Chapter No. 6
"Performing Spatial and Attribute Queries"
In this package, you will find:

A Biography of the author of the book
A preview chapter from the book, Chapter NO.6 "Performing Spatial and Attribute Queries"
A synopsis of the book’s content
Information on where to buy this book

About the Author

**Eric Pimpler** is the founder and owner of GeoSpatial Training Services (geospatialtraining.com) and has over 20 years of experience in implementing and teaching GIS solutions using Esri, Google Earth/Maps, and open source technology. Currently, he focuses on ArcGIS scripting with Python and the development of custom ArcGIS Server web and mobile applications using JavaScript. He is the author of *Programming ArcGIS 10.1 with Python Cookbook*.

Eric has a bachelor's degree in Geography from Texas A&M University and a master's degree in Applied Geography with a concentration in GIS from Texas State University.

For More Information:

Building Web and Mobile ArcGIS Server Applications with JavaScript

ArcGIS Server is the predominant platform used to develop GIS applications for the Web. There are a number of programming languages you can use to develop applications with ArcGIS Server, including JavaScript, Flex, and Silverlight. JavaScript has become the preferred language for developing applications on this platform, since it can be used for both web and mobile applications and doesn't require the installation of a plugin for the application to be used in a browser. Flex and Silverlight both fall short as languages for mobile development and both require the use of a plugin for the application to run in a browser.

This book will teach you how to build web-based GIS applications using the ArcGIS API for JavaScript. Using a practical, hands-on style of learning, you will learn how to develop fully functional applications with ArcGIS Server and develop a skill set that is in high demand.

You will learn how to create maps and add geographic layers from a variety of sources, including tiled and dynamic map services. In addition, you'll learn how to add graphics to the map and stream geographic features to the browser using FeatureLayer. Most applications also include specific functionalities implemented by ArcGIS Server as tasks. You'll learn how to use the various tasks provided by ArcGIS Server, including queries, identification of features, finding features by attributes, geoprocessing tasks, and more. Finally, you'll learn just how easy it is to develop mobile applications with the ArcGIS API for JavaScript.

For More Information:
What This Book Covers

Chapter 1, Introduction to HTML, CSS, and JavaScript, covers fundamental HTML, CSS, and JavaScript concepts before getting started with developing GIS applications with the ArcGIS API for JavaScript.

Chapter 2, Creating Maps and Adding Layers, teaches you how to create a map and add layers to the map. 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. The Map class 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, and feature. Readers will learn more about each of these layer types in this chapter.

Chapter 3, Adding Graphics to the Map, teaches the reader how to display temporary points, lines, and polygons in GraphicsLayer on the map. GraphicsLayer is a separate layer that always resides on top of other layers and stores all the graphics associated with the map.

Chapter 4, The Feature Layer, offers additional capabilities, apart from inheriting from GraphicsLayer, such as the ability to perform queries and selections. Feature layers are also used for online editing of features. Feature layers differ from tiled and dynamic map service layers, because feature layers bring geometry information to the client computer to be drawn and stored by the web browser. Feature layers potentially cut down on round trips to the server. A client can request the features it needs, and perform selections and queries on those features without having to request more information from the server.

Chapter 5, Using Widgets and Toolbars, covers out-of-the-box widgets that you can drop into your application for enhanced productivity. The BasemapGallery, Bookmarks, Print, Geocoding, Legend, Measurement, Scalebar, Gauge, and Overview map widgets are included. In addition, the ArcGIS API for JavaScript also includes helper classes for adding various toolbars to your applications, including navigation and drawing toolbars.

Chapter 6, Performing Spatial and Attribute Queries, covers the ArcGIS Server Query Task, which allows you to perform attribute and spatial queries against data layers in a map service that have been exposed. You can also combine these query types to perform a combination attribute and spatial query.

For More Information:
Chapter 7, Identifying and Finding Features, covers two common operations found in any GIS application. These operations require that the user click a feature on the map in the case of identification, or perform a query in the case of finding features. In either case, information about particular features is returned. In this chapter, the reader will learn how to use the IdentifyTask and FindTask objects to obtain information about features.

