Learning Mapfluence REST

Welcome to **Learning Mapfluence REST**, a quick, step-by-step introduction to the use of REST in Mapfluence. This document covers the topics listed below, which fall into three main categories.

What you need to know to work effectively with REST in Mapfluence:

- **Mapfluence REST Overview**
- **Mapfluence REST Topics**
- **Mapfluence REST Paths**
- **Data Catalog REST**
- **REST Maps Overview**

Step-by-step: using REST for mapping in Mapfluence:

- **REST for Simple Layers**
- **REST for Class Layers**
- **REST for Thematic Layers**
- **REST for Heatmap Layers**
- **REST for Label Layers**
- **REST for Named Layers**
- **REST for Legends**
- **REST for Tiles**

Step-by-step: using REST for queries in Mapfluence:

- **REST for Spatial Queries**
- **REST for Estimate Queries**
- **REST for Spatial Operations**
- **REST for Geocoding**

Mapfluence is a product of Urban Mapping, Inc. For further information please visit http://www.urbanmapping.com.

1. Mapfluence REST Overview

The following sections provide a general foundation for using Mapfluence via REST:

- About Mapfluence REST
- Mapfluence Mapping REST
- Mapfluence Query REST

1.1 About Mapfluence REST

Mapfluence 2.0 REST API provides access to Mapfluence via a RESTful HTTP interface. The interface includes resources for mapping, spatial queries, and access to the Mapfluence data catalog. The REST capabilities are a subset of the JS capabilities. Use of the REST API requires building an application around it to perform tasks such as constructing requests and handling responses.

A thorough explanation of REST (Representational State Transfer) architecture and practices is beyond the scope of this document. Ample information about REST is available on the Internet. For example, a helpful overview from IBM may be found at http://www.ibm.com/developerworks/webservices/library/ws-restful/. A familiarity with HTTP/REST vocabulary such as resources, requests, and responses is required to take full advantage of the information in the following sections.

In addition to general REST/HTTP knowledge, using Mapfluence through a RESTful HTTP interface requires familiarity with a number of Mapfluence-specified REST details, including Resource Path and Collections. These topics are covered in the Mapfluence REST guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/rest.

Note: Important additional concepts related to working with REST in Mapfluence are covered in **Mapfluence REST Topics**.

1.2 Mapfluence Mapping REST

A Mapfluence map combines a base map layer, which is a basic representation (land, water) of the area being mapped, with one or more additional overlaid layers displaying geographically-correlated information about that area. The information typically comes from the Mapfluence Data Catalog, though custom data catalogs may also be used. A searchable list of the available datasets in the Mapfluence Data Catalog (with each list item linking to a detailed description) is available at http://developer.urbanmapping.com/content/data-catalog.

Maps are generated in response to a mapping request from a client. The request specifies the area to be included and selects the features (objects or events) to be represented on each overlaid layer. Mapping requests involves calls to the following three Mapfluence objects:

- `/map` creates an interactive map.
- `/tile` renders and returns a composite image of a map's layers at a specific zoom (a tile).
- `/legend` retrieves a map legend describing a given map image layer.

In the REST approach, a mapping request takes the form of an HTTP GET. The path of the request is parsed by the Mapfluence server to determine what data the map's layers each represent, the order (front to back) in which they will be composited, how their graphical elements are to be rendered (colors, opacity, icons, font size, etc.), and the base map (Mapfluence, Bing, Google, etc.) onto which the layers will be overlaid. Mapfluence retrieves the specified data for each layer from the data catalog, builds the layers into a map, and renders a selected portion of the map as an image tile that is returned to the client in the HTTP response.

1.3 Mapfluence Query REST

Mapfluence queries return one of the following types of information:

- **Spatial queries** (`/spatialquery`, `/estimate`, `/spatialops`, and `/geocoder`): geographically correlated data from datasets in the Mapfluence Data Catalog.
- **Data catalog queries** (`/catalog`): information about the data catalog itself.

The basic concept of these queries is the same as for a map, meaning that a REST query takes the form of an HTTP GET, and the path of the request is parsed by the Mapfluence server to determine what data to return to the client. Unlike a mapping operation, however, results are returned in the form of text rather than as a map. Depending on the query and the type of data requested, the data is returned as JSON (for `/estimate`, `/spatialops`, `/catalog`, and sometimes for `/spatialquery`) or as GeoJSON (for `/geocoder` and sometimes for `/spatialquery`; see http://GeoJSON.org/).

Note: A searchable list of the available datasets in the Mapfluence Data Catalog (with each list item linking to a detailed description) is available at http://developer.urbanmapping.com/content/data-catalog.

2. Mapfluence REST Topics

The following topics cover important concepts that must be understood to work effectively with the Mapfluence REST API:

- Paths and Collections
- Resources in REST
- Geometries in REST
- Select Statements in REST
- Expressions in REST
- Date Filters in REST
- Where Filters in REST
- Color and Size in REST
- Literal Values in REST
- URL Escaping for REST

2.1 Paths and Collections

Using Mapfluence through a RESTful HTTP interface requires familiarity with a number of Mapfluence-specified REST details, including Resource Path and Collections. These topics are covered in the Mapfluence REST guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/rest.

2.2 Resources in REST

As with REST in general, in the Mapfluence REST API objects are accessed as HTTP resources, meaning that they reside at defined locations on the Mapfluence server and are each exposed via a URL that gives the path to a specific location (see http://en.wikipedia.org/wiki/Web_resource).

2.3 Geometries in REST

The geographic position of an object can be represented as a geometry (point, line, or polygon) of latitude and longitude coordinates. The options for specifying a geometry, including GeoJSON, feature ID, and range, are covered in the Mapfluence REST geometry guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/geometries.

2.4 Select Statements in REST

Many Mapfluence requests accept (or require) the select parameter to specify what data to retrieve. For an overview of how to construct select statements see the Mapfluence REST select statements guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/select.

2.5 Expressions in REST

Expressions expand the capability of select statements by letting you retrieve more than just data from an attribute column or feature field (e.g. display the sum of the values of two columns). Arithmetic, geometry, and aggregate expressions are supported. The use of expressions is covered in the Mapfluence REST expressions guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/expressions.

2.6 Date Filters in REST

All Mapfluence feature requests allow you to filter results by date and time. The structure of date/time filters depends on the type of geometry table — periodic or event — used in the request. For an explanation of using date filters please see the Mapfluence REST dates guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/dates.

2.7 Where Filters in REST

Where statements filter the features returned by Mapfluence feature requests. Mapfluence supports both data filters and geometry filters. For an explanation of the usage of where filters please see the Mapfluence REST where filter guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/where.

2.8 Color and Size in REST

To learn how the color and size of map points, lines, and polygons are specified in the Mapfluence REST API please refer to the Mapfluence styles guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/styles.

2.9 Literal Values in REST

To learn how literal values (array, Boolean, date, etc.) are specified in the Mapfluence REST API please refer to the Mapfluence REST literal values guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/literals.

2.10 URL Escaping for REST

Mapfluence requests often contain characters that are reserved for use in URLs. These characters must be properly escaped before being sent to the server. To learn how, please refer to the Mapfluence REST URI escaping guide at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/url-escaping.

3. Mapfluence REST Paths

As described in **Mapfluence REST Overview**, all of the information needed for Mapfluence to respond to a client request is built into the path used to make that request, which means that understanding how to build the path is key to getting the server to respond with the kind of map or data you want. These paths are covered in the following sections:

- About Mapfluence Paths
- Mapping Paths
- Spatial Query Paths
- Data Catalog Paths

3.1 About Mapfluence Paths

The paths that are passed to the Mapfluence server generally appear in one of two contexts:

- **HTML link**: A path used in an HTML document (either as a string or created dynamically by a script, e.g. JavaScript).
- **HTTP request**: An HTTP request that is built and sent by an application.

