Chapter 6: Adding Graphics to the Map

In addition to traditional map service layers exposed by ArcGIS Server an application can also draw point, line, and polygon graphics in a separate layer on the map. This layer, called a GraphicsLayer, stores all graphics associated with your map. In Chapter 4 we discussed the various types of layers including dynamic map service layers and tiled map service layers. Just as with these other types of layers, GraphicsLayer also inherits from the Layer class. Therefore, all the properties, methods, and events found on the Layer class will also be present on GraphicsLayer.

Graphics are displayed on top of any other layers that are present in an application. These graphics can be created by users or drawn by the application as results of tasks that have been submitted to ArcGIS Server. For example, a business analysis application might provide a tool that allows the user to draw a free-hand polygon representing a potential trade area. The polygon graphic would be displayed on top of the map, and could then be used as input to a geoprocessing task that pulls demographic information pertaining to the potential trade area.

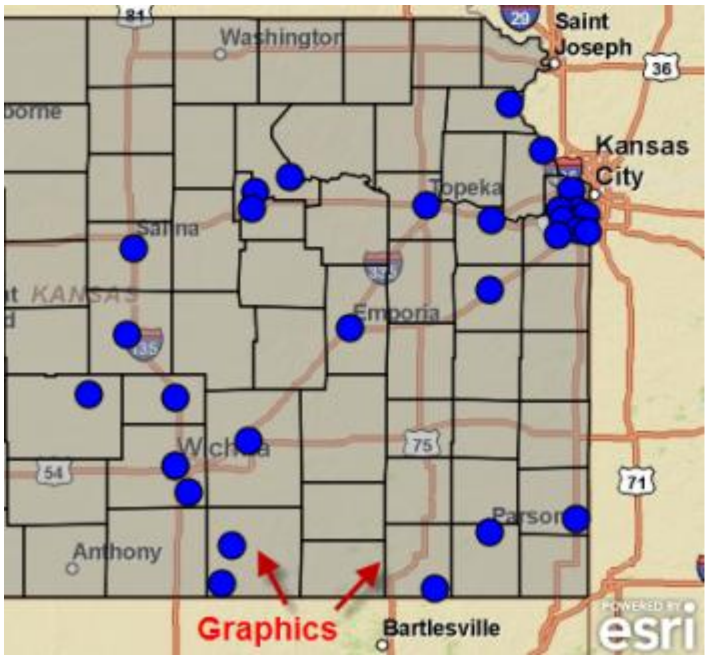


Figure 34: Graphics

Many ArcGIS Server tasks return their results as graphics. The Query Task can perform both attribute and spatial queries. The results of a query are then returned to the application in the form of a FeatureSet object which is simply an array of graphics. Application code is then used to loop through this FeatureSet and plot each graphic on the map. Perhaps you'd like to find and display all land parcels that intersect the 100 year flood plain. A Query Task could perform the spatial query and then return the results to your application where they'd then be displayed as polygon graphics on the map.

The Four Parts of a Graphic

A graphic is composed of four items: geometry, symbol, attributes, and an InfoTemplate. Most people intuitively understand the idea that a graphic has a geometric representation that describes where it is located. The geometry along with a symbol defines how the graphic is displayed. However, a graphic can also have attributes which are name-value pairs that describe the graphic as well as an Info Template that defines the format of the InfoWindow that appears when a graphic is clicked. After creation, graphic objects must be stored inside a GraphicsLayer object before they can be displayed on the map. This GraphicsLayer object functions as a container for all graphics that will be displayed.

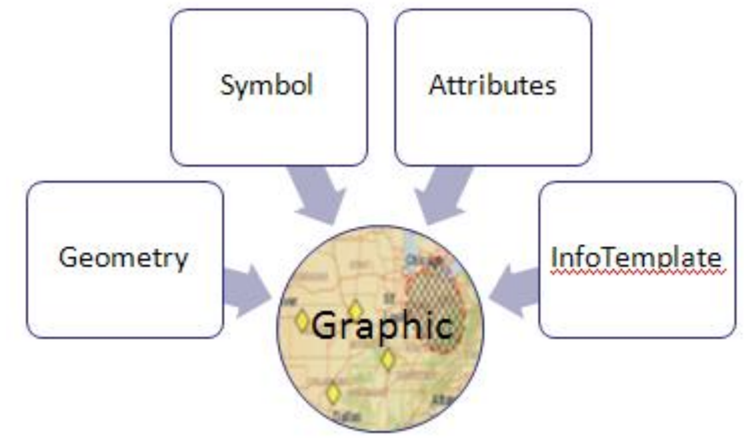
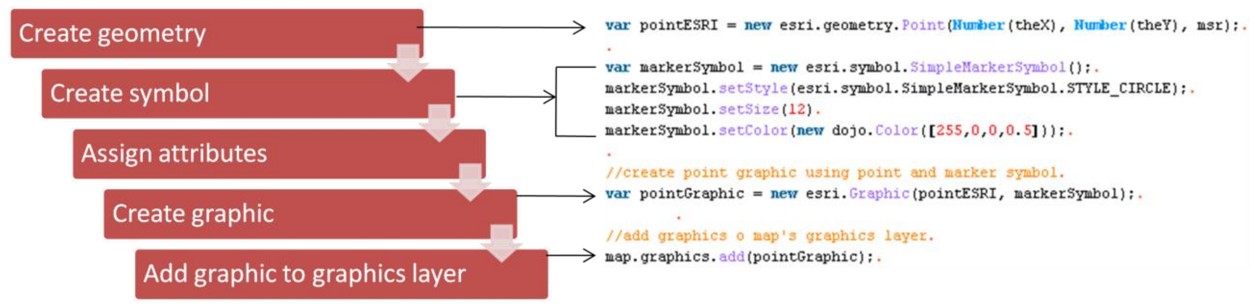