Chapter 8, Turning Addresses into Points and Points into Addresses, covers the use of the Locator task to perform geocoding and reverse geocoding. Geocoding is the process of assigning a coordinate to an address, while reverse geocoding assigns an address to a coordinate.

Chapter 9, Network Analyst Tasks, allows you to perform analyses on street networks, such as finding the best route from one address to another, finding the closest school, identifying a service area around a location, or responding to a set of orders with a fleet of service vehicles.

Chapter 10, Geoprocessing Tasks, allows you to execute custom models built in ArcGIS Desktop using ModelBuilder. Models are run in an automated fashion from either a desktop environment or via a centralized server accessed through a web application. Any tool found in ArcToolbox, whether that be a tool for your ArcGIS license level or a custom tool that you've built, can be used in a model and chained together with other tools. Once constructed, these models can be run on a centralized server and accessed via web applications. In this chapter, we will examine how you can access these geoprocessing tasks through the ArcGIS API for JavaScript.

Chapter 11, Integration with ArcGIS Online, details how you can use the ArcGIS API for JavaScript to access the data and maps created with ArcGIS.com. The website ArcGIS.com is for working with maps and other types of geographic information. On this site, you will find applications for building and sharing maps. You will also find useful basemaps, data, applications, and tools that you can view and use, plus communities you can join. For application developers, the really exciting news is that you can integrate ArcGIS.com content into your custom developed applications using the ArcGIS API for JavaScript. In this chapter, you will explore how ArcGIS.com maps can be added to your applications.

For More Information:
Chapter 12, Creating Mobile Applications, details how you can build mobile GIS applications using the ArcGIS API for JavaScript. ArcGIS Server support is currently provided for iOS, Android, and BlackBerry operating systems. The API is integrated with dojox/mobile. In this chapter, you'll learn about the compact build of the API that makes web mapping applications possible through web-kit browsers as well as the built-in gesture support.

Appendix, Application Design with ArcGIS Templates and Dojo, covers one of the most difficult tasks for many web developers which is designing and creating the user interface. The ArcGIS API for JavaScript and Dojo greatly simplifies this task. Dojo's layout dijits provide a simple, efficient way to create application layouts, and Esri has provided a number of sample application layouts and templates that you can use to get up and running quickly. In this appendix, the reader will learn techniques to design an application quickly.

For More Information:
Performing Spatial and Attribute Queries

Using ArcGIS Server Query tasks, you can perform attribute and spatial queries against data layers in a map service that has been exposed. You can also combine these query types to perform combined attribute and spatial queries. For example, you might need to find all land parcels with an appraised value greater than $100,000 and that intersect the 100-year floodplain. This would be an example of a combined query that includes both spatial and attribute components. In this chapter, you will learn how to perform attribute and spatial queries using the Query, QueryTask, and FeatureSet objects in the ArcGIS API for JavaScript.

We will cover the following topics in this chapter:

- Introducing tasks in ArcGIS Server
- An overview of attribute and spatial queries
- The Query object
- Executing the query with QueryTask
- Time to practice with spatial queries

Introducing tasks in ArcGIS Server

In the next few chapters of the book, we will discuss the many types of tasks that can be performed with the ArcGIS API for JavaScript. Tasks give you the ability to perform spatial and attribute queries, find features based on text searches, geocode addresses, identify features, and perform various geometry operations including buffering and distance measurements. All tasks are accessed through the esri/tasks resource.

For More Information:
Performing Spatial and Attribute Queries

All tasks in the ArcGIS API for JavaScript follow the same pattern. This pattern is easily recognizable once you've worked with one or more tasks for any length of time. An input object is used to supply input parameters to the task. Using these input parameters, the task performs its specific function and then an output object is returned containing the results of the task. The following diagram illustrates how each task accepts an input parameter object and returns an output object that can be used in your application.

![Diagram showing input, task, and output objects]

An overview of attribute and spatial queries

As you'll see with other tasks, queries are performed using a sequence of objects that typically include the input to the task, execution of the task, and a result set returned from the task. The input parameters for an attribute or spatial query are stored in a Query object which contains various parameters that can be set for the query. The QueryTask object executes the task using the input provided in the Query object, and a result set is returned in the form of a FeatureSet object, which contains an array of Graphic features that you can then plot on the map.

