an initialization function called `init()`. It's also common practice to name the initialization function "initialize()". The function name can be anything you wish that meets the JavaScript rules for function naming, but for the sake of this example we're simply calling it `init()`. You'll notice that the first line inside the <script> tag is `dojo.require("esri.map");` The ArcGIS API for JavaScript is built on top of Dojo so you use `dojo.require` to import resources for your application. In this case we need to access the ability to create a map which is provided through the "esri.map" resource.

Most of the work of creating the initial map and layers is accomplished inside the `init()` function. Our first order of business inside `init()` is to create a new map. This is done through the line `map = new esri.Map("map")`. Remember in Step 2 we added a <div> tag inside the body of our web page and gave it an ID of 'map'. What this line of code does is create a new map and place it inside our <div> container which contains an ID of 'map'.

After creating a map, we need to add layers of geographic data to the map. We'll cover adding data layers to your map later in the chapter but for now this is an example of adding a layer of GIS data to the application that references a map service.

Finally, outside the `init()` function we call `dojo.addOnLoad(init)` which initiates the `init()` function after the elements of the web page have loaded. The `dojo.addOnLoad(init)` function is the subject of the final step in our initialization process.

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis
  <script type="text/javascript" src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.6">
  <script type="text/javascript">
    dojo.require("esri.map");
    function init() {
      //create map, set initial extent and disable default info window behavior
      map = new esri.Map("map");
    }
    dojo.addOnLoad(init);
  </script>

</head>
<body class="tundra">
  <!-- map div -->
  <div id="map" style="width:800px; height:400px; border:1px solid #000;"></div>
  <br />

</body>
</html>

```

Annotations in the code example:

- Inside script tag (points to the opening <script> tag)
- Import the map resource (points to dojo.require("esri.map");)
- Create the Map (points to map = new esri.Map("map");)
- Initialize function (points to the function init() { ... }
- Create the Map (points to dojo.addOnLoad(init);)

Step 4: Call dojo.addOnLoad()

The final step in the process is to use the dojo.addOnLoad() function to trigger the execution of our initialization function. This function runs only after all HTML page elements for our application have been loaded. This is important because it ensures that any and all HTML elements can be used by our initialization function. Without this function we could not be certain that all elements on the page are available to the function and thus errors could occur. Refer back to the code example above to see the dojo.addOnLoad() function.

Summary

To summarize, for every ArcGIS Server API for JavaScript application that you create you will need to follow these four steps. Your application should appear similar to the code that you see below. Each step and its associated code reference are displayed.

```

<html>
<head>

    <title>Create Map</title>

    ①<link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.1/js/dojo/dijit/themes/claro/claro.css"

    ①<script type="text/javascript" src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.1"></script>

    <script type="text/javascript">
        dojo.require("esri.map");

    ③function init() {
        var map = new esri.Map("map");

        var imageParameters = new esri.layers.ImageParameters();
        imageParameters.format = "jpeg"; //set the image type to PNG24, note default is PNG8.
        //Takes a URL to a non cached map service.
        var dynamicMapServiceLayer = new esri.layers.ArcGISDynamicMapServiceLayer
            ("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Demographics/ESRI_Population_World/MapServer",
            {slider: false, nav:true, "opacity":0.5, "imageParameters":imageParameters});
        map.addLayer(dynamicMapServiceLayer);
    }

    ④dojo.addOnLoad(init);
    </script>
</head>
<body class="claro">
    ②<div id="map" style="width:900px; height:600px; border:1px solid #000;"></div>