Figure 35: Parts of a Graphic

All elements of a graphic are optional. However, the geometry and symbology of a graphic are almost always assigned. Without these two items there would be nothing to display on the map, and there isn't much point in having a graphic unless you're going to display it.

Below you will see a code example illustrating the typical process for creating a graphic and adding it to the graphics layer. In this case we are applying the geometry of the graphic as well as a symbol for depicting the graphic. However, we haven't specifically assigned attributes or an InfoTemplate to this graphic.



```
var pointESRI = new esri.geometry.Point(Number(theX), Number(theY), msr);

var markerSymbol = new esri.symbol.SimpleMarkerSymbol();
markerSymbol.setStyle(esri.symbol.SimpleMarkerSymbol.STYLE_CIRCLE);
markerSymbol.setSize(12).
markerSymbol.setColor(new dojo.Color([255,0,0,0.5]));

//create point graphic using point and marker symbol.
var pointGraphic = new esri.Graphic(pointESRI, markerSymbol);

//add graphics o map's graphics layer.
map.graphics.add(pointGraphic);
```

Figure 36: Coding Graphics

Creating Geometry for Graphics

Graphics will almost always have a geometry component which is necessary for placement on the map. These geometry objects can be points, multi-points, polylines, polygons or extents and can be created programmatically through these objects or they can be returned as output from a task such as a query.

Before creating any of these geometry types the esri.geometry resource needs to be imported.

```
dojo.require("esri.geometry");
```

The geometry resource contains classes for Geometry, Point, Multipoint, Polyline, Polygon, and Extent.

Geometry is the base class which is inherited by Point, MultiPoint, Polyline, Polygon, and Extent classes. The primary method of importance on this class is setSpatialReference() which allows you to set the spatial reference for a geometry.

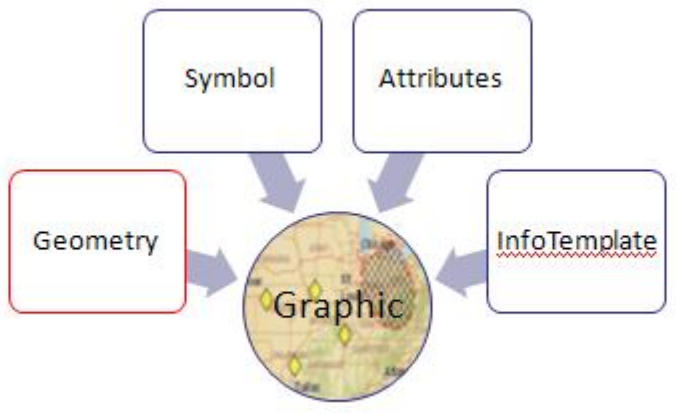


Figure 37: Graphic Geometry

```
var pt = new esri.geometry.Point(-118.15, 33.80, new esri.SpatialReference({wkid:4326});
```

The Point class defines a location by an X and Y coordinate, and can be in either map units or screen units. Multipoint, Polyline, Polygon, and Extent classes can be used to create their respective geometries.

Symbolizing Graphics

Each graphic that you create can be symbolized through one of the various symbol classes found in the API. Point graphics are symbolized by the SimpleMarkerSymbol class and the available shapes include circle, cross, diamond, square, and X. It is also possible to symbolize your points through the PictureMarkerSymbol class which uses an image to display the graphic. Linear features are symbolized through the SimpleLineSymbol class and can include solid lines, dashes, dots, or a combination. Polygons are symbolized through the SimpleFillSymbol class and can be solid, transparent, or cross hatch. In the event that you'd prefer to use an image in a repeating pattern for your polygons the PictureFillSymbol class is available. Text can also be added to a GraphicsLayer and is symbolized through the TextSymbol class.