The following examples show these two different forms of the same path, which requests a list of the datasets in the Mapfluence Data Catalog:

- HTML link:

```
http://query.mapfluence.com/2.0/<apiKey>/catalog/datasets
```

- HTTP request:

```
?# Request
GET /2.0/<apiKey>/catalog/datasets HTTP/1.1
Host: query.mapfluence.com
Accept: application/json
```

As shown above, Mapfluence REST paths include the following parts:

- **Host**: the location of the service, e.g. `http://query.mapfluence.com/`
- **Version**: the Mapfluence version, e.g. `2.0`.
  Note: For a discussion of versioning in the Mapfluence JavaScript API, which applies as well to REST versioning, see **Mapfluence Versioning** at http://developer.urbanmapping.com/docs/mapfluence/js/2.0/guides/versions.
- **API key**: A unique identifying key (e.g. `MFDOCS`) that Mapfluence requires in order to grant access to the system. Sign up for a key at http://developer.urbanmapping.com/devreg.

- **Resource path**: The string that will be parsed by the Mapfluence server to determine what data you are requesting and how you want it returned, (e.g. `/catalog/datasets`). The resource path may also include an HTTP query string.

While these basic path components (except version) are present in all Mapfluence REST requests, the structure of the complete path varies depending whether the request is for mapping or text data.

Note: For brevity, example paths in the remainder of this document will be expressed in the HTML link form described above. These paths may be used in the address field of a browser to see the returned map or data.

3.2 Mapping Paths

The following example illustrates the path structure used for a mapping request to the Mapfluence visual API, in this case for a map with a base layer and one simple layer overlay showing school district boundaries:

```
http://query.mapfluence.com/2.0/MFDOCS/map/from=umi.us_schl_dists.unified_geometry|mode=simple|border=black_1|colors=clear/b/?lat=37.6&lng=-122.2&width=640&height=480&zoom=10
```

In this mapping example, the path is composed of the following parts:

- `http://query.mapfluence.com`: the host.
- `/2.0`: the Mapfluence version.
- `/MFDOCS`: an API key; substitute your own key.
- `/map`: the endpoint used by Mapfluence to route the request to mapping resources on the server.
- `/from=umi.us_schl_dists.unified_geometrymode=simple|border=black_1|colors=clear`: the string that will be parsed by the server to determine what to return in the response. In this example, the string defines a single layer; multiple layers may also be defined, with the layers each separated by a slash.
- `/b/`: a base map code identifying the source of the base map, in this case Bing Maps. For Google Maps this part of the path would be `/g/`, and for Mapfluence's own base map it would be `/mapfluence/`.
- `?lat=37.6&lng=-122.2&width=640&height=480&zoom=10`: an optional HTTP query string used to pass additional parameters for the returned map. In this case the query string specifies the centerpoint coordinates (lat for latitude, lng for longitude), the size (width, height), and the zoom of the requested map. One additional query parameter that is not shown here but is also supported is the Boolean `legend`, which enables (if 1) or disables (if 0, the default) the display of a legend for a simple, class, or thematic layer.

3.3 Spatial Query Paths

The following example illustrates the path structure used for a non-mapping request, in this case a spatial query that asks for a feature named “Alaska” from the data catalog’s geometry table of U.S. states (from the U.S. Census dataset):

```
http://query.mapfluence.com/2.0/MFDOCS/spatialquery.json?from=umi.us_census00
.state_geometry&select=name&where=name:'Alaska'&date=2010-08-01T07:00:00Z
```

In this query example, the path is composed of the following parts:

- `http://query.mapfluence.com`: the host.
- `/2.0`: the Mapfluence API version (see http://developer.urbanmapping.com/docs/mapfluence/js/2.0/guides/versions)
- `/MFDOCS`: The API key (use your own key instead).
- `/spatialquery.json`: the location of the Mapfluence resource that will respond to the query, in this case a spatial query resource.
- `?from=umi.us_census00.state_geometry&select=name&where=name:'Alaska'&date=2010-08-01T07:00:00Z`: the query string that will be parsed by the server to determine what to return in the response.

In spatial queries (`/spatialquery`, `/estimate`, `/spatialops`, and `/geocoder`) many different parameters may be specified using the query string (for specifics, refer to the REST reference for each type of spatial query).

Note: Query strings are case sensitive.

3.4 Data Catalog Paths

The path structure described in **Spatial Query Paths** is used not only for spatial queries but also for data catalog queries (`/catalog`), such as the following request for a list of datasets in the Mapfluence data catalog:

```
http://query.mapfluence.com/2.0/MFDOCS/catalog/datasets
```

In data catalog queries the query string is optional. The only currently supported query string parameter is `start`, which is used to specify the index of the first item when a query returns a list. In a `catalog/datasets` query, for example, if `start=50` then the first dataset in the returned list will be the dataset at index 50 in the collection of datasets.

```
http://query.mapfluence.com/2.0/MFDOCS/catalog/datasets/?start=50
```

4. Data Catalog REST

As described in **Mapfluence REST Overview**, a Mapfluence request may be for data returned as a map, for data returned as text, or for metadata about the Mapfluence data catalog. In each case, the source of the information returned from the query is the Mapfluence catalog. The Mapfluence data catalog is a collection of datasets, each of which is made up of one or more tables of information. The use of REST to access the Mapfluence Data Catalog is covered in the following sections:

- Mapfluence Paths and SQL
- Catalog Access via REST
- Using Geometry Tables
- Using Attribute Tables
- Using Geometries with Attributes

4.1 Mapfluence Paths and SQL

The Mapfluence REST API has been designed to incorporate, where possible, familiar SQL database terminology into the structure of mapping and query requests. For example, consider a request for data on EPA brownfields where air contamination was found in 2009. In SQL such a query, which would include `FROM` and `WHERE` statements, might look like this:

```
SELECT geometry
FROM epabrnfld.bf2009_geom
WHERE epabrnfld.bf2009.air_f = 't'
```

In a Mapfluence REST query, the part of the path that specifies the data source would include `from` and `where` statements like this:

```
from=epabrnfld.bf2009_geom|where=epabrnfld.bf2009.air_f:t
```

4.2 Catalog Access via REST

The Mapfluence REST API includes three main categories of queries used to access metadata about the contents of the data catalog:

- Dataset queries, e.g. `/catalog/datasets`
- Geometry table queries, e.g. `/catalog/geometryTable/`
- Attribute table queries, e.g. `/catalog/attributeTable/`

These data catalog queries allow you to discover which datasets, attribute tables, and geometry tables are available within the Mapfluence Data Catalog. In the following examples, the first query returns a list of all datasets in the data catalog, and the second query returns the metadata associated with the dataset umi.us_census00, whose display name is "Census Demographics and Boundaries":

```
http://query.mapfluence.com/2.0/MFDOCS/catalog/datasets

http://query.mapfluence.com/2.0/MFDOCS/catalog/dataset/umi.us_census00
```

Notes:
» For a list of data catalog queries in each category (dataset, geometry, and attribute), as well as links to the reference for each query, see **Catalog Resources** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/catalog/intro.
» For a list of the available datasets in the Mapfluence Data Catalog (with each list item linking to a detailed description) see http://developer.mapfluence.com/content/data-catalog.