    </body>
</html>

```

Loading this file in a web browser will produce a map of world population as seen in the figure below.

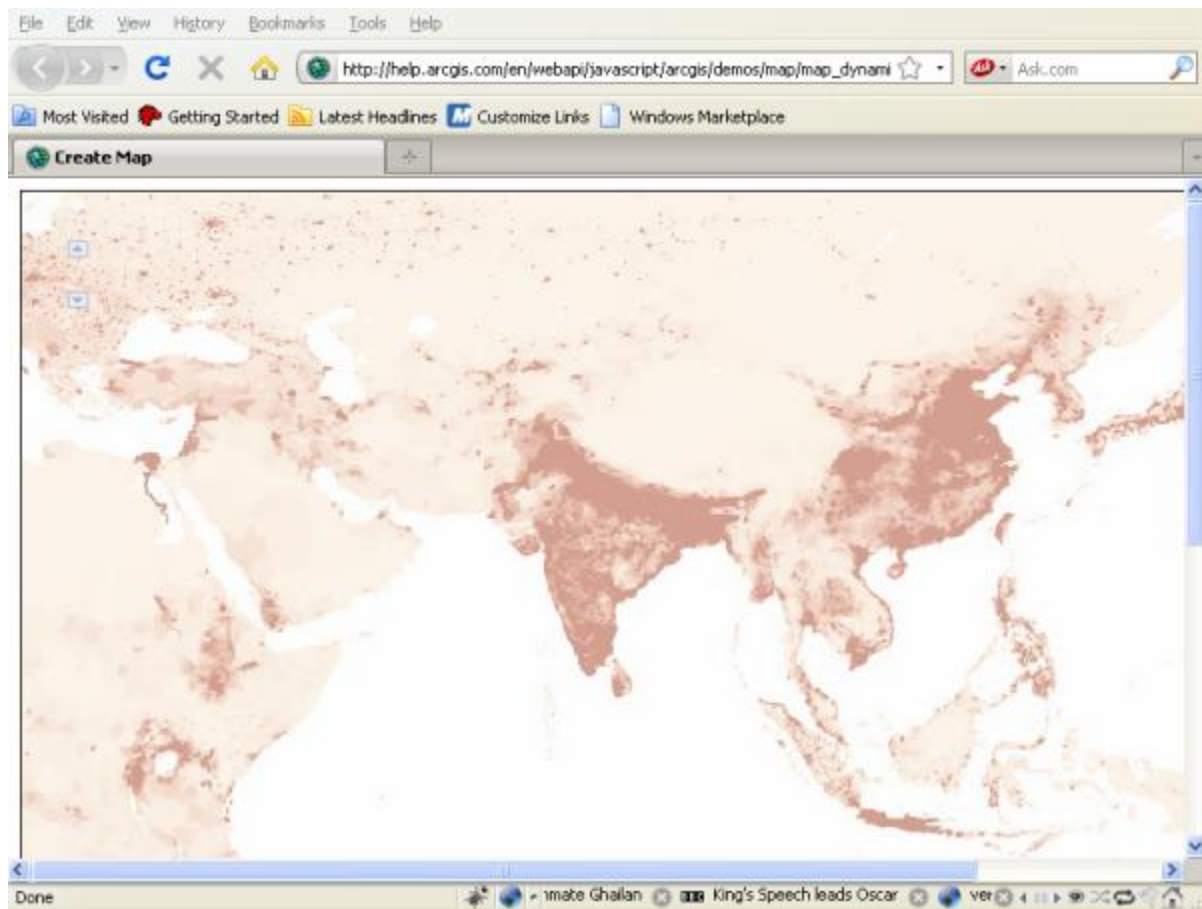


Figure 21: Creating a Map

Creating a Map

In the four step process described above we introduced the process that you'll need to follow for each application that you build with the ArcGIS Server API for JavaScript. In step 3 you learned how to create an initialization JavaScript function. The purpose of the initialization script is to create your map, add layers, and perform any other setup routines necessary to get your application started. Creating a Map is invariably one of the first things that you'll do and in this section we'll take a closer look at the various options you have for creating an instance of the Map class.

In object-oriented programming the creation of a class instance is often done through the use of a constructor. A constructor is a function that is used to create or initialize a new object. In this case the constructor is used to create a new Map object. Constructors frequently take one or more parameters that can be used to set the initial state of an object.

Including Resources

However, before you can call the constructor for a Map you must first reference the resource that provides the map. This is accomplished through the use of `dojo.require()`. `Dojo.require()` is used to import resources into your web page. Various resources are provided by the API for JavaScript including the `dojo.require("esri.map")` resource which must be provided before you can create a map or work with geometry, graphics, and symbols. Once a reference to the resource has been provided you can use the Map constructor to create the Map.

```
<script type="text/javascript">  
    dojo.require("esri.map");  
  
    function init() {  
        map = new esri.Map("map");  
    }  
  
</script>
```

The most commonly used resources are listed in the table below.

Resource	Use for:
esri.map	Map, geometry, graphics, and symbols
esri.layers.agsdynamic	ArcGISDynamicMapServiceLayer
esri.layers.agstiled	ArcGISTiledMapServiceLayer
esri.tasks.find	Find task
esri.tasks.geometry	Geometry task
esri.tasks.gp	Geoprocessing task
esri.tasks.identify	Identify task
esri.tasks.locator	Locator task
esri.tasks.query	Query task
esri.toolbars.draw	Draw
esri.toolbars.navigation	Navigation

The Map Constructor

The constructor for the Map class is shown below where 'divID' is a reference to the container for the map (usually a <div>) and the options specify one or more optional parameters such as

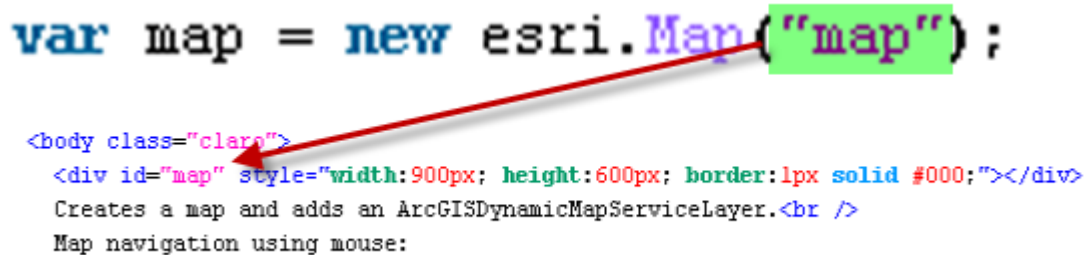
the initial map extent. Take note of the question mark that follows 'options' in the constructor signature. This simply indicates that this is an optional parameter. Therefore, the only parameter that you must pass to the constructor is the 'divID'

esri.map(divID, options?)

When creating a new object using a constructor in JavaScript you will need to use the "new" keyword as seen in the Map constructor example below. The code example also assigns this new Map object to an object variable called 'map'. This 'map' variable holds a reference to the newly created map.

The Map constructor can be passed two parameters including the container where the map should reside and various options for the map.

A <div> ID is a required parameter for the constructor and is used to specify the container for the map. This is illustrated in the figure below. When creating a new instance of the Map class it is required that you provided a reference to the <div> tag that will serve as the container for your map. This is done by passing in the ID of the <div> tag (in this case it's 'map').