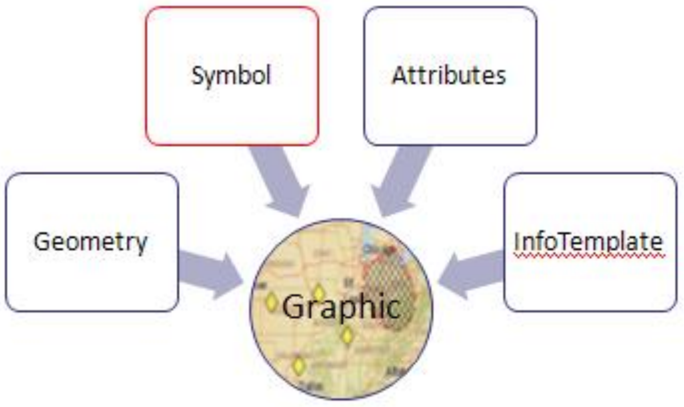
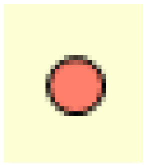


Figure 38: Graphic Symbology

Points or multi-points can be symbolized through the SimpleMarkerSymbol class which has various properties that can be set including the style, size, outline, and color. Style is set through the SimpleMarkerSymbol.setStyle( ) method which takes accepts a constant value as a parameter that corresponds to the type of symbol that is drawn (circle, cross, diamond, etc). Point graphics can also have an outline which is created through the SimpleLineSymbol class.

```
var markerSymbol = new esri.symbol.SimpleMarkerSymbol();
markerSymbol.setStyle(esri.symbol.SimpleMarkerSymbol.STYLE_CIRCLE);
markerSymbol.setSize(12)
markerSymbol.setColor(new dojo.Color([25,0,0,0.5]));
```

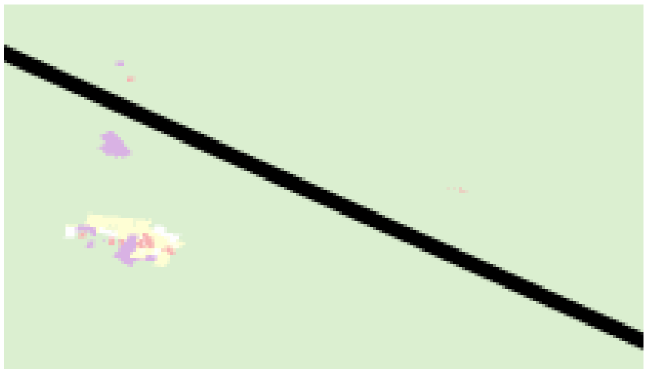


Linear features are symbolized with the SimpleLineSymbol class and can be a solid line or a combination of dots and dashes. Other properties of line symbols include color as defined with dojo.color and a width property for setting the thickness of your line.

```
var polyline = new esri.geometry.Polyline(msr)
var path = [new esri.geometry.Point(-123.123, 45.45, msr),<additional vertices>];
polyline.addPath(path);

var lineSymbol = new esri.symbol.SimpleLineSymbol().setWidth(5);

var polylineGraphic = new esri.Graphic(polyline, lineSymbol);
map.graphics.add(polylineGraphic);
```



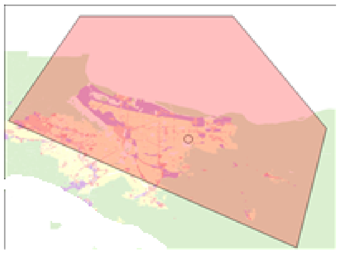
Polygons are symbolized with the SimpleFillSymbol class which allows for the drawing of polygons in solid, transparent, or cross hatch patterns. Polygons can also have an outline specified by a SimpleLineSymbol object.

```
var polygon = new esri.geometry.Polygon(msr);
var ring = [[-122.98, 45.55], [-122.21, 45.21], [-122.13, 45.53]];
polygon.addRing(ring);

var fillSymbol = new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255,0,0,0.25]));

var polygonGraphic = new esri.Graphic(polygon, fillSymbol);

map.graphics.add(polygonGraphic);
```



Symbolizing Graphics with Renderers

A renderer can be used to define a set of symbols for graphics contained within a GraphicsLayer. These symbols can have different colors and/or sizes based on an attribute. The four types of renderer in the ArcGIS Server API for JavaScript include SimpleRenderer, ClassBreaksRenderer, UniqueValueRenderer, and TemporalRenderer. We'll examine each type of renderer.

The rendering process will be the same regardless of the type of renderer you use. You first need to create an instance of the renderer, define the symbology for the renderer, and finally, apply the renderer to the GraphicsLayer. In the event that you don't want to apply the same renderer to all graphics in the GraphicsLayer you will need to create multiple GraphicsLayer objects to organize your graphics. This rendering process is illustrated below.