4.3 Using Geometry Tables

Once you choose a dataset, you can drill down further to return metadata about the contents of tables and specific attributes their features. In the following example, the first query returns a list of geometry tables associated with the Census Demographics and Boundaries dataset (umi.us_census00). The second query returns a list of the feature sets of the geometry table umi.us_census00.county_geometry, which contains the polygonal geometry of U.S. counties. The third returns the geometry of a single feature (Traverse County, MN) identified by its feature ID (27155):

```
http://query.mapfluence.com/2.0/MFDOCS/catalog/dataset/umi.us_census00/geometryTables

http://query.mapfluence.com/2.0/MFDOCS/catalog/geometryTable/umi.us_census00.county_geometry/featureSets

http://query.mapfluence.com/2.0/MFDOCS/catalog/feature/umi.us_census00.county_geometry.27155
```

With the first geometry table query above we learn that within the Census Demographics and Boundaries dataset there are geometry tables for a number of different feature types:

Geometry tables	Display name
umi.us_census00.blkgrp_geometry	2000 Block Group Boundaries
umi.us_census00.cbsa_geometry	Core Based Statistical Area Boundaries, 2003
umi.us_census00.county_geometry	2000 County and County Equivalent Boundaries
umi.us_census00.place_geometry	2000 Incorporated and Census-Designated Place

	Boundaries
umi.us_census00.state_geometry	2000 State and State Equivalent Boundaries
umi.us_census00.tract_geometry	2000 Tract Boundaries
umi.us_census00.urb_ac_geometry	2000 Urban Areas and Urban Clusters Boundaries
umi.us_census00.zcta_geometry	2000 5-Digit ZIP Code Tabulation Area Boundaries

Geometry tables like those listed in the left-hand column may be used during mapping to specify (using the `from` property in a layer definition) the types of areas and/or boundaries that should be displayed on a map layer.

4.4 Using Attribute Tables

As with geometry tables, you can also drill down into attribute tables. In the following example, the first query returns a list of attribute tables associated with the Census Demographics and Boundaries dataset (umi.us_census00). The second query returns a list of the attribute record sets of the attribute table umi.us_census00.county, which contains records on U.S. counties. The third returns metadata, in this case the Federal Information Processing Standard (FIPS) code, from the specified column (`.fips`) in an attribute table:

```
http://query.mapfluence.com/2.0/MFDOCS/catalog/dataset/umi.us_census00/attributeTables

http://query.mapfluence.com/2.0/MFDOCS/catalog/attributeTable/umi.us_census00.county/attributeRecordSets

http://query.mapfluence.com/2.0/MFDOCS/catalog/attributeColumn/umi.us_census00.county.fips
```

Using the metadata returned from attribute table queries enables you to identify a data source that can be specified (using the `select` property) in a spatial query.

4.5 Using Geometries with Attributes

Mapping often involves showing how the value of a given attribute varies among geometries. For example we might want to show how median household income (an attribute) varies depending on census tracts (a geometry). To do this we need to include two types of property definitions in our mapping path:

- The attribute whose values are to be displayed is specified with the `select` property, which identifies the source of the data (an attribute table).

- The geometry unit (e.g. census tract, city, county, etc.) for which the attribute values are to be displayed is specified with the `from` property, which identifies the source of the geometries (a geometry table).

Using our example of median household income by census tract, a thematic layer (`mode=theme`) displaying this information would be defined as follows, with select specifying the source of the census data and `from` specifying the source of the census tract geometry (the other properties will be explained in subsequent sections of this document):

```
/mode=theme|select=umi.us_census00.stats.med_hhinc|from=umi.us_census00.tract
_geometry|breaks=200000,400000,600000,800000,1000000|colors=FFFFFF80,0000FF80
,00FF0080,FFA50080,FF000080
```

The list of geometry tables available for a given attribute table may be discovered using the data catalog calls described in previous sections. They can also be found manually with a data catalog search at http://developer.mapfluence.com/content/data-catalog.

5. REST Maps Overview

The Mapfluence REST API enables the building of HTTP requests that build maps and return a rendered tile to the requesting client. The use of REST to build paths for the various types of mapping layers are covered in the following sections:

- Base Maps with REST
- Map Layers with REST

Note: For complete details on the purpose and valid values of map properties, refer to the **/map** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/MapPermalink.

5.1 Base Maps with REST

A base map, which is an image that displays land, water, and perhaps other geographical features (national boundaries, public lands, etc), serves as the bottom layer for all data overlays in a mapping application, and is therefore required for all mapping operations. Mapfluence supports several different sources for base maps.

As described in **Mapping Paths**, in REST the base map source is specified using a base map code:

- Bing Maps, indicated with code “b”.
- Google Maps, indicated with code “g”.
- Mapfluence Maps (default), indicated with code “`mapfluence`”.

The base map code appears in the path after the map layer specifications and before the query string (if any). If no base map code is present in the path, the default (Mapfluence) base map will be used.

5.2 Map Layers with REST

As we’ve seen in **Mapping Paths**, the string that lies in between the endpoint and the base map code is the part of the path that actually defines the map that is built from data catalog information, rendered on top of the base map (as specified with the base map code), and returned by the Mapfluence server.

For each layer used in the map the layer definition string is made up of a series of `property=value` pairs, each of which is separated from the next by a pipe ("|"). The following example shows a string used to specify the properties of single simple layer:

```
/mode=simple|from=umi.us_schl_dists.unified_geometry|border=black_1|colors=clear
```

If multiple layers are include in a map, they are specified in order from bottom to top (back to front), with each layer separated from the next by a slash ("/"). The following example shows a map definition made up of three overlaid layers: a thematic layer of median household income (from U.S. Census statistics), a simple layer of school district boundaries, and a simple layer of airport locations. As you can see, the properties to define depend on the type of layer (`mode=simple`, `class`, `theme`, etc.):

```
/mode=theme|select=umi.us_census00.stats.med_hhinc|from=umi.us_census00.tract_geometry|breaks=200000,400000,600000,800000,1000000|colors=FFFFFF80,0000FF80,00FF0080,FFA50080,FF000080/mode=simple|from=umi.us_schl_dists.unified_geometry|border=black_1|colors=clear/mode=simple|from=umi.ntad_airports.geometry|icons=icon_air_transportation_24x20.png
```

6. REST for Simple Layers

Not surprisingly, the simplest type of Mapfluence layer is the simple layer, which displays the geometries (points, lines, or polygons) of features along with, optionally, the boundaries of those geometries.

Simple layers are referred to as simple because all geometries that meet the criteria for inclusion in the layer are displayed the same way. A map showing all census designated places (population concentrations as defined by the U.S. Census Bureau), for example, might use a simple layer in which all CDPs are indicated with the same color and opacity, and all boundaries between CDPs are indicated with lines of the same color and weight.