```
var map = new esri.Map("map");
```

`<body class="clear">`
`<div id="map" style="width:900px; height:600px; border:1px solid #000;"></div>`
Creates a map and adds an ArcGISDynamicMapServiceLayer.

Map navigation using mouse:

A red arrow points from the "map" string in the JavaScript code to the "map" ID attribute in the HTML code.

In addition, you can also pass in multiple options that control various aspects of the map including the initial extent, display of navigation controls, graphic display during panning, control of the slider, levels of detail, and more.

Let's take a closer look at how options are specified in the map constructor. Options, the second parameter in the constructor, are always enclosed with brackets. Inside the brackets each option has a specific name and is followed by a colon and then the data value that controls the option. In the event that you need to submit multiple options to the constructor, each option is separated by a comma. Below you see a code example showing how options are submitted to the Map constructor. In this case we are defining options for the initial extent, slider, and nav.

```
var map = new esri.Map("map", {extent: new esri.geometry.Extent(-124,31,-113,42), slider: false, nav: true});
```

Setting the initial extent is probably the most commonly used option of those available. This parameter controls the initial geographic extent of the map, but there are several other options that you can set including:

- layer – The base layer used to initialize the map
- nav – True/False value indicating whether to display pan buttons on the map
- displayGraphicsOnPan – True/False value indicating whether graphics should be displayed during panning.
- slider – True/False value indicating if the zoom slider should be displayed

The Spatial Reference

The MXD or MSD used to create a map service defines the spatial reference that will be used for that service. Dynamic map service layers have the ability to project data on the fly to that of other map services in your application. So, if you are using a WGS 84 base map from ArcGIS Online, your dynamic UTM service will overlay the base map using on the fly projection. Map service projection on the fly does not work with cached map services. So for those map services that you are going to cache you need to change the projection of the map to the projection of the other services you are going to mash up with.

Working with Map Service Layers

A map without data layers is sort of like an artist with a blank canvas. The data layers that you add to your map give it meaning and set the stage for analysis. There are two primary types of map services that provide data layers that can be added to your map: dynamic map service layers and tiled map service layers.

