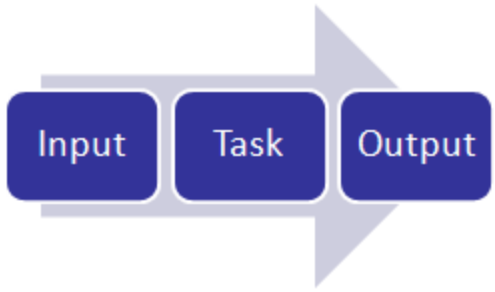
Chapter 12: Spatial and Attribute Queries

In this chapter we will discuss the many types of tasks that can be performed with the ArcGIS Server 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 which can be specified through dojo.require( ).

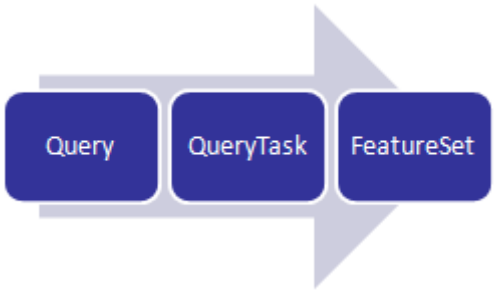
All tasks in the ArcGIS Server API for JavaScript follow the same pattern. This pattern is easily recognizable once you've worked with one or more tasks for any length of time. A diagram of this task process is provided below. An input object is used to supply input parameters to the task. Using these input parameters, the task performs a specific function and then an output object is returned containing the results of the task. The first task we'll examine is QueryTask. Notice the pattern of this task object as you work through the materials. This same basic pattern will be repeated with all the task objects that we cover in the next few chapters.



Performing Spatial and Attribute Queries with Query Task

With the ArcGIS Server Query Task you can 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. Some examples would perhaps be illustrative at this point. An attribute query might search for all land parcels with a valuation of greater than $100,000. A spatial query could be used to find all land parcels that intersect a 100 year floodplain, and a combination query might search for all land parcels with a valuation of greater than $100,000 and whose geometry intersects the 100 year floodplain.

As you'll see with most tasks, queries are performed using a sequence of objects that typically include input to the tasks, execution of the tasks, and a result set returned from the operation.



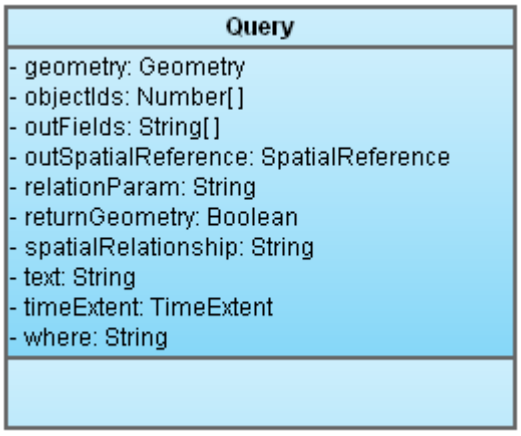
Query Class

The input for your query is stored in a Query object (see figure below) which contains various parameters that can be set for the query. At least one of the following properties must be set: *geometry*, *text*, or *where*. Commonly used optional properties include *outFields*, *outSpatialReference*, and *returnGeometry*.

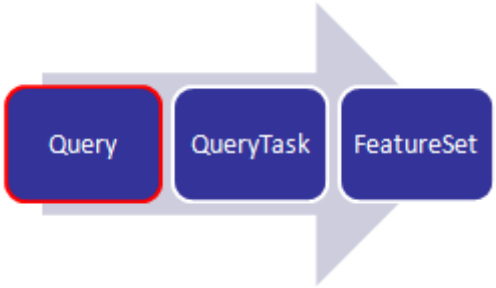
In the code example you see below, the *text* property has been set along with *outfields* and *returnGeometry*. The *text* and *where* properties are used to define an attribute query while the *geometry* and *spatialRelationship* properties in combination define a spatial query. Setting the *returnGeometry* property to 'false' indicates that we do not want ArcGIS Server to return features that matched the query. Only the tabular data will be returned in this case. This would be typical for an application that wants to display the results in a non-map component such as a grid. The *outfields* property is an array of attribute fields that should be returned in the results. Field names should be enclosed by double quotes and separated by commas. The fields must actually exist in the layer and must be defined by the actual field names rather than a field alias. You should get into the habit of setting this property to limit the fields to only those you expect to use in the results. Performance of your application will be improved since it avoids the performance hit of having to return fields of information that will not be used.

```
var myQuery;

//build query filter
myQuery = new esri.tasks.Query();
myQuery.returnGeometry = false;
myQuery.outFields = ["STATE_NAME","POP2007","MALES","FEMALES"];

myQuery.text = 'California';|
```