In the following sections we'll use Mapfluence REST to create maps with two different kinds of simple layers:

- Simple Polygonal Layer
- Simple Point Layer

6.1 Simple Polygonal Layer

One common simple layer type displays polygonal geometries, such as cities or counties. Let's build the path for a map with a base layer and one simple layer overlay that shows polygonal geometries, in this case school district boundaries:

1. Start with the host, version, API key, and endpoint:

```
http://query.mapfluence.com/2.0/MFDOCS/map
```

2. Add the layer definition, which means specifying the values of the required layer properties, separating each property value pair with a pipe:
   - mode=simple: the layer type, in this case `simple`.
   - from=us_schl_dists.uni2008_geom: the source of the geometries that will be overlaid, which is the geometry table (`uni2008_geom`) in the Mapfluence dataset covering U.S. school districts.
   - border=black_1: the color and line weight of the border, which shows the boundaries between districts.
   - style=00000000: the color and transparency, stated as RGBA expressed in hex, that will be used to render the geometries. In this case, since the last two digits are 00, the alpha channel is set to fully transparent (geometries are rendered as clear) and the first six digits don't have any effect as long as they represent a valid HTML hex color.
   Note: In a simple layer, `style` is singular.

```
/mode=simple|from=umi.us_schl_dists.unified_geometry|border=black_1|style=000
00000
```

3. We'll also add a query string after the base map code to specify the centerpoint coordinates (lat for latitude, lng for longitude) and the zoom:

```
/?lat=37.5&lng=-120&zoom=7
```

4. Put it all together and you have the complete REST path that Mapfluence will use to build the specified map and return a rendered tile to the requesting client:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=simple|from=umi.us_schl_dists
.unified_geometry|border=black_1|style=00000000/?lat=37.5&lng=-120&zoom=7
```

Here's an excerpt of the map, showing school district boundaries, that is returned from the above URL (to see the full map, open a browser window and enter the path into the address field):

6.2 Simple Point Layer

Another common simple layer type displays features as points at a specific location. Now let's build the path for a map with a base layer and one simple layer overlay that shows features, in this case airports, as points. The airports will be indicated with an icon that we'll specify from the set of available Mapfluence icons at . We'll also use the query string to set the centerpoint, size, and zoom of the map we want returned:

1. Start with the host, version, API key, and endpoint:

```
http://query.mapfluence.com/2.0/MFDOCS/map
```

2. Add the layer definition, which means specifying the values of the required layer properties, separating each property value pair with a pipe:
   - mode=simple: the layer type, in this case `simple`.
   - from=umi.ntad_airports.geometry: the source of the geometries that will be overlaid, which is the geometry table (`arpt_loc_geom`) in the Mapfluence dataset covering U.S. airports.
   - style=icon_air_transportation_24x20: the Mapfluence icon, in this case an airplane, that will be overlaid to indicate the location of each airport. The icon dimensions are specified in pixels (horizontal x vertical) after an underscore.
   Note: In a simple layer, `style` is singular.

   ```
   /mode=simple|from=umi.ntad_airports.geometry|style=icon_air_transportation_24x20
   ```

3. This time, instead of using the default Mapfluence maps we'll add a base map code for Bing Maps after the layer definition:

   ```
   /b
   ```

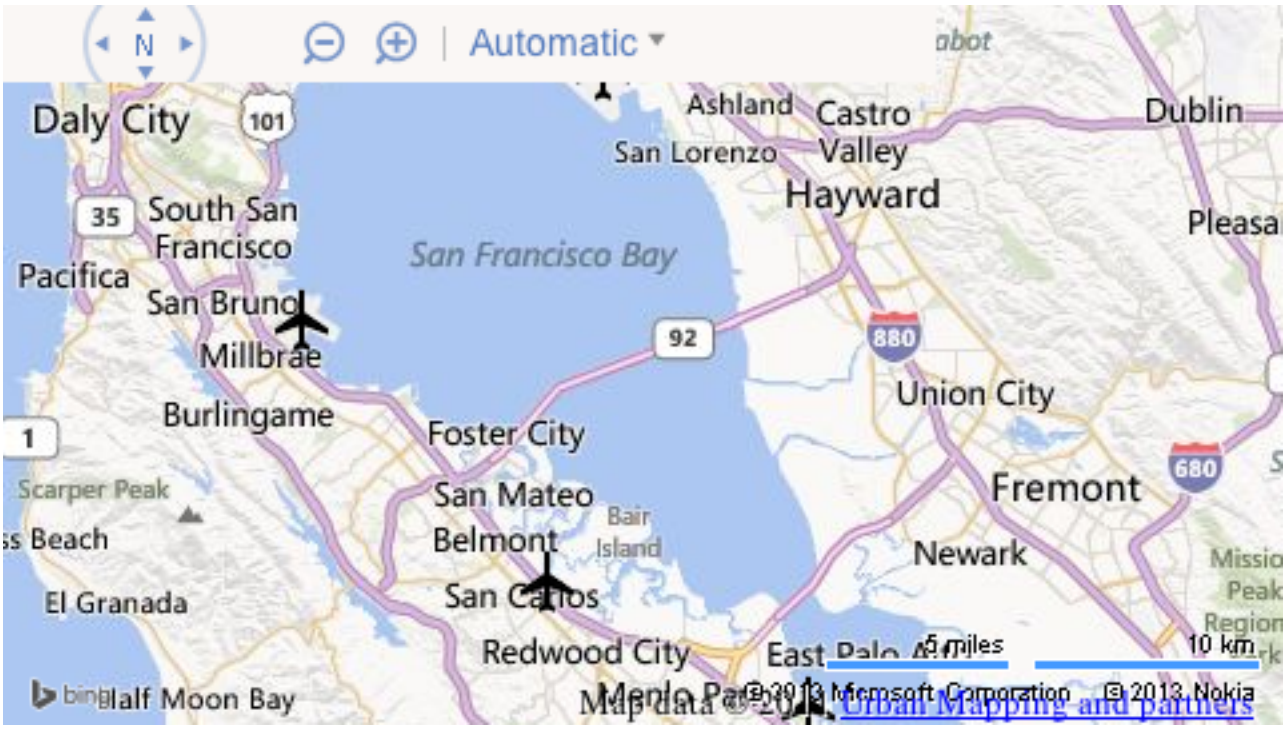
4. As before, we'll add a query string after the base map code to specify the centerpoint coordinates (lat for latitude, lng for longitude) and the zoom, but this time we'll also specify the desired size of the returned map (width, height),:

   ```
   /?lat=37.6&lng=-122.2&zoom=10&width=640&height=480
   ```

5. Put it all together and you have the complete REST path for the map:

   ```
   http://query.mapfluence.com/2.0/MFDOCS/map/mode=simple|from=umi.ntad_airports.geometry|style=icon_air_transportation_24x20/b/?lat=37.6&lng=-122.2&zoom=10&width=640&height=480
   ```

Here's an excerpt of the map, showing airport locations, that is returned from the above URL (to see the full map, open a browser window and enter the path into the address field):



Notes:

» For complete details on the purpose and valid values of simple layer properties, both required and optional, refer to the **Simple Layer** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/layer/Simple.

» A list of available icons, as well as instructions on their use, can be found in **Mapfluence API Icons** at http://tile.mapfluence.com/2.0/icons.html.

» The properties `max_zoom` and `min_zoom` may be used to define different display characteristics for a point (e.g. size in pixels) depending on the current zoom of the map.

7. REST for Class Layers

While a simple layer supports only two classes of geometries (those included in the layer and those that are not), a class layer enables the categorization of features into multiple classes that each represent different values for a selected property. The areas falling into the different classes are distinguished from each other by varying the display characteristics (e.g. color, shading, or opacity) defined for each class. The result is that the display of each feature's geometry depends on that feature's class.

One example of how classes work is the way the U.S. Census Bureau classifies places, which may be cities, towns, or CDP (concentrations of population referred to as "census-designated places"). The following steps illustrate how to map these three types of places as classes in a class layer:

1. Start with the host, version, API key, and endpoint:

```
http://query.mapfluence.com/2.0/MFDOCS/map
```

2. Add the layer definition, which means specifying the values of the required layer properties, separating each property value pair with a pipe:
   - **`mode=class`:** the layer type, in this case `class`.
   - **`select=umi.us_census00.place.place_type`:** the source of the attribute by which the geometries will be classified, in this case an attribute table classifying places by type (city, town, or CDP).
   - **`from=umi.us_census00.place_geometry`:** the source of the geometries that will be overlaid, which is the geometry table in the Mapfluence dataset covering U.S. places.
   - **`date=2009-01-01T08:00:00Z`:** a date, which is used to filter the results using the date attribute in the specified geometry table (the `from` property) and attribute table (the `select` property).
   - **`values=city,town,CDP`:** a list of attribute values that are to be included in the map, in this case the types (classes) of place.
   - **`border=000000_1.3`:** the color and line weight of the border, which shows the boundaries between geometries.
   - **`styles=ff0000, 0000ff, 00ff00`:** a list of colors that will be used to render the geometry associated with the value at the same index in the values list (e.g. ff0000 for cities, 00ff00 for towns, 0000ff for CDPs). These colors could alternatively be expressed as valid HTML color names (red, blue, green).
   - **`opacity=0.4`:** the overall opacity of the layer. This is an alternative to adding an alpha channel specification to the end of each color in the `styles` property.

   Note: In a class layer, `styles` is plural.