Dynamic Map Services

Dynamic map service layers reference map services that create a map image on the fly and then return the image to the application. This type of map service may be composed of one or more layers of information. For example, the Demographics map service displayed in the figure below is composed of 9 different layers representing demographic information at various levels of geography.

Demographics (MapServer)

View In: [ArcMap](#) [ArcGIS Explorer](#) [ArcGIS JavaScript](#) [Google Earth](#)

View Footprint In: [Google Earth](#)

Service Description:

Map Name: Layers

Layers:

- [Demographics/ESRI Census USA](#) (0)
 - [Census Block Points](#) (1)
 - [Census Block Group](#) (2)
 - [Counties](#) (3)
 - [Coarse Counties](#) (4)
 - [Detailed Counties](#) (5)
 - [States](#) (6)
- [ESRI StreetMap World 2D](#) (7)
 - [World Street Map](#) (8)

Figure 22: Demographics Map Service

While they can take somewhat longer to display in a client application since they must be generated “on the fly”, dynamic map service layers are more versatile than tiled map service layers in that you can control the features displayed through layer definitions, set the visibility of various layers within the service, and define temporal information for the layer. For example, in the Demographics map service layer detailed above you might elect to only display Census Block Groups in your application. This is the type of versatility provided by dynamic map service layers that you don’t get with tiled map service layers.

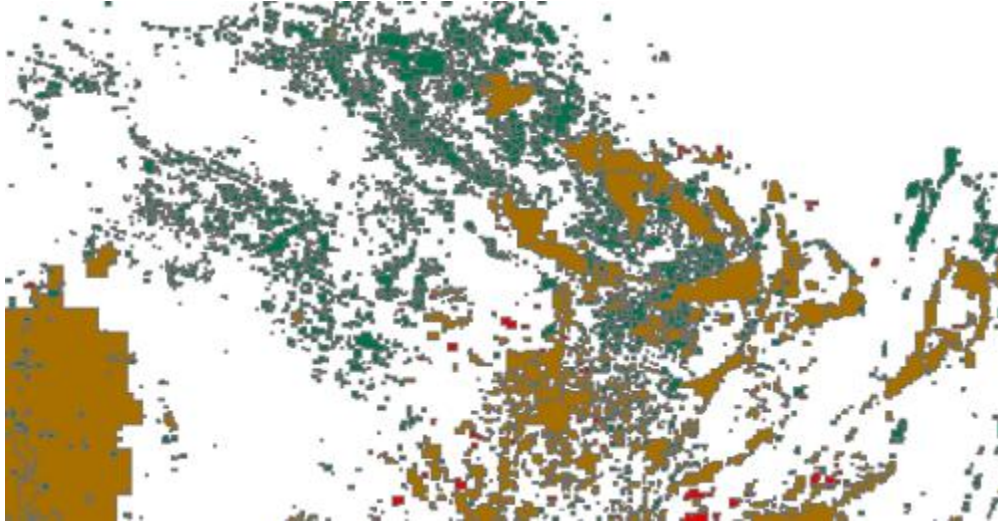


Figure 23: Dynamic Map Service Layers

Tiled map service layers (see Figure X) reference a pre-defined cache of map tiles instead of dynamically rendered images.



Figure 24: Tiled (Cached) Layer

Tiled Map Service Layers

The easiest way to understand the concept of tiled map services is to think about a grid that has been draped across the surface of a map. Each cell within the grid has the same size and will be used to cut the map into individual image files called tiles. The individual tiles are stored as image files on a server and retrieved as necessary depending upon the map extent and scale. This same process is often repeated at various map scales. The end result is a cache of tilesets that have been generated for various map scales. When the map is displayed in the application it will appear to be seamless even though it is composed of many individual tiles.

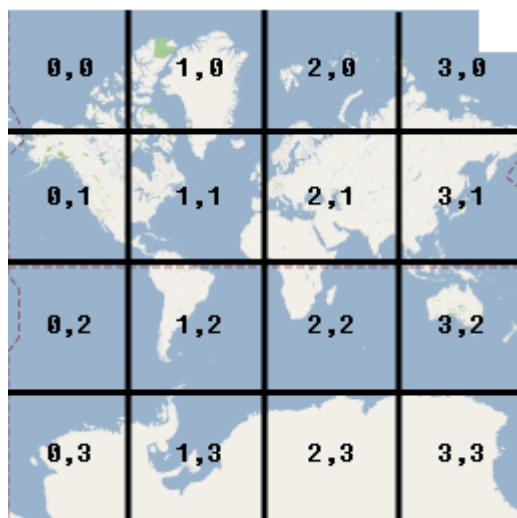


Figure 25: Map Tiles