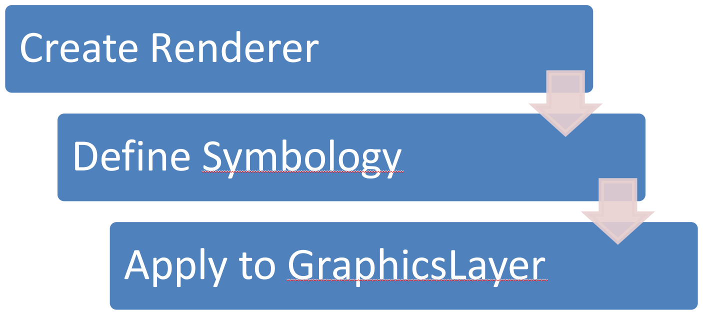
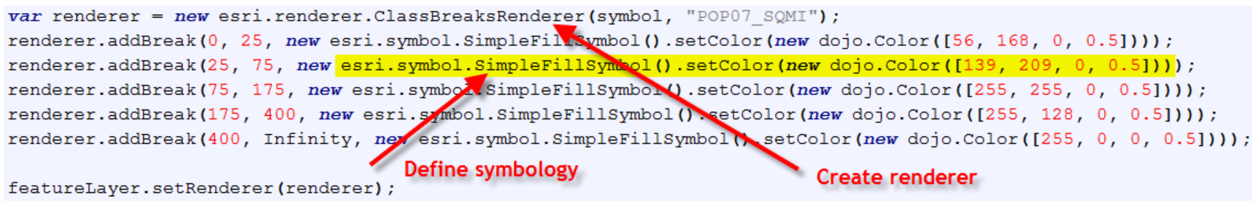


Figure 39: Rendering Process

The code example below shows the basic programmatic structure for creating and applying a renderer to a GraphicsLayer.

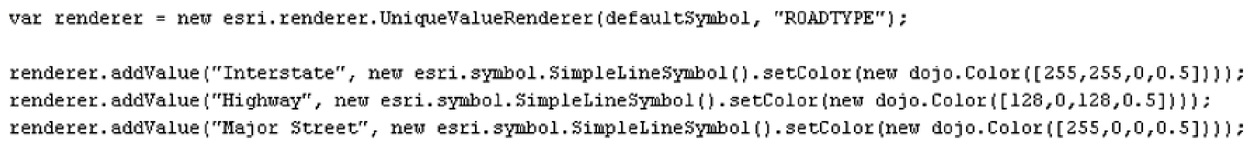


```
var renderer = new esri.renderer.ClassBreaksRenderer(symbol, "POP07_SQMI");
renderer.addBreak(0, 25, new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([56, 168, 0, 0.5])));
renderer.addBreak(25, 75, new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([139, 209, 0, 0.5])));
renderer.addBreak(75, 175, new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 255, 0, 0.5])));
renderer.addBreak(175, 400, new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 128, 0, 0.5])));
renderer.addBreak(400, Infinity, new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 0, 0, 0.5])));

featureLayer.setRenderer(renderer);
```

The simplest type of renderer is the SimpleRenderer which simply applies the same symbol for all graphics.

A UniqueValueRenderer can be used to symbolize graphics based on a matching attribute which is typically a field containing string data. For example, if you have a roads feature class you might want to symbolize each feature based on road type. Each road type would have a different symbol. The code example on this slide shows how you would programmatically create a UniqueValueRenderer and add values and symbols to the structure.



```
var renderer = new esri.renderer.UniqueValueRenderer(defaultSymbol, "ROADTYPE");

renderer.addValue("Interstate", new esri.symbol.SimpleLineSymbol().setColor(new dojo.Color([255,255,0,0.5])));
renderer.addValue("Highway", new esri.symbol.SimpleLineSymbol().setColor(new dojo.Color([128,0,128,0.5])));
renderer.addValue("Major Street", new esri.symbol.SimpleLineSymbol().setColor(new dojo.Color([255,0,0,0.5])));
```

A ClassBreaksRenderer works with data stored as a numeric attribute. Each graphic will be symbolized according to the value for that particular attribute in accordance with breaks in the data. The breaks define the values at which the symbol will change. For example, with a Parcel Feature Class you might want to symbolize parcels based on values found in the PropertyValue field. You'd first want to create a new instance of ClassBreaksRenderer and then define the breaks for the data. The "Infinity" and "-Infinity" values can be used as the lower and upper boundaries for your data if needed as seen in the code example below where we use the "Infinity" keyword to signify a class break of any values greater than 250,000.

```
var renderer = new esri.renderer.ClassBreaksRenderer(symbol, "PROPERTYVALUE");

renderer.addBreak(0,50000,new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 0, 0,0.5])));
renderer.addBreak(50001,100000,new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 255, 0,0.5])));
renderer.addBreak(100001,250000,new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([0,255,0,0.5])));
renderer.addBreak(250001,Infinity,new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([0,255,0,0.5])));
```

Assigning Attributes to Graphics

The attributes of a graphic are the name value pairs that describe that object. In many cases graphics are generated as the result of a task operation such as QueryTask. In these cases a geometry object is composed of both geometry and attributes and you'd then need to symbolize each graphic accordingly. The field attributes associated with the layer is the attributes for the graphic. The attributes returned can be limited through the use of the 'outFields' property. If your graphics are being created programmatically then you'll need to assign the attributes in your code using the Graphic.setAttributes() method seen in the code example below.