```
/mode=class|select=umi.us_census00.place.place_type|from=umi.us_census00.place_geometry|date=2009-01-01T08:00:00Z|values=city,town,CDP|border=000000_1.3|styles=ff0000,0000ff,00ff00|opacity=0.4
```

3. Add the base map code (in this case for Bing Maps) after the layer definition:

```
/b
```

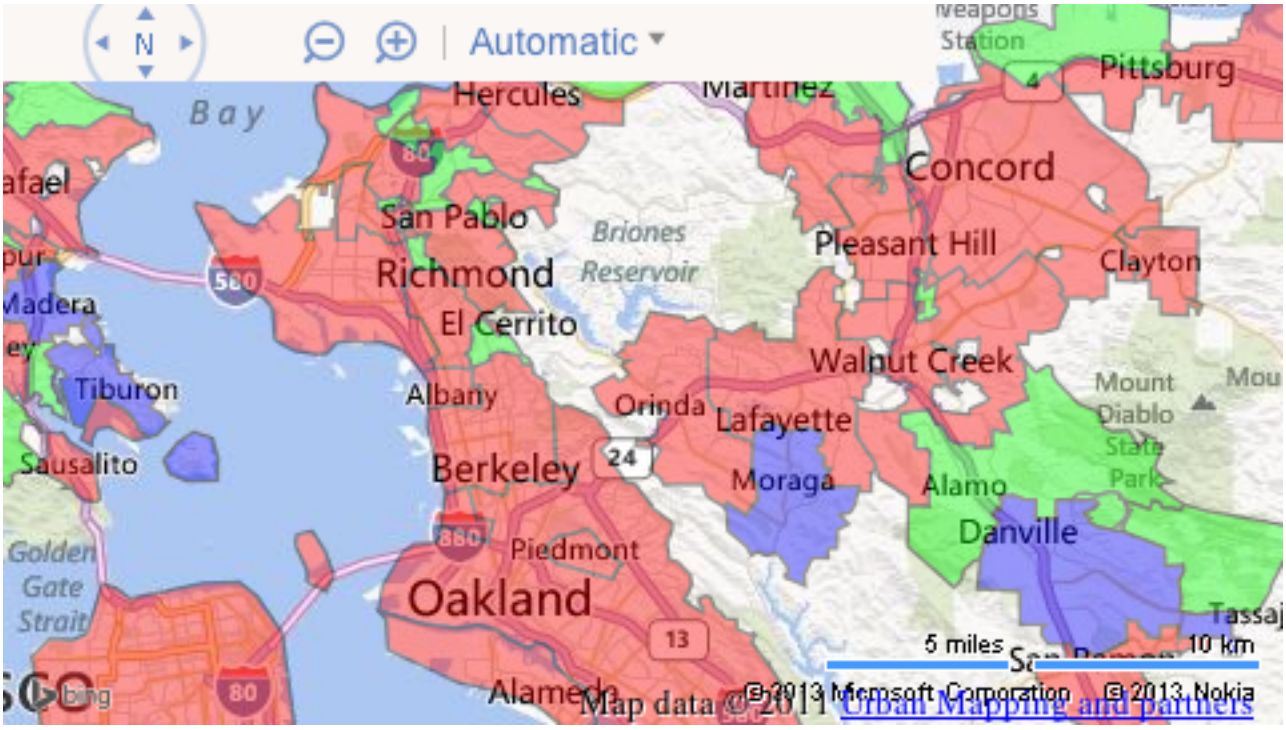
4. Add a query string after the base map code to specify the map's centerpoint, size, and zoom:

```
/?lat=37.6&lng=-122.2&width=640&height=480&zoom=10
```

5. Put it all together and you have the complete REST path for the map:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=class|select=umi.us_census00.place.place_type|from=umi.us_census00.place_geometry|date=2009-01-01T08:00:00Z|values=city,town,CDP|border=000000_1.3|styles=ff0000,0000ff,00ff00|opacity=0.4/b/?lat=37.9&lng=-122.2&width=640&height=480&zoom=10
```

Here's an excerpt of the map, showing the different classes of place types, that is returned from the above URL (to see the full map, open a browser window and enter the path into the address field):



Note: For complete details on the purpose and valid values of class layer properties, both required and optional, refer to the **Class Layer** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/layer/Class.

8. REST for Thematic Layers

A thematic layer uses visual differences (color and/or opacity) to distinguish the geometries of features (e.g. counties, states, etc.) with quantitatively different values of a given numeric attribute. These differences relate to variations in the value of the selected property, but instead of classes based on an abstract definition (e.g. “chartered city”, “unincorporated,” etc.), the property is numeric and the differences are quantitative. For example, if the property you are interested in mapping is median household income, you might use a thematic layer with five colors for five different income ranges (e.g. under $20,000, $20,000 to $50,000, $50,000 to $100,000, $100,000 to $200,000, and over $200,000).

Another application of thematic layers might be to map the relationship between location and voting history. The mapfluence data catalog, for example, includes information on the proportion of votes in various jurisdictions that were cast for the Democratic candidate in the 2008 presidential election. The following steps illustrate how to map election results in a thematic layer with counties divided into six color categories based on their proportion:

1. Start with the host, version, API key, and endpoint:

   ```
   http://query.mapfluence.com/2.0/MFDOCS/map
   ```

2. Add the layer definition, which means specifying the values of the required layer properties, separating each property value pair with a pipe:
   - `mode=theme`: the layer type, in this case `theme`.
   - `select=umi.us_elections.2008_pres.prt_dem_pty`: the source of the data by which the geometries will be categorized, in this case an attribute table containing the proportion of the vote that went to the candidate of the Democratic party in the 2008 presidential election.
   - `from=umi.us_census00.county_geometry`: the source of the geometries that will be overlaid, which is the geometry table in the U.S. Census dataset covering U.S. counties.
   - `breaks=0.33,0.45,0.5,0.55,0.67`: a list of break points within the valid range of numeric values for the attribute, in this case the proportion of voters who voted for the Democratic candidate. These five breaks create six categories of geometries (counties) based on the percent of voters voting Democratic: less than 33%, 33-45%, 45-50, 50-55%, 55-67%, and over 67%.
   - `border=808080_1`: the color and line weight of the border, which shows the boundaries between geometries.
   - `styles=ff0000,ff8000,ffff00,00ff00,008080,0000ff`: a list of colors (red, orange, yellow, green, teal, blue) that will be used to render the categories of geometries defined by `breaks`.
   - `opacity=0.5`: the overall opacity of the layer. This is an alternative to adding an alpha channel specification to the end of each color in the `styles` property.
   Note: In a thematic layer, `styles` is plural.

```
/mode=theme|select=umi.us_elections.2008_pres.prt_dem_pty|from=umi.us_census00.county_geometry|breaks=0.33,0.45,0.55,0.67|border=808080_1|styles=ff0000,ff8000,ffff00,00ff80,0000ff|opacity=0.5
```