These tiled, or cached, map layers are often used as base maps including imagery, street maps, topographic maps, and others. Tiled map services tend to display somewhat faster since they don't have the overhead of creating images on the fly each time there is a request for a map.

Operational layers are then draped on top of the tiled base maps and these are often dynamic layers. While they can be somewhat slower in terms of performance, dynamic map service layers have the advantage of being able to define their appearance on the fly.

Using the Layer Classes

Using the Layer classes from the API for JavaScript you can reference map services hosted by ArcGIS Server and other map servers. Referring to the object model diagram in [Figure X](#) you can see that all layer classes inherit from the Layer base class. The Layer class has no constructor so you can't specifically create an object from this class. This class simply defines properties, methods, and events that are inherited by all sub-classes of Layer.

DynamicMapServiceLayer, TiledMapServiceLayer, and GraphicsLayer all inherit directly from the Layer class. DynamicMapServiceLayer and TiledMapServiceLayer also act as base classes. DynamicMapServiceLayer is the base class for dynamic map services while

TiledMapServiceLayer is the base class for tiled map services. [Chapter <X>](#) is devoted entirely to graphics and the GraphicsLayer so we'll save our discussion of this layer type for later in the book. Layer, DynamicMapServiceLayer, and TiledMapServiceLayer are all base classes meaning that you can't specifically create an object from these classes in your application.

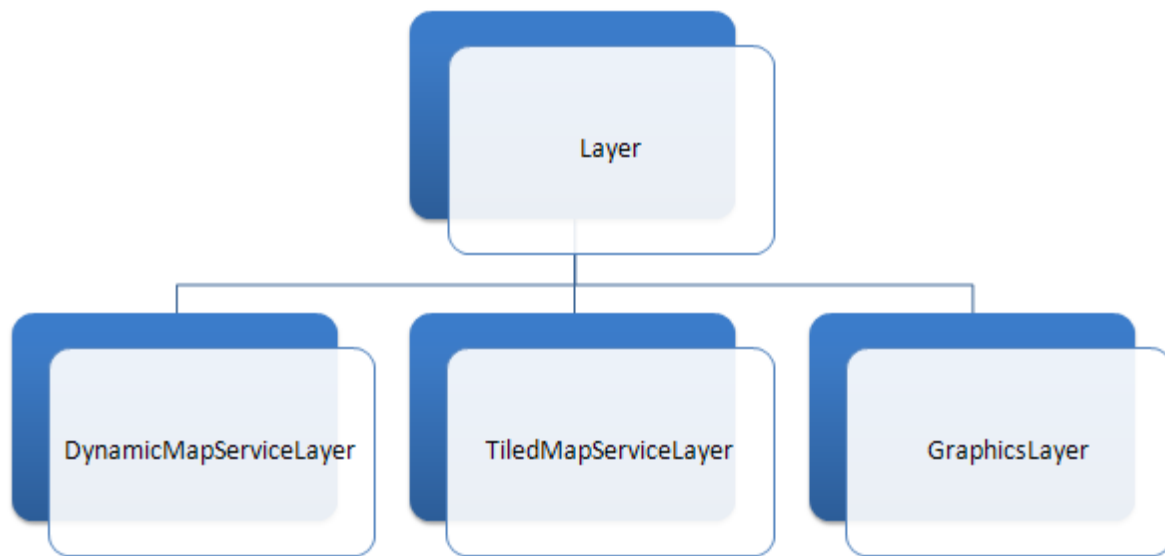


Figure 26: Layer OMD

Tiled Map Service Layers

ArcGISTiledMapServiceLayer

As I mentioned, tiled map service layers reference a cache of pre-defined images that are tiled together to create a seamless map display. These are often used as base maps.

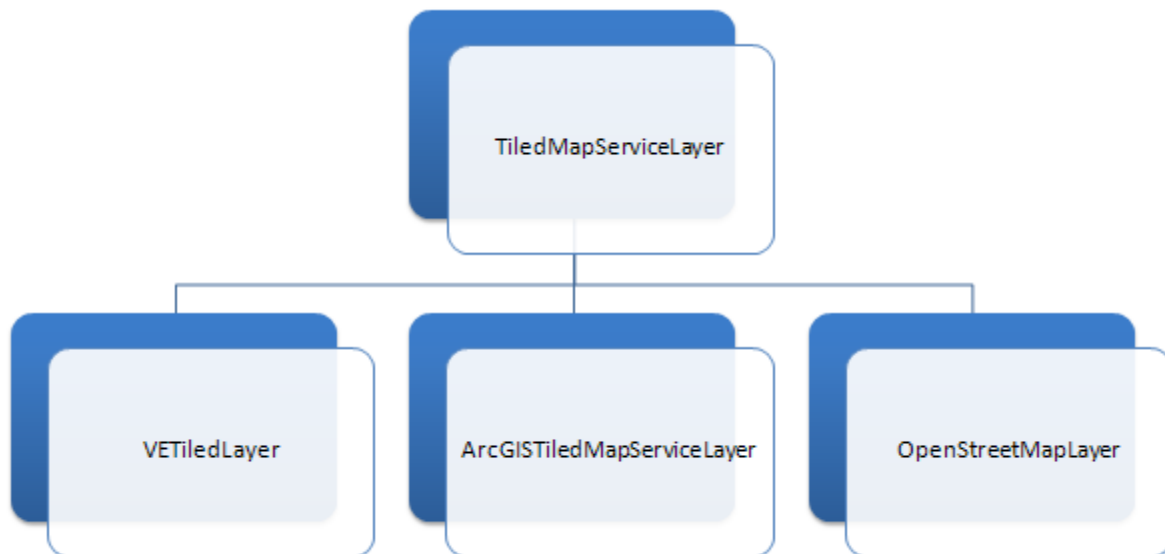


Figure 27: Tiled Map Service Layers

The `ArcGISTiledMapServiceLayer` class is used when referencing a tiled (cached) map service exposed by ArcGIS Server. Since this type of object works against a tiled set of maps that have

been cached, performance is often improved. The constructor for `ArcGISTiledMapServiceLayer` takes an URL pointer to the map service along with options that allow you to assign an ID to the map service, and control transparency and visibility. In the code example below, notice that the constructor for `ArcGISTiledMapServiceLayer` takes a parameter that is a URL to a map service. After an instance of a layer has been created it is then added to the map using the `Map.addLayer()` method which accepts a variable that contains a reference to the tiled map service layer.