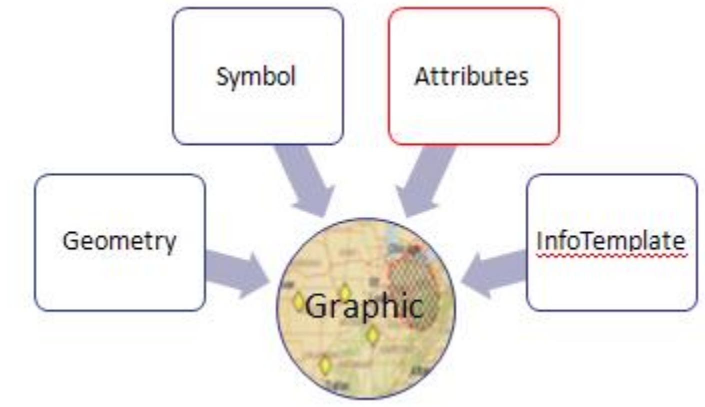


Figure 40: Graphic Attributes

```
graphic.setAttributes( {"XCoord":evt.mapPoint.x, "YCoord":evt.mapPoint.y, "Plant":"Mesa Mint"});
```

Displaying Graphic Attributes in an InfoTemplate

In addition to attributes a graphic can also have an InfoTemplate that defines how the attribute data is displayed in a pop-up window, also known as an Info Window. A point attribute variable is defined in the code example you see below and contains name (key) value pairs. In this particular case we have keys that include address, city, and state. Each of these names or keys has a value. This variable is the third parameter in the constructor for a new point graphic. An InfoTemplate defines the format of the Info Window that appears and contains a title and an optional content template string. Content for the template is added with the InfoTemplate.content property as seen in the code examples below.

Content for an InfoTemplate can be defined as a string, HTML, or a placeholder. The most basic type of content is simply a string of characters that is displayed as plain text in the InfoWindow.

```
infoWinTemplate.content = "This is the content for the InfoWindow";
```

An InfoTemplate can also be defined using HTML tags. This is a much more flexible way of adding content to your InfoWindow because it enables you to embed links, images, video, audio, and pretty much any other type of HTML tag.

```
infoWinTemplate.content = "This is some content with <b>HTML</b> markup!
        <p>This is a paragraph
        </p><p>Another Paragraph</p>";
```

Finally, you can also use placeholders to define content. Placeholders are created using the format ${FIELD_NAME} to define an attribute value that should be displayed in a particular placeholder. The attribute defined inside the placeholder must be the exact column name from a layer.

```
infoWinTemplate.content = "<b>Name</b>: ${NAME}" +
                     "<br><b>Address</b>: ${ADDRESS}" +
                     "<br><b>Social Security</b>: ${SSN}" +
                     "<br><b>Birthday</b>: ${DOB}";
```

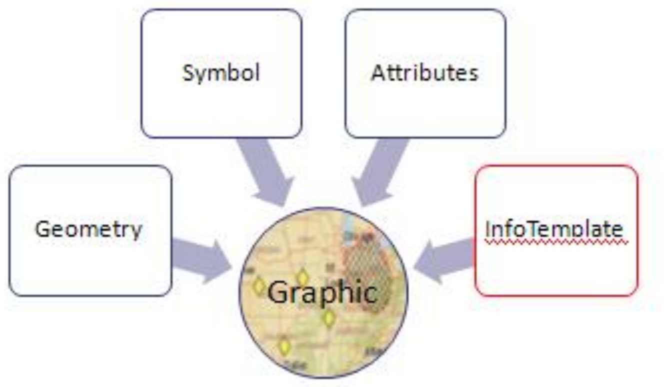


Figure 41: Graphic InfoTemplate

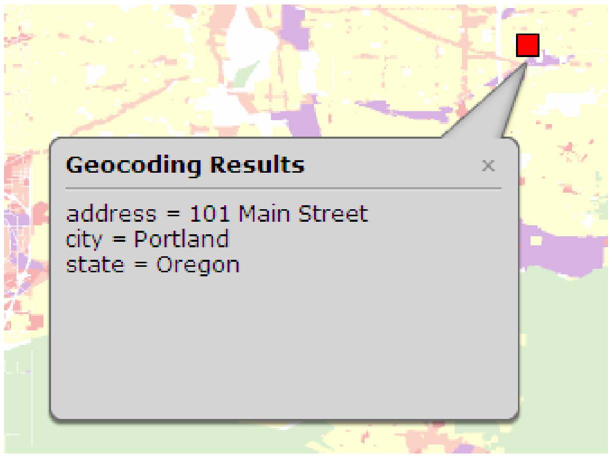
```
var point = new esri.geometry.Point(Number(theX), Number(theY), msr);

var markerSymbol = new esri.symbol.SimpleMarkerSymbol();
markerSymbol.setStyle(esri.symbol.SimpleMarkerSymbol.STYLE_SQUARE);
markerSymbol.setSize(12);
markerSymbol.setColor(new dojo.Color([255,0,0]));

var ptAttributes = {address:"101 Main Street", city:"Portland", state:"Oregon"};
var ptInfoTemplate = new esri.InfoTemplate("Geocoding Results");

var ptGraphic = new esri.Graphic(point, markerSymbol, ptAttributes).setInfoTemplate(ptInfoTemplate);

map.graphics.add(ptGraphic);
```


Geocoding Results

address = 101 Main Street
city = Portland
state = Oregon

Figure 42: InfoTemplate Example

Info Windows

An infoTemplate, as you learned in the last section, is used to define how content will be displayed in an InfoWindow. Both the title and content of the window are defined with an InfoTemplate. Now we’ll turn our attention to the actual InfoWindow. In addition to displaying the attributes of a feature and other text, an InfoWindow can contain charts, pictures, video, and pretty much anything else that can be defined with HTML tags. A single InfoWindow is associated with a Map object and contains both a title as well as content.