The Query object, used as input to a QueryTask, is defined by properties that include geometry, where, and text. The geometry property is used to input a geometry that will be used in a spatial query and will be a point, line, or polygon geometry. The where property is used to define an attribute query, while the text property is used to perform a where clause containing a like operator. The Query object can also contain a number of optional properties including the ability to define the fields that will be returned as a result of the query, the output spatial reference for the returned geometry, and the actual geometry of the features that meet the query conditions.

![Diagram showing query, query task, and feature set]

For More Information:
The preceding diagram defines the object sequence you will use when creating attribute and spatial queries.

**The Query object**

In order for the `QueryTask` object to execute a query against a layer in a map service, it needs input parameters that are defined with a `Query` object. Input parameters define whether the query will be spatial, attribute, or a combination of the two. Attribute queries can be defined by either the `where` or `text` properties. These properties are used to define a SQL attribute query. We’ll look at the difference between `Query.where` and `Query.text` in a later section.

Spatial queries require that you set the `Query.geometry` property to define the input geometric shape to be used in a spatial query.

A new instance of the `Query` object can be created through the use of a constructor as seen in the following code example:

```javascript
var query = new Query();
```

**Defining the query properties**

As I mentioned in the introduction to this section, you can set various parameters on the `Query` object. It is required that you either define the properties for an attribute query (`Query.where` or `Query.text`) or the `Query.geometry` property for a spatial query. You can also use a combination of attribute and spatial query properties.

**Attribute queries**

The `Query` object provides two properties that can be used in an attribute query: `Query.where` and `Query.text`. In the following code example, I’m setting the `Query.where` property so that only records where the `STATE_NAME` field equal to ‘Texas’ are returned. This is just a standard SQL query. Notice that I’ve enclosed the word Texas with quotes. When performing an attribute query against a text column, you need to enclose the text being evaluated with either single or double quotes. This isn’t needed if you are performing an attribute query against a column containing other data types such as numbers or Booleans:

```javascript
query.where = "STATE_NAME = 'Texas'";
```
Performing Spatial and Attribute Queries

You can also use the `Query.text` property to perform an attribute query. This is a shorthand way for creating a `where` clause using `like`. The field used in the query is the display field for the layer defined in the map document. You can determine the display field for a layer in the services directory. This is illustrated in the following screenshot where `ZONING_NAME` is the display field. It is this display field that is queried using the `Query.text` property.

```javascript
//Query.text uses the Display Name for the layer
query.text = stateName;
```

In the following code example, we use `query.text` to perform an attribute query that returns all fields where the state name is entered by the user in a form field on the web page:

```javascript
query = new Query();
query.returnGeometry = false;
query.outFields = ['*'];
query.text = dom.byId("stateName").value;
queryTask.execute(query, showResults);
```
Spatial queries

To perform a spatial query against a layer, you'll need to pass in a valid geometry object to be used in the spatial filter along with a spatial relationship. Valid geometries include instances of Extent, Point, Polyline, and Polygon. The spatial relationship is set through the \texttt{Query.spatialRelationship} property and is applied during the query. The spatial relationship is defined through the use of one of the following constant values: \texttt{SPATIAL\_REL\_INTERSECTS}, \texttt{SPATIAL\_REL\_CONTAINS}, \texttt{SPATIAL\_REL\_CROSSES}, \texttt{SPATIAL\_REL\_ENVELOPE\_INTERSECTS}, \texttt{SPATIAL\_REL\_OVERLAPS}, \texttt{SPATIAL\_REL\_TOUCHES}, \texttt{SPATIAL\_REL\_WITHIN}, and \texttt{SPATIAL\_REL\_RELATION}. The table in the following screenshot describes each of the spatial relationship values:

<table>
<thead>
<tr>
<th>Spatial Relationship</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SPATIAL_REL_CONTAINS}</td>
<td>Part or all of a feature from feature class 1 is contained within a feature from feature class 2.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_CROSSES}</td>
<td>The feature from feature class 1 crosses a feature from feature class 2.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_ENVELOPE_INTERSECTS}</td>
<td>The envelope of feature class 1 intersects with the envelope of feature class 2.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_INDEX_INTERSECTS}</td>
<td>The envelope of the query feature class intersects the index entry for the target feature class.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_INTERSECTS}</td>
<td>Part of a feature from feature class 1 is contained in a feature from feature class 2.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_OVERLAPS}</td>
<td>Features from feature class 1 overlap feature in feature class 2.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_RELATION}</td>
<td>Allows specification of any relationship defined using Shape Comparison Language.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_TOUCHES}</td>
<td>The feature from feature class 1 touches the border of a feature from feature class 2.</td>
</tr>
<tr>
<td>\texttt{SPATIAL_REL_WITHIN}</td>
<td>The feature from feature class 1 is completely enclosed by the feature from feature class 2.</td>
</tr>
</tbody>
</table>

The following code example sets a \texttt{Point} object as the geometry passed into the spatial filter in addition to setting the spatial relationship:

```javascript
query.geometry = evt.mapPoint;
query.spatialRelationship = \texttt{SPATIAL\_REL\_INTERSECTS};
```

Limiting the fields returned

For performance reasons, you should limit the fields that are returned in the \texttt{FeatureSet} object to only those fields that are needed in your application. Every column of information attached to the \texttt{FeatureSet} object is additional data that must be passed from the server to the browser, which can cause your application to perform slower than it should. To limit the returned fields, you assign an array containing a list of fields that should be returned to the \texttt{Query.outFields} property as seen in the following code example. To return all fields you can use \texttt{outFields = ['*']}. 

For More Information:

Performing Spatial and Attribute Queries

In addition, you can control the return of the geometry for each feature through the Query.returnGeometry property. By default, the geometry will be returned; however, in some cases, your application may not need the geometry. For example, if you need to populate a table with the attribute information from a layer, you don't necessarily need the geometry. In this case, you can set Query.returnGeometry = false:

```javascript
query.outFields =
    ["NAME", "POP2000", "POP2007", "POP00_SQMI", "POP07_SQMI"];
query.returnGeometry = false;
```

Executing the query with QueryTask

Once you've defined the input properties in a Query object, you can use QueryTask to execute the query. Before the query can be executed, you must first create an instance of the QueryTask object. A QueryTask object is created by passing a URL to the layer against which the query will be executed inside the constructor for the object. The following code example shows how a QueryTask object is created. Notice that it includes an index number at the end of the URL that references a specific layer in the map service to be queried:

```javascript
myQueryTask = new QueryTask("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Demographics/ESRI_CENSUS_USA/MapServer/5");
```

After creation, the QueryTask object can be used to execute a query against the layer with an input Query object using the QueryTask.execute() method. QueryTask.execute() accepts three parameters including an input Query object along with success and error callback functions. The syntax for QueryTask.execute() is provided in the following code. The input Query object is passed as the first parameter:

```javascript
QueryTask.execute(parameters, callback?, errback?)
```

Assuming that the query executes without any error, the success callback function will be called and a FeatureSet object is passed into the function. If an error occurs during the execution of the query, then an error callback function is executed. Both the success and error callback functions are optional; however, you should always define functions to handle both cases.

At this point, you may be wondering about these callback and errback functions. Most tasks in ArcGIS Server return an instance of dojo/Deferred. A Deferred object is a class that is used as the foundation for managing asynchronous threads in Dojo. Tasks in ArcGIS Server can be either synchronous or asynchronous.
Asynchronous and synchronous define how the client (the application using the task) interacts with the server and gets the result from the task. When a service is set to synchronous, the client waits for the task to complete. Typically, a synchronous task executes quickly (several seconds or lesser). An asynchronous task typically takes longer to execute, and the client doesn't wait for the task to complete. The end user is free to continue using the application while the task executes. When a task completes on the server, it calls the callback function and passes the results into this function where they can then be used in some way. They are often displayed on the map.