```
//create tiled map layer
var tiled = new esri.layers.ArcGISTiledMapServiceLayer("http://server.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/MapServer");
//add layer to map
map.addLayer(tiled);
```

`ArcGISTiledMapServiceLayer` is used primarily for the fast display of cached map data. You can also control the levels at which the data will be displayed. For instance, you may want to display data from a generalized `ArcGISTiledMapService` showing interstates and highways while your users are zoomed out at levels 0-6 and then switch to a more detailed `ArcGISTiledMapService` once the user zooms in further. You can also control the transparency of each layer added to the map.

VETiledLayer and OpenStreetMapLayer

The `VETiledLayer` and `OpenMapStreetLayer` classes are used to load a Bing Maps or Open Street Map layers respectively. The tilesets for these layers are not pulled from an ArcGIS Server instance but rather servers hosted by Microsoft and Open Street Map. The constructor for both classes takes a single parameter that defines the options for the layer. For `VETiledLayer` the options would include a Bing Maps key that you can obtain from the Bing Maps website.

Options for the `OpenStreetMapLayer` constructor include a transparency value, display levels, id, visibility, and the tile servers to use. The code example below shows the creation of an `OpenStreetMapLayer` object with various options passed in.

```
osmLayer = new esri.layers.OpenStreetMapLayer({
  id: "myOSMLayer",
  visible: true,
  opacity: .75,
  displayLevels: [0,1,2]
});
```

The options for the `OpenStreetMapLayer` constructor are optional so you could simply call the constructor without passing in any options.

```
osmLayer = new esri.layers.OpenStreetMapLayer();  
map.addLayer(osmLayer);
```

Dynamic Map Service Layers

ArcGISDynamicMapServiceLayer

As the name suggests, the *ArcGISDynamicMapServiceLayer* class is used to create dynamic maps served by ArcGIS Server. Just as with *ArcGISTiledMapServiceLayer* the constructor for *ArcGISDynamicMapServiceLayer* takes a URL that points to the map service along with optional parameters used to assign an ID to the service, determine the transparency of the map image and a visibility option which sets the initial visibility of the layer to true or false. The class name *ArcGISDynamicMapServiceLayer* can be somewhat misleading. Although it appears to reference an individual data layer this is in fact not the case. It refers to a map service rather than a data layer. Individual layers inside the map service can be turned on/off through the *setVisibleLayers()* method.

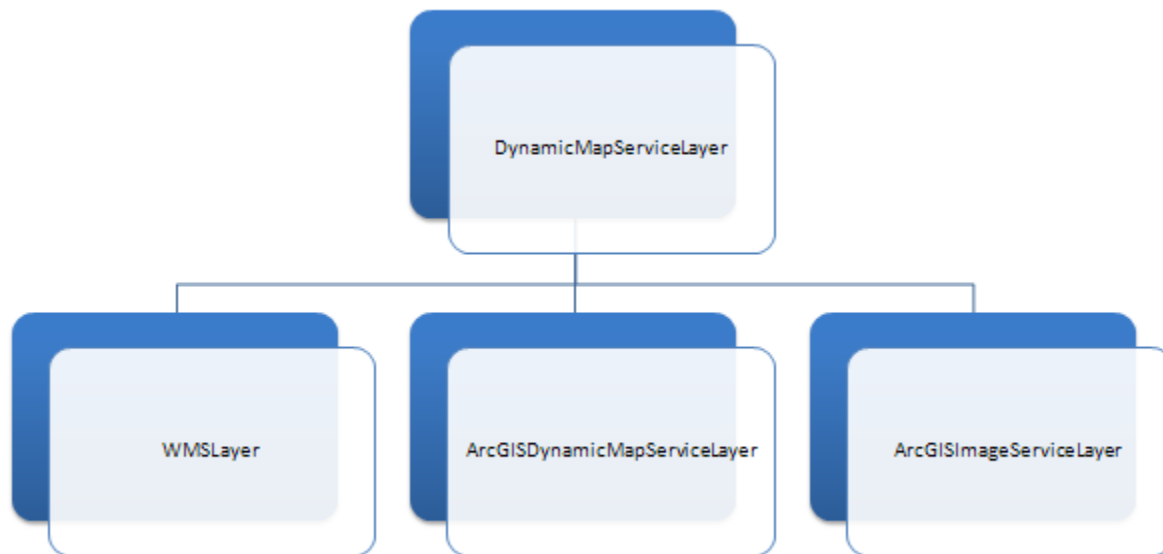


Figure 28: Dynamic Map Service Layers

Creating an instance of *ArcGISDynamicMapServiceLayer* will look very similar to *ArcGISTiledMapServiceLayer*. The code example below illustrates this. The constructor accepts a URL that points to the map service. The second parameter defines the optional parameters that you can supply to control transparency, visibility, and image parameters.


```
var opLayer = new esri.layers.ArcGISDynamicMapServiceLayer(
    "http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Demographics/ESRI_Population_World/MapServer",
    {"opacity":0.5})
);
map.addLayer(opLayer)
```

With an instance of `ArcGISDynamicMapServiceLayer` you can perform a number of operations. Obviously you can create maps that display the data in the service, but you can also query data from layers in the service, control feature display through layer definitions, control individual layer visibility, set time related information, export maps as images, control background transparency, and more.

After a new instance has been created you need to add it to your map so that it will be visible.

Adding Layers to the Map

The `Map.addLayer()` method takes an instance of a layer (`ArcGISDynamicMapServiceLayer` or `ArcGISTiledMapServiceLayer`) as the first parameter, and an optional index that specifies where it should be placed. In the code example below we create a new instance of `ArcGISDynamicMapServiceLayer` pointing to a URL for the service. We then call `Map.addLayer()`, passing in the new instance of the layer. The layers in the service will now be visible on the map. The `addLayers()` method takes an array of layer objects and adds them all at once.