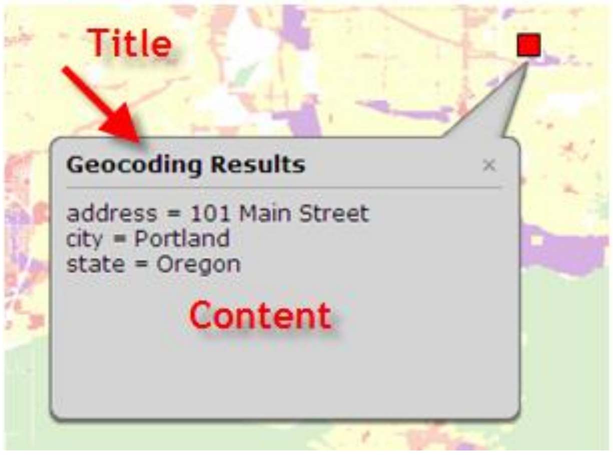


Figure 43: Setting the Title and Content

An InfoWindow is most commonly associated with a graphic and by default is displayed when clicked. However, an InfoWindow is associated with the Map so it can be displayed is response to other events such as the completion of a query or some other action not associated with a graphic.

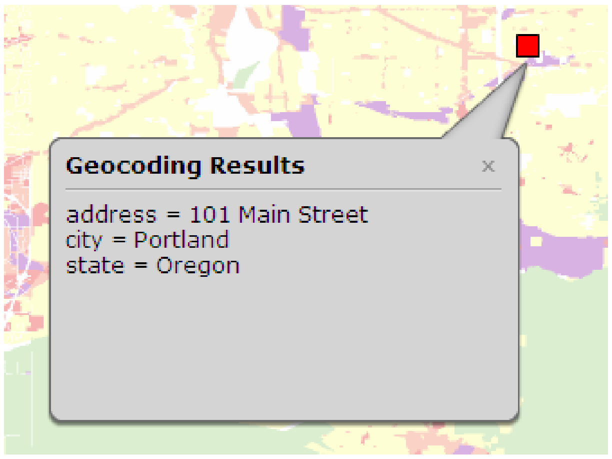
Displaying the InfoWindow

The InfoWindow is displayed by calling the InfoWindow.show() method as seen in the code example below. Note that the InfoWindow is a property of the Map object and is accessed with dot notation like any other property. Once you've obtained an instance of the InfoWindow you can set the title, content, and show the window.

```
map.infoWindow.setTitle("Coordinates");
map.infoWindow.setContent("lat/lon : " + evt.mapPoint.y + ", " + evt.mapPoint.x);
map.infoWindow.show(evt.screenPoint,map.getInfoWindowAnchor(evt.screenPoint));
```

Customizing the InfoWindow

By default the InfoWindow will appear as we've seen in the figure below with a light grey background, title in bold at the top of the window, content below, and an 'X' in the upper right hand corner that closes the button. However, there may be times when you'd like to alter the look of the InfoWindow. This can be accomplished by adding Dojo dijits, altering the CSS for the window, using an image as the background for the window, or the creation of a custom window.



address = 101 Main Street
city = Portland
state = Oregon

Figure 44: InfoWindow

The Dojo TabContainer dijit is often used to organize several pieces of information inside a single InfoWindow. Other Dojo dijits can also be used to enhance the information that is displayed. The figure below combines a TabContainer dijit along with a Dojo chart to create an attractive InfoWindow.

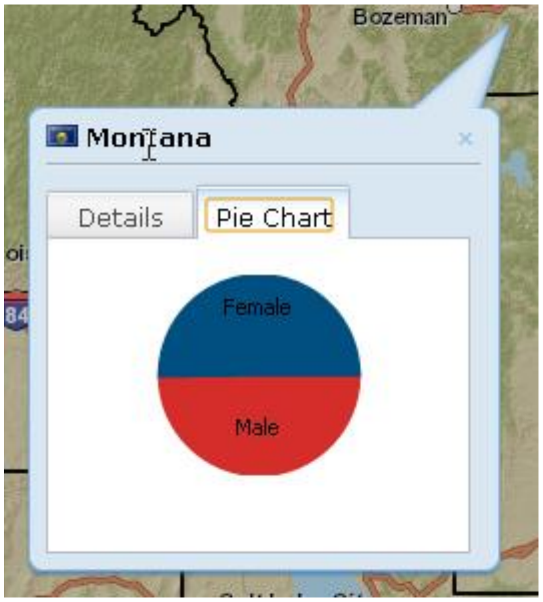
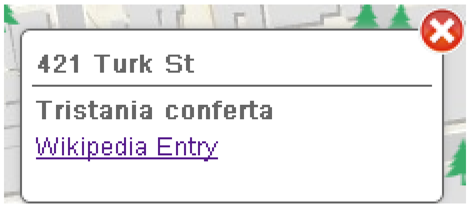