Let's take a look at a more complete code example. In the following code example, notice that we first create a new variable called myQueryTask, which points to layer 6 (the index numbers are 0 based) in the ESRI_CENSUS_USA map service. We then create the Query object containing the input properties of the query and finally, we use the execute() method on QueryTask to perform the query. The execute() method returns a FeatureSet object that contains the results of the query and these features are processed through a callback function called showResults, which is specified in the execute() method. If an error occurs during the execution of the task, the errorCallback() function will be called:

```javascript
myQueryTask = new QueryTask("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Demographics/ESRI_CENSUS_USA/MapServer/5");
//build query filter
myQuery = new Query();
myQuery.returnGeometry = false;
myQuery.outFields = ["STATE_NAME", "POP2007", "MALES", "FEMALES"];  
myQuery.text = 'Oregon';
//execute query
myQueryTask.execute(myQuery, showResults, errorCallback);

function showResults(fs) {
    //do something with the results
    //they are returned as a featureset object
}

function errorCallback() {
    alert("An error occurred during task execution");
}
```

For More Information:
Performing Spatial and Attribute Queries

Getting query results

As I mentioned earlier, the results of a query are stored in a `FeatureSet` object that includes an array of graphics, which you can then plot on your map if you wish.

Each feature (graphic) in the array can contain geometry, attributes, symbology, and an `InfoTemplate` as described in Chapter 3, Adding Graphics to the Map. Typically, these features are plotted on the map as graphics. The following code example shows a callback function that is executed when a query has completed execution. A `FeatureSet` object is passed into the callback function and the graphics are drawn on the map:

```javascript
function addPolysToMap(featureSet) {
    var features = featureSet.features;
    var feature;
    for (i=0, il=features.length; i<il; i++) {
        feature = features[i];
        attributes = feature.attributes;
        pop = attributes.POP90_SQMI;
        map.graphics.add(features[i].setSymbol(sym));
    }
}
```

Time to practice with spatial queries

In this exercise, you will learn how to perform spatial queries using the `Query`, `QueryTask`, and `FeatureSet` objects in the ArcGIS API for JavaScript. Using a Zoning layer from the City of Portland, you will query parcel records and display the results on a map.

Perform the following steps to complete the exercise:

2. Remove the JavaScript content from the `<script>` tag that I have highlighted in the following code snippet:

   ```
   dojo.require("esri.map");
   
   function init(){
   var map = new esri.Map("mapDiv", {
   center: [-56.049, 38.485],
   zoom: 3,
   }
   ```

For More Information:

```javascript
basemap: "streets"
});
}
dojo.ready(init);
</script>

3. Create the variables that you'll use in the application.

```javascript
<
var map, query, queryTask;
var symbol, infoTemplate;
</script>

4. Add the `require()` function as seen in the following highlighted code:

```javascript
<
require(["esri/map", "esri/tasks/query", "esri/tasks/QueryTask",
    "esri/tasks/FeatureSet",
    "esri/symbols/SimpleFillSymbol",
    "esri/symbols/SimpleLineSymbol", "esri/InfoTemplate",
    "dojo/_base/Color", "dojo/on", "dojo/domReady!"
], function(Map, Query, QueryTask, FeatureSet,
    SimpleFillSymbol, SimpleLineSymbol, InfoTemplate, Color,
    on) {

});
</script>

5. Inside the `require()` function, create the `Map` object that you'll use in the application. The map will be centered on the Louisville, KY, area:

```javascript
require(["esri/map", "esri/tasks/query", "esri/tasks/QueryTask",
    "esri/tasks/FeatureSet",
    "esri/symbols/SimpleFillSymbol",
    "esri/symbols/SimpleLineSymbol", "esri/InfoTemplate",
    "dojo/_base/Color", "dojo/on", "dojo/domReady!"
], function(Map, Query, QueryTask, FeatureSet,
    SimpleFillSymbol, SimpleLineSymbol, InfoTemplate, Color,
    on) {

});
```
Performing Spatial and Attribute Queries

```javascript
map = new Map("mapDiv",{
    basemap: "streets",
    center: [-85.748, 38.249], //long, lat
    zoom: 13
});

});