```
var aquaUrl = "http://gistxsawh01/arcgis/rest/services/Aqua/MapServer";
aquaMap = new esri.layers.ArcGISDynamicMapServiceLayer(aquaUrl);
```

```
map.addLayer(aquaMap);
```



Figure 29: Adding a Layer

In addition to being able to add layers to a map you can also remove layers from a map using `Map.removeLayer()` or `Map.removeAllLayers()`.

Practice Time - Hello Map

It is a long standing tradition of programming books to have you write an incredibly simply “Hello World” application to get started. Well, we’re going to start our own similar tradition by creating a “Hello Map” application! It won’t be quite as simple as most “Hello World”

applications but if you've understood the previously described basic steps for creating an application with the API for JavaScript then you shouldn't have too much trouble.

Getting and Setting the Map Extent

One of the first things you'll want to master is getting and setting the map extent. By default the initial extent of a map within your application is the extent of the map when it was last saved in the map document file (.mxd) used to create the map service. In some cases this may be exactly what you want, but in the event that you need to set a map extent other than the default you have several options. The first and easiest method of setting the initial map extent is to specify the coordinates directly in the constructor for the Map object.

The initial extent of the map is not a required parameter, and thus if you leave out this information the map will simply use the default extent. This is shown in the first code example below where only the ID of the container is specified.

```
map = new esri.Map("map");
```

To set the initial extent in the constructor you simply specify an extent and use this extent as an option in the Map constructor. The code example below illustrates this process. A new instance of the Extent class is created and stored in a variable called 'initExtent'. This variable containing the extent is then used in the constructor for the Map object.

```
var initExtent = new esri.geometry.Extent({"xmin":-79.35, "ymin":30.45, "xmax":-85.32, "ymax":45.35});  
map = new esri.Map("map", {extent:initExtent});
```

Rather than setting the initial extent through the constructor you could use the Map.setExtent method seen in the code example below.

```
var initExtent = new esri.geometry.Extent(  
    {"xmin":-79.35, "ymin":30.45, "xmax":-85.32, "ymax":45.35});  
map = new esri.Map("map");  
map.setExtent(initExtent);
```

There may be times when you are using multiple map services in your application. In this case, setting the initial map extent can be done either through the constructor for your map or by using the Map.fullExtent method on one of the services. For example, it is common to use a map service that provides base layer capabilities containing aerial imagery along with a map service containing your own local operational data sources. The code example below uses the fullExtent() method.

```
map = new esri.Map("map", {extent:esri.geometry.geographicToWebMercator(myService.fullExtent())});
```

The current extent of a map can be obtained either through the Map.extent property or the onExtentChange event. Please note that the Map.extent property is read-only so don't attempt to set the map extent through this property.

Setting the Visible Layers from a Map Service

You can control the visibility of individual layers within a dynamic map service layer using the setVisibleLayers() method. This only applies to dynamic map service layers, not tiled map service layers. This method takes an array of integers corresponding to the data layers in the map service. This array is 0 based, so the first layer in the map service occupies position 0. In our Demographics map service illustrated in the figure below, Demographics/ESRI_Census_USA occupies index 0. Therefore, in the event that we'd like to display only the Census Block Points and Census Block Groups from this service we could use setVisibleLayers() as seen in the code example below.

```
var opLayer = new esri.layers.ArcGISDynamicMapServiceLayer(  
    "http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Demographics/ESRI_Population_World/MapServer",  
    {"opacity":0.5})  
);  
opLayer.setVisibleLayers([1,2]);  
map.addLayer(opLayer)
```

Demographics (MapServer)

View In: [ArcMap](#) [ArcGIS Explorer](#) [ArcGIS JavaScript](#) [Google Earth](#)

View Footprint In: [Google Earth](#)

Service Description:

Map Name: Layers

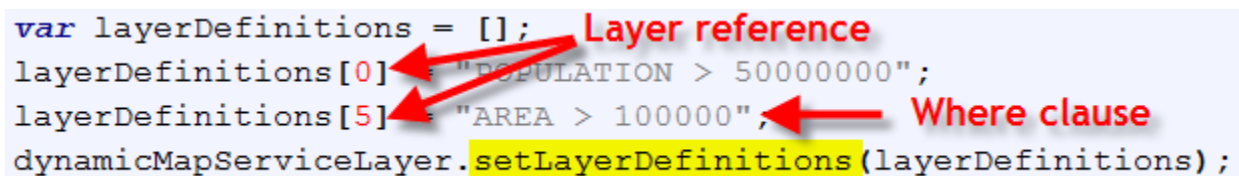
Layers:

- [Demographics/ESRI_Census_USA](#) (0)
 - [Census Block Points](#) (1)
 - [Census Block Group](#) (2)
 - [Counties](#) (3)
 - [Coarse Counties](#) (4)
 - [Detailed Counties](#) (5)
 - [States](#) (6)
- [ESRI_StreetMap_World_2D](#) (7)
 - [World Street Map](#) (8)

Setting a Definition Expression

In ArcGIS Desktop you can use a Definition Expression to limit the features in a data layer that will be displayed. A definition expression is simply a SQL query against the columns and rows in a layer. Only the features whose attributes meet the query are displayed. For example, if you only wanted to display cities with a population greater than 1 million the expression would be something like "POPULATION > 1000000". The ArcGIS Server API for JavaScript contains a `setLayerDefinitions()` method that accepts an array of definitions that can be applied against `ArcGISDynamicMapServiceLayer` to control the display of features in the resulting map. The code example below shows how this is done. You first create an array that will hold multiple where clauses which will serve as the definition expressions for each layer. In this case we are defining layer definitions for the first and sixth layer. The array is zero based so the first array is at index 0. The where clauses are placed into the array and then passed into the `setLayerDefinitions` method. ArcGIS Server then renders the features that match the where clauses for each layer.