Figure 45: Adding a Chart to an InfoWindow

The InfoWindow can also be simplified by loading the InfoWindowLite resource using dojo.require. This will create a simpler InfoWindow like the one you see below.



421 Turk St
Tristania conferta
Wikipedia Entry

Figure 46: Simplified InfoWindow

Using CSS the InfoWindow can then be styled to an even greater degree. The CSS code below would produce an InfoWindow with a dark border color and a white-green gradient applied to the background.

```
#map_infowindow.simpleInfoWindow {
    border: 2px solid #455268;
    background-color: #dfe5d7;
    background: -moz-linear-gradient(top, #fcfff4 0%, #dfe5d7 40%, #b3bead 100%);
    background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#fcfff4), color-stop(40%,#dfe5d7), color-stop(100%,#b3bead));
}
```

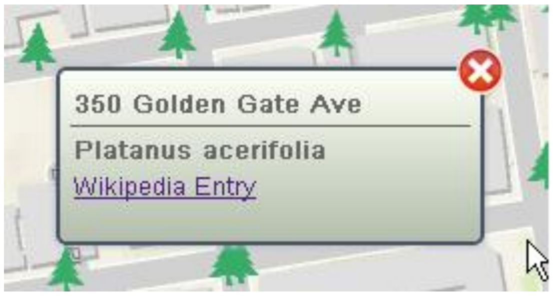


Figure 47: Styling an InfoWindow

In addition to using CSS to style an InfoWindow you can also create your own custom InfoWindow using a background image. This requires that a .png format image be created for the background, close button and pointer using some sort of photo-editing program. ESRI also provides a .png file (http://help.arcgis.com/en/webapi/javascript/arcgis/help/jshelp/images/claroinfowindow.png) that can be downloaded and modified.

For even more advanced customization of InfoWindows you can also extend the InfoWindowBase class. However, this will require that you implement several required methods, properties, and events to ensure that the custom InfoWindow provides a minimum level of functionality.

Popups

A new type of information window, introduced at version 2.3 of the API, is the Popup window. The Popup and PopupTemplate classes were added to the API that gives custom applications the same experience as pop-ups created using ArcGIS.com. The new Popup window has a more modern styling than the InfoWindow and also provides additional functionality. In addition to displaying a title and content, Popup window also provides navigation through selected features, zooming to the selected feature, highlighting of a selected feature, and the ability to maximize the window. You can see an example of these new functions in the figure below.

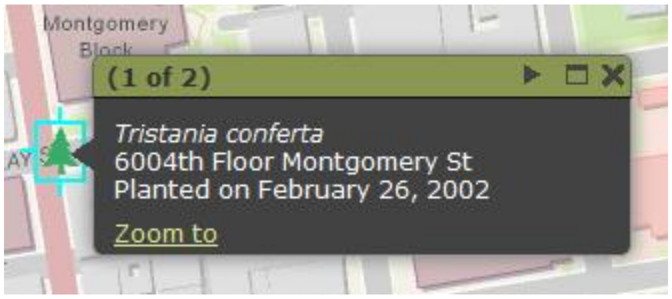


Figure 48: Popup Window

To replace the default InfoWindow with a PopUp window in an application several lines of code need to be added to the application. First, add the popup stylesheet to the application as seen below.