6. Create the symbol that will be used to display the results of the query:
require(["esri/map", "esri/tasks/query", "esri/tasks/QueryTask",
    "esri/tasks/FeatureSet",
    "esri/symbols/SimpleFillSymbol",
    "esri/symbols/SimpleLineSymbol", "esri/InfoTemplate",
    "dojo/_base/Color", "dojo/on", "dojo/domReady!"]
}, function(Map, Query, QueryTask, FeatureSet,
    SimpleFillSymbol, SimpleLineSymbol, InfoTemplate,
    Color, on) {
    map = new Map("map",{
        basemap: "streets",
        center: [-85.748, 38.249], //long, lat
        zoom: 13
    });

    symbol = new SimpleFillSymbol(SimpleFillSymbol.STYLE_SOLID,
        new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID, new
        Color([111, 0, 255]), 2), new Color([255, 255, 0, 0.25]));
    infoTemplate = new InfoTemplate("${OBJECTID}", "${*}"");
});

7. Now, inside the require() function, we are going to initialize the
queryTask variable and then register the QueryTask.complete event.
Add the following highlighted lines of code:
require(["esri/map", "esri/tasks/query", "esri/tasks/QueryTask",
    "esri/tasks/FeatureSet",
    "esri/symbols/SimpleFillSymbol",
    "esri/symbols/SimpleLineSymbol", "esri/InfoTemplate",
    "dojo/_base/Color", "dojo/on", "dojo/domReady!"]
}, function(Map, Query, QueryTask, FeatureSet,
    SimpleFillSymbol, SimpleLineSymbol, InfoTemplate,
    Color, on) {
    queryTask = new QueryTask();
    queryTask.on("complete", function(event) {
        // handle the query response
    });
});
```

For More Information:
map = new Map("mapDiv",{
    basemap: "streets",
    center:[-85.748, 38.249], //long, lat
    zoom: 13
});

symbol = new SimpleFillSymbol(SimpleFillSymbol.STYLE_SOLID,
    new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID, new
    Color([111, 0, 255]), 2), new
    Color([255,255,0,0.25]));

infoTemplate = new InfoTemplate("${OBJECTID}", "${*}");

queryTask = new QueryTask("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Louisville/LOJIC_LandRecords_Louisville/MapServer/2");

queryTask.on("complete", addToMap);

The constructor for QueryTask must be a valid URL pointer to a data layer exposed through a map service. In this case, we are creating a reference to the Zoning layer in the LOJIC_LandRecords_Louisville map service. What this indicates is that we are going to perform a query against this layer. If you will remember from a previous chapter, dojo.on() is used to register events. In this case, we are registering the complete event for our new QueryTask object. This event fires when the query has been completed, and in this case will call the addToMap() function specified as a parameter to on().

8. Now we'll define the input parameters for the task by creating a Query object. In the first line, we create a new Query instance, and then we set the Query.returnGeometry and Query.outFields properties. Setting Query.returnGeometry equal to true indicates that ArcGIS Server should return the geometric definition of the features that matched the query, while in Query.outFields we've specified a wildcard indicating that all fields associated with the Zoning layer should be returned for the features returned as a result of the query. Add the following highlighted lines of code just below the code you entered in the previous step:

require(['
    "esri/map", "esri/tasks/query", "esri/tasks/QueryTask",
    "esri/tasks/FeatureSet", "esri/symbols/SimpleFillSymbol",
    "esri/symbols/SimpleLineSymbol", "esri/InfoTemplate",
    "dojo/_base/Color", "dojo/on", "dojo/domReady!"
], function(Map, Query, QueryTask, FeatureSet, 
        SimpleFillSymbol, SimpleLineSymbol, InfoTemplate, Color, 
        on) {

For More Information:
Performing Spatial and Attribute Queries

```javascript
map = new Map("mapDiv", {
  basemap: "streets",
  center: [-85.748, 38.249], // long, lat
  zoom: 13
});

symbol = new SimpleFillSymbol(SimpleFillSymbol.STYLE_SOLID,
  new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID, new
    Color([111, 0, 255]), 2), new Color([255, 255, 0, 0.25]));

infoTemplate = new InfoTemplate("${OBJECTID}", "${*}"");

queryTask = new QueryTask("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Louisville/LOJIC_LandRecords_Louisville/MapServer/2");
queryTask.on("complete", addToMap);

query = new Query();
query.returnGeometry = true;
query.outFields = ["*"];
```

9. Add a line of code that registers the `Map.click` event to a `doQuery` function. The `doQuery` function will be passed the point on the map that was clicked by the user. This map point will be used as the geometry in the spatial query. In the next step, we will create the `doQuery` function that will accept the point clicked on the map:

```javascript
  , function(Map, Query, QueryTask, FeatureSet, SimpleFillSymbol, SimpleLineSymbol, InfoTemplate, Color, on) {
    map = new Map("mapDiv", {
      basemap: "streets",
      center: [-85.748, 38.249], // long, lat
      zoom: 13
    });
```

For More Information:
symbol = new SimpleFillSymbol(SimpleFillSymbol.STYLE_SOLID,
    new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID, new
    Color([111, 0, 255]), 2), new Color([255, 255, 0, 0.25]));

infoTemplate = new InfoTemplate("${OBJECTID}", "${*}");

map.on("click", doQuery);

queryTask = new QueryTask("http://sampleserver1.arcgisonline.com/
    ArcGIS/rest/services/Louisville/LOJIC_LandRecords_Louisville/
    MapServer/2");
queryTask.on("complete", addToMap);

query = new Query();
query.returnGeometry = true;
query.outFields = ["*"];

10. Now we’ll create the doQuery function that executes the QueryTask using
    the Query properties we set in the require() function along with the map
    point clicked by the user, which is used in the Query.geometry function. 
    The doQuery function accepts a point that was clicked on the map, which 
    can be retrieved using the mapPoint property. The mapPoint property returns
    a Point object, which is then used to set the Query.geometry property 
    that will be used to find the zoning parcel the user has clicked on the map. 
    Finally, the QueryTask.execute() method is executed. After the task has 
    executed, a FeatureSet object containing the records that match the query
    will be returned. The question now is where are the results returned? Add the
    following code block just below the closing brace for the require() function:

function doQuery(evt) {
    //clear currently displayed results
    map.graphics.clear();

    query.geometry = evt.mapPoint;
    query.outSpatialReference = map.spatialReference;
    queryTask.execute(query);
}

For More Information:
www.packtpub.com/building-web-and-mobile-arcgis-server-applications-with-
javascript/book
11. Remember that we registered the `QueryTask.complete` event to run the `addToMap()` function. We haven't created this function yet. Add the following code to create the `addToMap()` function. This function will accept a `FeatureSet` object returned as a result of the query and plot the features on the map. Also notice that an info template is defined for the feature. This will create an `InfoWindow` object to display the attributes of the returned feature:

```javascript
function addToMap(results) {
    var featureArray = results.featureSet.features;
    var feature = featureArray[0];
    map.graphics.add(feature.setSymbol(symbol).
        setInfoTemplate(infoTemplate));
}
```

You can view the solution code for this exercise in the `spatialquery.html` file.

12. Click on the **Run** button to execute the code. You should see the map in the following screenshot. If not, you may need to recheck your code for accuracy.

![Map Screenshot](image)

---

**For More Information:**
Click anywhere on the map to run the query. You should see the highlighted zoning polygon there, similar to what you can see in the following screenshot:

For More Information:
Performing Spatial and Attribute Queries

Now, click on the highlighted zoning polygon to display an Info Window that details the attributes associated with the polygon.

In the task just completed, you learned how to use the `Query` and `QueryTask` objects to create a spatial query that locates the zoning polygon that intersects the point the user has clicked on the map.

For More Information:  
Summary
In this chapter, we introduced the concept of tasks in ArcGIS Server. ArcGIS Server provides a number of tasks for commonly used operations in a web mapping application. Attribute and spatial queries are common operations in web mapping applications. To support these queries, the ArcGIS API for JavaScript provides a QueryTask object that can be used to execute these queries on the server. When created, the QueryTask object accepts a URL that points to a layer that will be queried in a map server. Various input parameters to the QueryTask are provided through the Query object. Input parameters can include a where property to perform attribute queries, a geometry property to perform spatial queries, an outFields property to define the set of fields that should be returned, and several other supporting properties. After the query has completed on the server, a FeatureSet object is returned to a callback function defined in the application. The callback function can then display the FeatureSet (which is just an array of Graphic objects) on the map. In the next chapter, you will learn how to use two additional tasks: IdentifyTask and FindTask. Both can be used to return the attributes of features.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.