```
var layerDefinitions = [];
layerDefinitions[0] = "POPULATION > 50000000";
layerDefinitions[5] = "AREA > 100000";
dynamicMapServiceLayer.setLayerDefinitions(layerDefinitions);
```



Map Navigation – Map Extent, Zooming, and Panning

Now that you know a little about maps and the layers that reside within those maps it's time to learn how to control map navigation in your application. In most cases your users will need to be able to navigate around the map through panning and zooming. The ArcGIS Server API for JavaScript provides a number of user interface widgets and toolbars that you can use to allow your user to change the current map extent through zooming and panning. Map navigation can also occur through keyboard navigation and mouse navigation. In addition to these user interface components and hardware interfaces, map navigation can also be controlled programmatically.

Map Navigation Widgets and Toolbars

The simplest way to provide map navigation control to your application is through the addition of various widgets and toolbars. By default, when you create a new map and add layers a zoom slider (see figure x below) is included with the map. This slider allows the user to zoom in and out on the map. You don't have to do anything programmatically to have the zoom slider appear on your map. It is present by default.



Figure 30: Navigation Widget

However, you can remove the slider if necessary from your application simply by setting the 'slider' option to 'false' when creating an instance of the Map object.

```
{slider: false, nav:true, opacity:0.4, imageParameters:imageParameters}
```

You can also add pan buttons that will pan the map in the direction that the arrow points when clicked. By default pan buttons will not appear on the map. You must specifically set the 'nav' option to 'true' when creating your Map object.

```
{slider: false, nav:true, opacity:0.4, imageParameters:imageParameters}
```

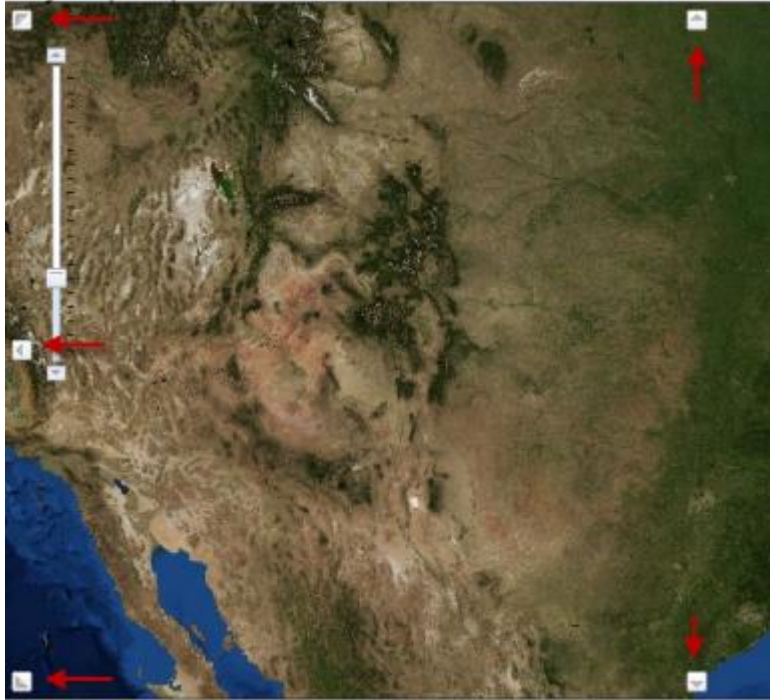


Figure 31: Pan Buttons

The ArcGIS API for JavaScript also gives you the ability to add several types of toolbars to your application including a navigation toolbar containing buttons for zooming in and out, panning, full extent, next extent and previous extent. The topic of toolbar creation is covered in detail in a later chapter so we'll save that discussion for later. I do want to make you aware of the existence of this toolbar though. Creating a toolbar with the API for JavaScript is not a user interface component that you just “drop into” your application, but it is a fairly simple process to add these toolbar to your application.



Figure 32: Navigation Toolbar

Controlling Map Navigation with the Mouse and Keyboard

Users can also control map navigation with the mouse and/or keyboard devices. By default, users can do the following:

- Drag the mouse to pan
- Mouse Scroll Forward to zoom in
- Mouse Scroll Backward to zoom out

- SHIFT + Drag the mouse to zoom in
- SHIFT + CTRL + Drag the mouse to zoom out
- SHIFT + Click to recenter
- Double Click to Center and Zoom in
- SHIFT + Double Click to Center and Zoom in
- Use arrow keys to pan
- Use + key to zoom in a level
- Use - key to zoom out a level

These options can be disabled using one of several Map methods. For example, to disable scroll wheel zooming you would use the `Map.disableScrollWheelZoom()` method.

Note: For a more in-depth treatment of the ArcGIS Server API for JavaScript please refer to our [Building Custom ArcGIS Server Applications with JavaScript](#). This course is offered as both a traditional instructor led course and an instructor guided online course. For more information on this course please visit our website at geospatialtraining.com