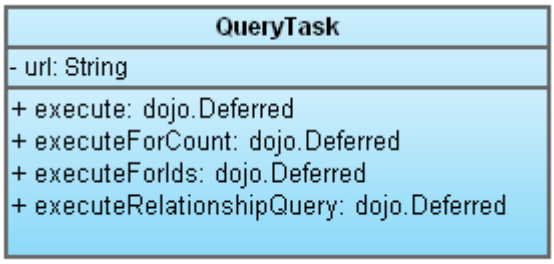
The QueryTask object executes the task using the input provided in the Query object, and the result set is stored in a FeatureSet object which contains an array of Graphic features.



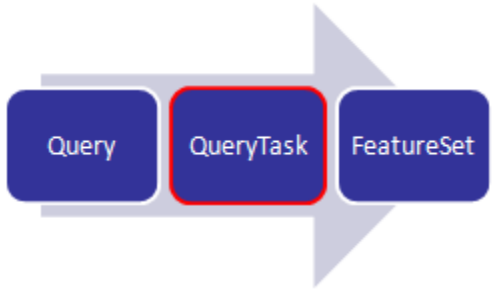
A Query object is used as input to a QueryTask and is defined by properties such as geometry, where, and text. The geometry property is used to input a geometry that will be used in a spatial query and will be a point, line, or polygon geometry. The 'where' property is used to define an attribute query, and the 'text' property is used to perform a 'where' clause containing a 'like' modifier. 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 (outfields), the output spatial reference for the return geometry (outputSpatialReference), and the actual geometry of the features that meet the query conditions.



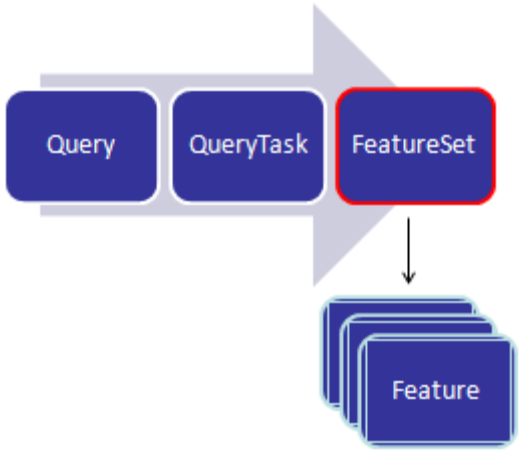
Once you've defined the input properties in a Query object you can then use QueryTask to execute the query on a layer. Notice in the code example below that we first create a new variable called 'myQueryTask' which points to layer 6 in the ESRI_CENSUS_USA map service. We then create the Query object containing the input properties of the query and finally we use the execute( ) method on QueryTask to perform the query. Execute 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.




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

//build query filter
myQuery = new esri.tasks.Query();
myQuery.returnGeometry = false;
myQuery.outFields = ["STATE_NAME","POP2007","MALES","FEMALES"];

myQuery.text = 'Oregon';