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

Import the popup class module.

```
dojo.require("esri.dijit.Popup");
```

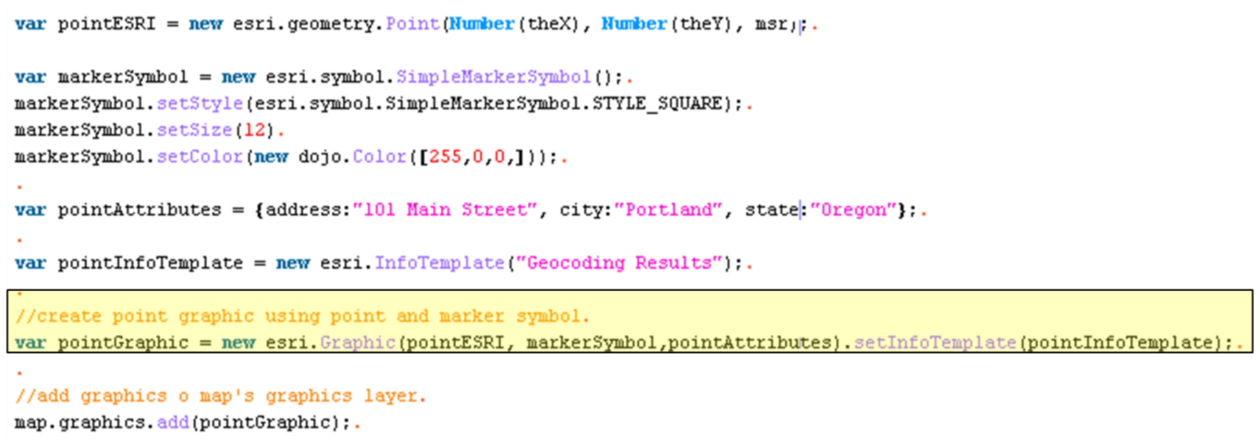
Create a new instance of the Popup class. Various options can be defined to change the look and functionality of the Popup class. These options include the feature highlight symbols for selected features, x and y offsets for positioning the popup, a zoom factor, and others.

```
var popup = new esri.dijit.Popup({
  offsetX:10,
  offsetY:10,
  zoomFactor:2
}, dojo.create("div"));
```

Finally, create the Map and pass in an instance of the popup as an option.

```
var map = new esri.Map("mapDiv",{ infoWindow:popup });
```

Creating the Graphic
Once the optional geometry, symbology, attributes, and info template have been define a new Graphic object can be created with these parameters used as input to the constructor for the Graphic. Notice in the code example below that we create variables for the geometry (pointESRI), symbology (markerSymbol), point attributes (pointAttributes), and InfoTemplate (pointInfoTemplate) and then apply these variables as input to the constructor for our new graphic called pointGraphic. Finally, this graphic is added to the GraphicsLayer.



```
var pointESRI = new esri.geometry.Point(Number(theX), Number(theY), msr);

var markerSymbol = new esri.symbol.SimpleMarkerSymbol();
markerSymbol.setStyle(esri.symbol.SimpleMarkerSymbol.STYLE_SQUARE);
markerSymbol.setSize(12)
markerSymbol.setColor(new dojo.Color([255,0,0,]));

var pointAttributes = {address:"101 Main Street", city:"Portland", state:"Oregon"};

var pointInfoTemplate = new esri.InfoTemplate("Geocoding Results");

//create point graphic using point and marker symbol.
var pointGraphic = new esri.Graphic(pointESRI, markerSymbol,pointAttributes).setInfoTemplate(pointInfoTemplate);

//add graphics o map's graphics layer.
map.graphics.add(pointGraphic);
```

Adding Graphics to the GraphicsLayer

Before any graphics can be displayed on the map they must added to the GraphicsLayer. Each map has a GraphicsLayer which contains an array of graphics that is initially empty until you add the graphics. This layer can contain any type of graphic object meaning that you can mix in points, lines, and polygons at the same time. Graphics are added to the layer through the add( ) method and can also be removed individually through the remove( ) method. In the event that you need to remove all graphics simultaneously the clear( ) method can be used. GraphicsLayer also has a number of events that can be registered including onClick, onMouseDown, and others.

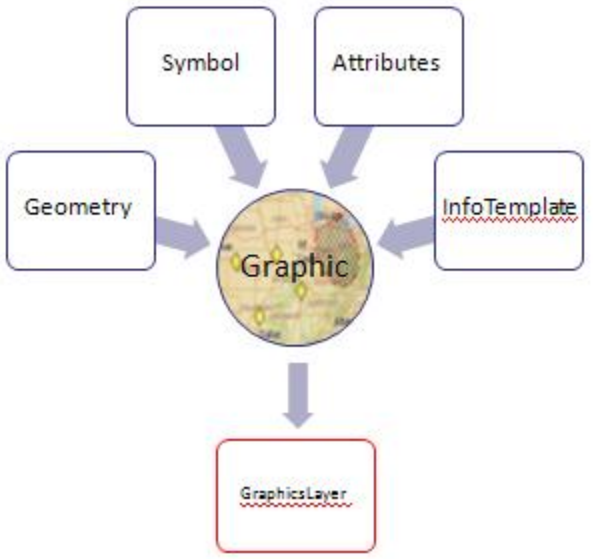
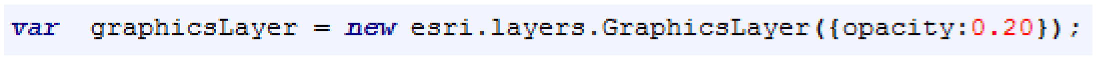


Figure 49: GraphicLayer

Multiple Graphics Layers

In addition to the default GraphicsLayer associated with the Map object, multiple graphics layers are supported by the API making it much easier to organize different types of graphics. These graphic layers can easily be removed or added as needed. For example, you can put polygon graphics representing counties in one graphics layer and point graphics representing traffic incidents in another graphics layer. To create a new GraphicsLayer use the GraphicsLayer constructor along with any optional parameters.



```
var  graphicsLayer = new esri.layers.GraphicsLayer({opacity:0.20});
```