//execute query
myQueryTask.execute(myQuery,showResults);
```



As I mentioned, the results of a query are stored in a FeatureSet object which is simply an array of Feature graphics which you can then plot on your map if you wish. Each feature in the array can contain geometry, attributes, symbology, and an InfoTemplate. Typically these features are plotted on the map as graphics.



Now that you understand the basic concepts of performing a query using a QueryTask we'll take a look at some specific examples.

Performing Spatial Queries

Spatial queries, such as finding parcels of land that intersect a 100-year floodplain can be accomplished through QueryTask by setting the 'geometry' and optionally the 'spatialRelationship' properties on the Query object. The geometry property defines a geometry to use as the spatial filter. Valid geometry types include Extent, Point, Multipoint, Polyline, or Polygon. The spatial relationship as specified by 'spatialRelationship' is applied to the geometry while performing the query. One of the valid constants seen below that defines the type of spatial relationship needs to be supplied.

SPATIAL_REL_CONTAINS	Part or all of a feature from feature class 1 is contained within a feature from feature class 2.
SPATIAL_REL_CROSSES	The feature from feature class 1 crosses a feature from feature class 2.
SPATIAL_REL_ENVELOPEINTERSECTS	The envelope of feature class 1 intersects with the envelope of feature class 2.
SPATIAL_REL_INDEXINTERSECTS	The envelope of the query feature class intersects the index entry for the target feature class.
SPATIAL_REL_INTERSECTS	Part of a feature from feature class 1 is contained in a feature from feature class 2.
SPATIAL_REL_OVERLAPS	Features from feature class 1 overlap features in feature class 2.
SPATIAL_REL_RELATION	Allows specification of any relationship defined using the Shape Comparison Language.
SPATIAL_REL_TOUCHES	The feature from feature class 1 touches the border of a feature from feature class 2.
SPATIAL_REL_WITHIN	The feature from feature class 1 is completely enclosed by the feature from feature class 2

The code example below details this procedure.

```
function populateProperty() {
   //Create Find Task using the URL of the map service to search

   var queryTask = new esri.tasks.QueryTask(urlMain + "ArcGIS/rest/services/RLTK2/FeatureServer/0");
   dojo.connect(queryTask, "onComplete", showPropertyResults);

   //Create the find parameters
   var queryParams = new esri.tasks.Query;
   queryParams.returnGeometry = false;
   queryParams.outFields = ["OBJECTID","Name","Address","City","State","Zip"];
   queryParams.geometry = tradeAreaExt;
   queryParams.spatialRelationship = esri.tasks.Query.SPATIAL_REL_INTERSECTS;
   queryTask.execute(queryParams);
}
```

Performing Attribute Queries

Attribute queries can be performed against layers and tables exposed by an ArcGIS Server instance. The 'where' and 'text' properties are used to define attribute queries. Any legal SQL 'where clause' can be used in the 'where' property. Similarly, the 'text property defines a where clause as a 'like' statement. However, the field used to perform the query is the display field defined in the map document (mxd). You can also consult the Services Directory for the layer you are querying to determine the display field.

```
function populateProperty() {
   //Create Find Task using the URL of the map service to search

   var queryTask = new esri.tasks.QueryTask(urlMain + "ArcGIS/rest/services/RLTK2/FeatureServer/0");
   dojo.connect(queryTask, "onComplete", showPropertyResults);

   //Create the find parameters
   var queryParams = new esri.tasks.Query;
   queryParams.returnGeometry = false;
   queryParams.outFields = ["OBJECTID","Name","Address","City","State","Zip"];
   queryParams.where = "OBJECTID >= 1";

   queryTask.execute(queryParams);
}
```

Combining Spatial and Attribute Queries

Spatial and attribute queries can easily be combined simply by including the relevant properties for each type of query.

```
function populateProperty() {
   //Create Find Task using the URL of the map service to search

   var queryTask = new esri.tasks.QueryTask(urlMain + "ArcGIS/rest/services/RLTK2/FeatureServer/0");
   dojo.connect(queryTask, "onComplete", showPropertyResults);

   //Create the find parameters
   var queryParams = new esri.tasks.Query;
   queryParams.returnGeometry = false;
   queryParams.outFields = ["OBJECTID","Name","Address","City","State","Zip"];
   queryParams.where = "OBJECTID >= 1";
   queryParams.spatialRelationship = esri.tasks.Query.SPATIAL_REL_INTERSECTS;
   queryParams.geometry = tradeAreaExt;

   queryTask.execute(queryParams);
}
```