3. Add the base map code (in this case for Google Maps) after the layer definition:

```
/g
```

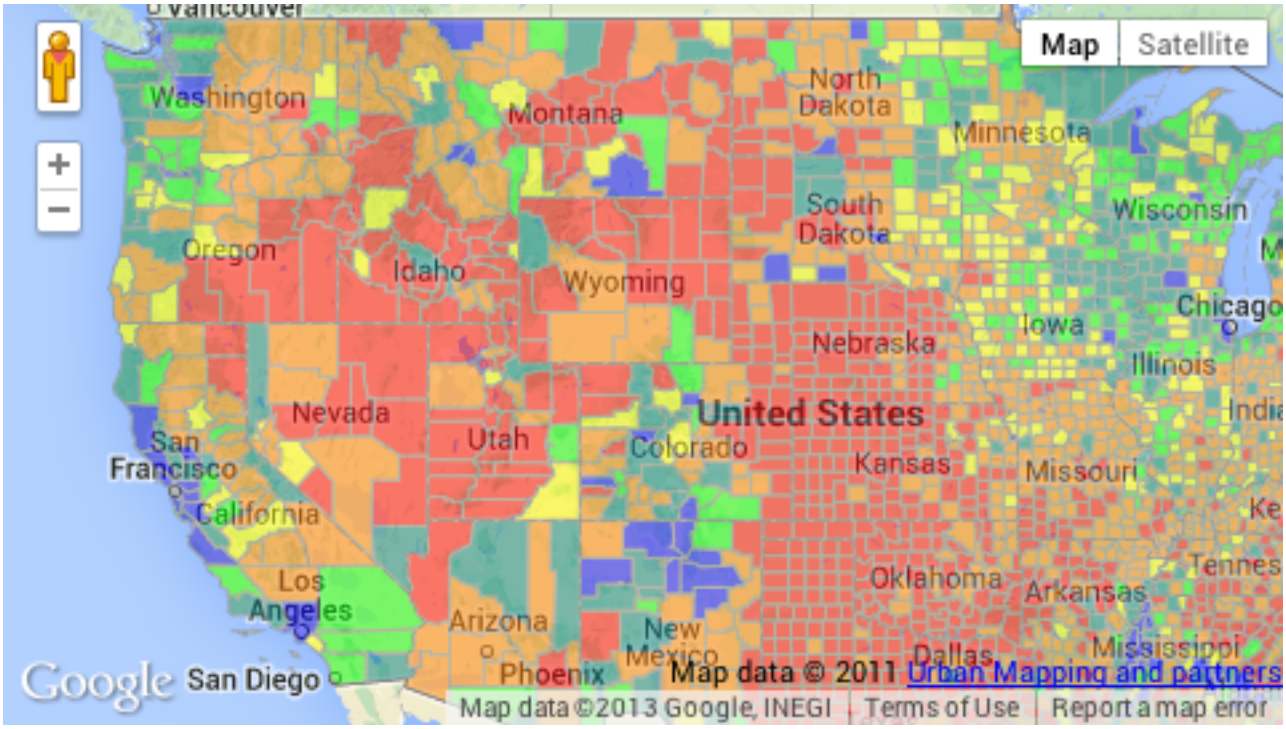
4. Add a query string after the base map code to specify the map's centerpoint, size, and zoom:

```
/?lat=39&lng=-96&width=800&height=480&zoom=4
```

5. Put it all together and you have the complete REST path for the map:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=theme|select=umi.us_elections.2008_pres.prt_dem_pty|from=umi.us_census00.county_geometry|breaks=0.33,0.45,0.55,0.67|border=808080_1|styles=ff0000,ff8000,ffff00,00ff80,0000ff|opacity=0.5/g/?lat=39&lng=-96&width=800&height=480&zoom=4
```

Here's an excerpt of the map, showing the proportion of voters in each county that voted Democratic in the 2008 presidential election, that is returned from the above URL (to see the full map, open a browser window and enter the path into the address field):



Note: For complete details on the purpose and valid values of thematic layer properties, both required and optional, refer to the **Thematic Layer** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/layer/Thematic.

9. REST for Heatmap Layers

A heatmap layer displays ranges of point density (location of selected type of event or feature), with different density ranges displayed using different color gradients. The number of ranges and the color gradient within each range are both derived from the number of color values specified with the `colors` property.

To determine the ranges and gradients, first the point density information is rounded to a scale of 0-255. Next the number of colors specified in the colors property is evaluated to determine the number of density ranges, with each consecutive color value pair in the `colors` string defining one range. If, for example, there are five colors values there would be four ranges (range 1 for values 1-2, 2 for 2-3, 3 for 3-4, and 4 for 4-5). The densities are then divided between these density ranges (densities 0-63 to density range 1, 64-127 to 2, 128-191 to 3, 192-255 to 4). Then each range is assigned a color gradient such that the densities in each range are represented by a gradient between the corresponding pairs of colors in the colors property (the gradient for density range 1 being between color 1 and color 2, for density range 2 being between color 2 and color 3, etc.).

The following steps illustrate how to create a heatmap layer, in this case using data from the Mapfluence data catalog on the concentration of fatalities from tornado touchdowns:

1. Start with the host, version, API key, and endpoint:

```
http://query.mapfluence.com/2.0/MFDOCS/map
```

2. Add the layer definition, which means specifying the values of the required layer properties, separating each property value pair with a pipe:
   - **`mode=heatmap`:** the layer type, in this case `heatmap`.
   - **`select=umi.noaa_tornado.attributes.num_fatal`:** the source of the event data from which point density is derived, in this case an attribute table from NOAA containing tornado touchdown fatalities.
   - **`from=umi.noaa_tornado.geometry`:** the source of the point geometries that will be overlaid, which is a geometry table in the NOAA dataset.
   - **`colors=00000000,00ff00,ffff00,ff8000,ff0000`:** a list of valid HTML color values. The first is hex RGBA, with the alpha channel given as `00` to specify full transparency. The remaining colors are standard hex RGB.
   - **`opacity=0.5`:** the overall opacity of the layer. This is an alternative to adding an alpha channel specification to the end of each color in the `colors` property.

```
/mode=heatmap|select=umi.noaa_tornado.attributes.num_fatal|from=umi.noaa_tornado.geometry|colors=00000000,00ff00,ffff00,ff8000,ff0000|opacity=0.5
```

3. Add the base map code (in this case for Google Maps) after the layer definition:

```
/g
```

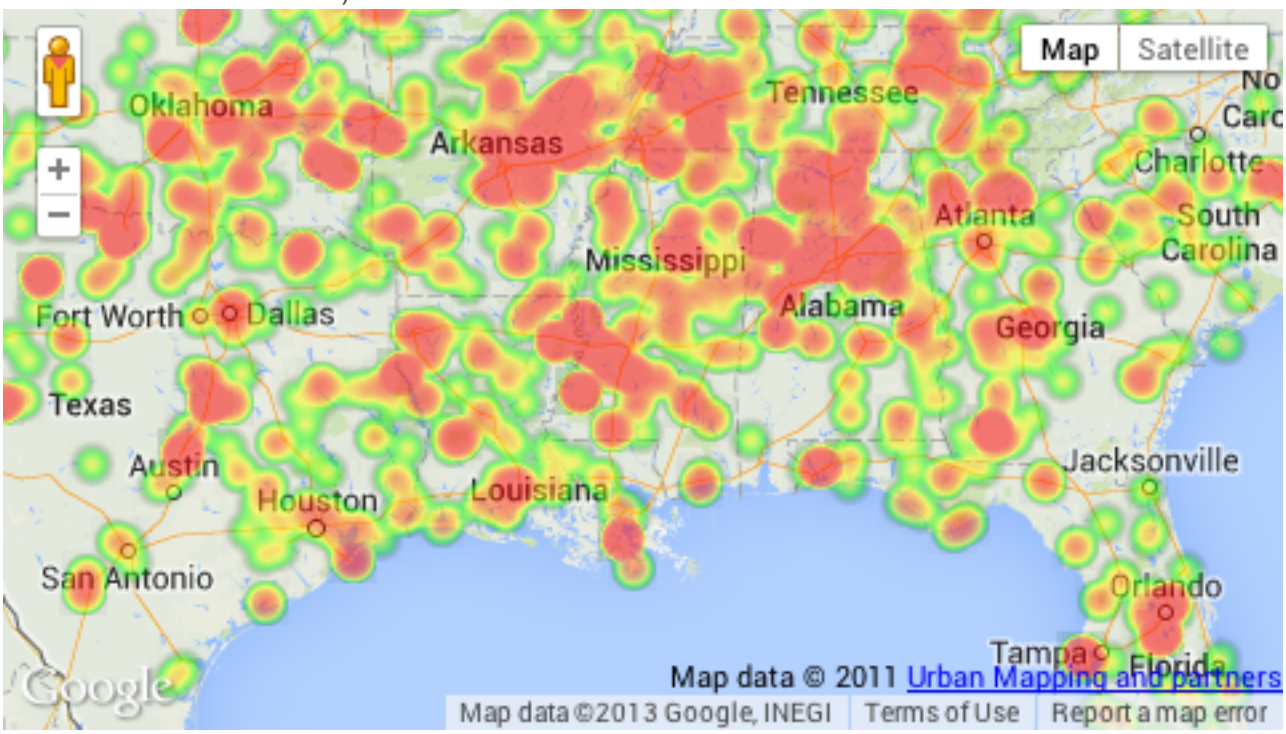
4. Add a query string after the base map code to specify the map's centerpoint, size, and zoom:

```
/?lat=38&lng=-90&width=720&height=640&zoom=5
```

5. Put it all together and you have the complete REST path for the map:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=heatmap|select=umi.noaa_tornado.attributes.num_fatal|from=umi.noaa_tornado.geometry|colors=00000000,00ff00,ffff00,ff8000,ff0000|opacity=0.5/g/?lat=38&lng=-90&width=720&height=640&zoom=5
```

Here's an excerpt of the map, showing the concentration of fatalities from tornado touchdowns, that is returned from the above URL (to see the full map, open a browser window and enter the URL into the address field):



Note: For complete details on the purpose and valid values of heatmap layer properties, both required and optional, refer to the **Heatmap Layer** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/layer/Heatmap.

10. REST for Label Layers

A label layer overlays text from a selected source onto an underlying map, positioning the text to label specified geometries (e.g. U.S. states, cities, roads, etc.).

Since a label layer is typically used to identify features of another layer, the following steps illustrate how to create a map with a both a label layer and a simple layer, in this case using a Mapfluence dataset that identifies the boundaries of areas that NOAA has defined as climate divisions:

1. Start with the host, version, API key, and endpoint:

```
http://query.mapfluence.com/2.0/MFDOCS/map
```

2. Add the layer definition for the simple layer, separating each property value pair with a pipe:
   - mode=simple: the layer type, in this case `simple`.
   - from=umi.noaa_boundaries.climate_division_geometry: the source of the geometries that will be overlaid, which in this case are the NOAA Climate Division Boundaries geometries that we are going to label.
   - border=802020_1: the color (in this case a dark red) and line weight of the border, which shows the boundaries between climate divisions.
   - style=00000000: the color and transparency, stated as RGBA expressed in hex, that will be used to render the geometries. In this case, since the last two digits are 00, the alpha channel is set to fully transparent (geometries are rendered as clear) and the first six digits don't have any effect as long as they represent a valid HTML hex color.
   Note: In a simple layer, `style` is singular.

```
/mode=simple|umi.noaa_boundaries.climate_division_geometry|border=802020_1|st
yle=00000000
```

3. Add the layer definition for the label layer, which means specifying the values of the required layer properties, separating each property value pair with a pipe. We specify the label layer first because we want it rendered on top of the simple layer.
   Note: The optional opacity property is not specified because default opacity is 1 (full) which is what one might typically want for labels:
   - mode=label: the layer type, in this case `label`.
   - select=name: the source of the label data, in this case the `name` column (with display name of "Division Name") of the NOAA Climate Division Boundaries attribute table in the Climate Division Boundaries dataset.
   - from=umi.noaa_boundaries.climate_division_geometry: the source of the geometries over which each label will be overlaid, which is the geometry table in the Climate Division Boundaries dataset.
   - face_name=FreeSans Bold: the name of the font face to use for the labels. The default font face for labels is DejaVu Sans Book. An alternative face may be chosen from the list of available fonts at http://query.mapfluence.com/2.0/fonts.html.

- size=16: the font size.
- fill=a02828: the color of the label (in this case a medium red), stated as a valid HTML RGB hex expression.

```
/mode=label|select=name|from=umi.noaa_boundaries.climate_division_geometry|face_name=FreeSans Bold|size=16|fill=a02828
```

4. Add the base map code (in this case for Google Maps) after the last layer definition:

```
/g
```

5. Add a query string after the base map code to specify the map's centerpoint, size, and zoom, which we use in this case to focus the map on Florida:

```
?lat=27.7&lng=-83.52&width=760&height=720&zoom=7
```

6. Put it all together and you have the complete REST path for the map:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=simple|umi.noaa_boundaries.climate_division_geometry|border=802020_1|style=00000000/mode=label|select=name|from=umi.noaa_boundaries.climate_division_geometry|face_name=FreeSans Bold|size=16|fill=a02828/g/?lat=27.7&lng=-83.52&width=760&height=720&zoom=7
```

Here's an excerpt of the map, showing NOAA climate divisions in the American Southeast, that is returned from the above URL (to see the full map, open a browser window and enter the URL into the address field):

Note: For complete details on the purpose and valid values of label layer properties, both required and optional, refer to the **Label Layer** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/layer/Label.

11. REST for Named Layers

In addition to the fully-featured base maps from Bing Maps, Google Maps, and Mapfluence, Mapfluence includes a set of "named layers" that enable one or more features of the Mapfluence base map (e.g. roads, place names, boundaries, etc.) to be overlaid as a layer. The following named layers are available:

- Land Cover
- Roads
- Road Labels
- Country Boundaries
- Country Labels
- State/Province Boundaries
- State/Province Labels
- Place Boundaries
- Place Labels
- Composite of all of the above layers

Named layers are typically used in two types of situations:

- To restore the visibility of base map features that are obscured when overlaid geometries are not fully transparent.
- To create the appearance of a base map that has fewer features than the standard base map.

Note: For best appearance, use the default (Mapfluence) base map when using named layers.

The following steps illustrate how to create a map with both named and class layers. Suppose, for example, that we want to see if there's any difference in the concentration of roads in the different types of U.S. Census Bureau place classifications (city, town, or CDP), but we don't care about the names of the places or roads themselves. We'll show the census places with a class layer. Beneath the class layer we'll put the named layer for land cover, which will (at default full opacity) cover all land areas on the base map, thereby removing from view the base map's cities, roads, place names, etc. Over the class layer we'll put a named layer for roads. We won't use a base map specifier, so the base map (only the water areas of which will be visible) will default to Mapfluence.

1. Start with the host, version, API key, and endpoint. At this point the rendered result would show only the Mapfluence base layer:

```
http://query.mapfluence.com/2.0/MFDOCS/map
```

2. Add a layer definition (separating each property value pair with a pipe) for the land-only named layer. If this layer definition is appended to the basic path above, the rendered result will no longer show any features other than basic land areas:

```
/mode=named|from=umi_land
```

3. Add a class layer definition. After appending this layer definition to the path from the two steps above, the rendered result will show the three types of places as colored semi-transparent overlays.
   - **`mode=class`:** the layer type, in this case `class`.
   - **`select=umi.us_census00.place.place_type`:** the source of the attribute by which the geometries will be classified, in this case an attribute table classifying places by type (city, town, or CDP).
   - **`from=umi.us_census00.place_geometry`:** the source of the geometries that will be overlaid, which is the geometry table in the Mapfluence dataset covering U.S. places.
   - **`date=2009-01-01T08:00:00Z`:** a date, which is used to filter the results using the date attribute in the specified geometry table (the `from` property) and attribute table (the `select` property).
   - **`values=city,town,CDP`:** a list of attribute values that are to be included in the map, in this case the types (classes) of place.
   - **`border=000000_1.3`:** the color and line weight of the border, which shows the boundaries between geometries.
   - **`styles=c00000,00c000,0000c0`:** a list of colors that will be used to render the geometry associated with the value at the same index in the values list (e.g. ff0000 for CDPs, 00ff00 for cities, 0000ff for towns). These colors could alternatively be expressed as valid HTML color names (red, green, blue).
   - **`opacity=0.4`:** the overall opacity of the layer. This is an alternative to adding an alpha channel specification to the end of each color in the `styles` property.

   Note: In a class layer, `styles` is plural.

```
/mode=class|select=umi.us_census00.place.place_type|from=umi.us_census00.place_geometry|date=2009-01-01T08:00:00Z|values=city,town,CDP|border=000000_1.3|styles=ff0000,0000ff,00ff00|opacity=0.4
```

4. Add a layer definition for the roads-only named layer. After appending this layer definition to the path from the three steps above, the rendered result will show overlaid roads.

```
/mode=named|from=umi_road
```

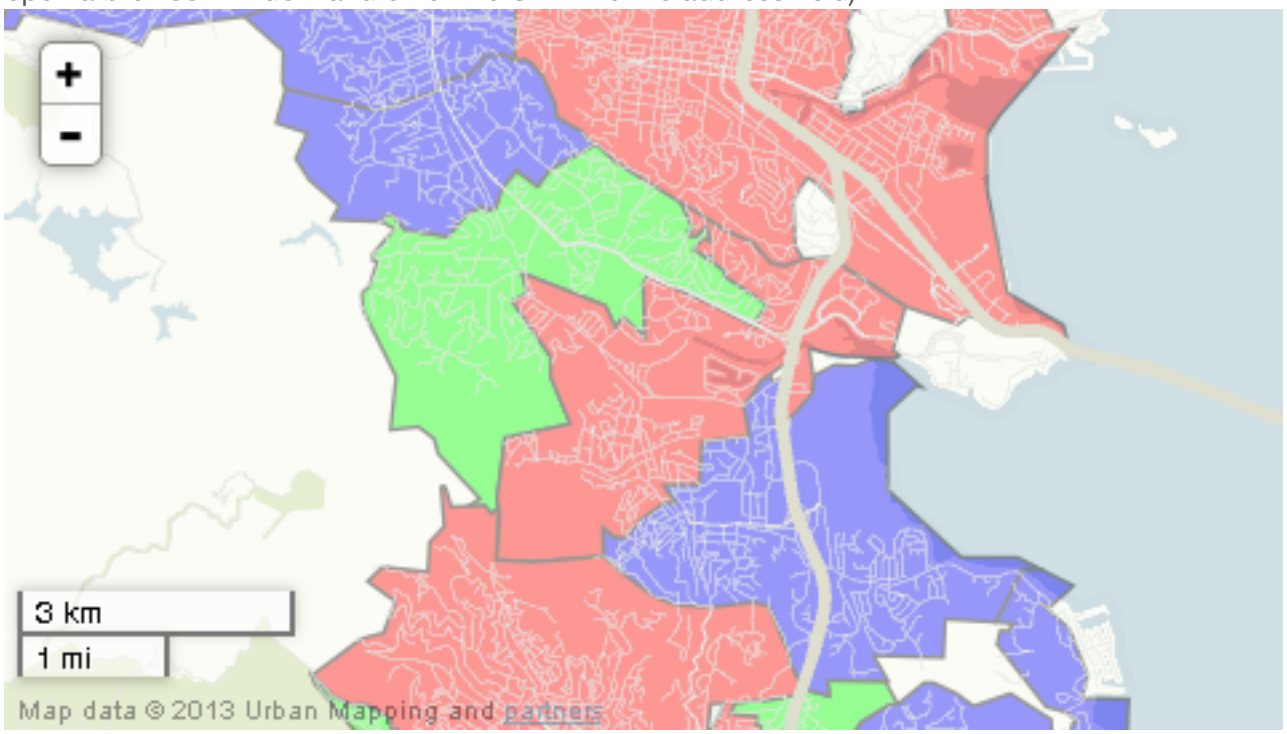
5. Add a query string after the base map code to specify the map's centerpoint, size, and zoom, which we use in this case to focus the map on Marin County, in the San Francisco Bay Area:

```
?lat=37.95&lng=-122.57&width=800&height=600&zoom=12
```

6. Put it all together and you have the complete REST path for the map:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=named|from=umi_land/mode=class|select=umi.us_census00.place.place_type|from=umi.us_census00.place_geometry|date=2009-01-01T08:00:00Z|values=city,town,CDP|border=000000_1.3|styles=ff0000,0000ff,00ff00|opacity=0.4/mode=named|from=umi_road/?lat=37.95&lng=-122.57&width=800&height=600&zoom=12
```

Here's an excerpt of the map, showing the Roads named layer overlayed on the place-type class layer and the Mapfluence base map, that is returned from the above URL (to see the full map, open a browser window and enter the URL into the address field):



Note: For complete details on the purpose and valid values of named layer properties, both required and optional, refer to the **Named Layer** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/layer/Named.

12. REST for Legends

A legend provides a key to the information displayed on a simple, class, or thematic layer. Consider, for example, our map of the proportion of 2008 votes in each county that were for the Democratic presidential candidate (see **REST for Thematic Layers**). Each county is represented by one of six colors (red, orange, yellow, green, teal, blue) corresponding to the proportion from low to high. But there's no indication on the map of what those colors mean. The following steps illustrate how to add a legend providing that information:

1. Recreate the path for the original map by starting with the host, version, API key, and endpoint, then adding the layer definition, the base map code, and the query string that specifies the map's centerpoint, size, and zoom:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=theme|select=umi.us_elections
.2008_pres.prt_dem_pty|from=umi.us_census00.county_geometry|breaks=0.33,0.45,
0.55,0.67|border=808080_1|styles=ff0000,ff8000,ffff00,00ff80,0000ff|opacity=0
.5/g/?lat=39&lng=-96&width=800&height=480&zoom=4
```

2. At end of the query string, append the Boolean legend parameter and set the value to 1 (enabled):

```
/?lat=39&lng=-96&width=800&height=480&zoom=4&legend=1
```

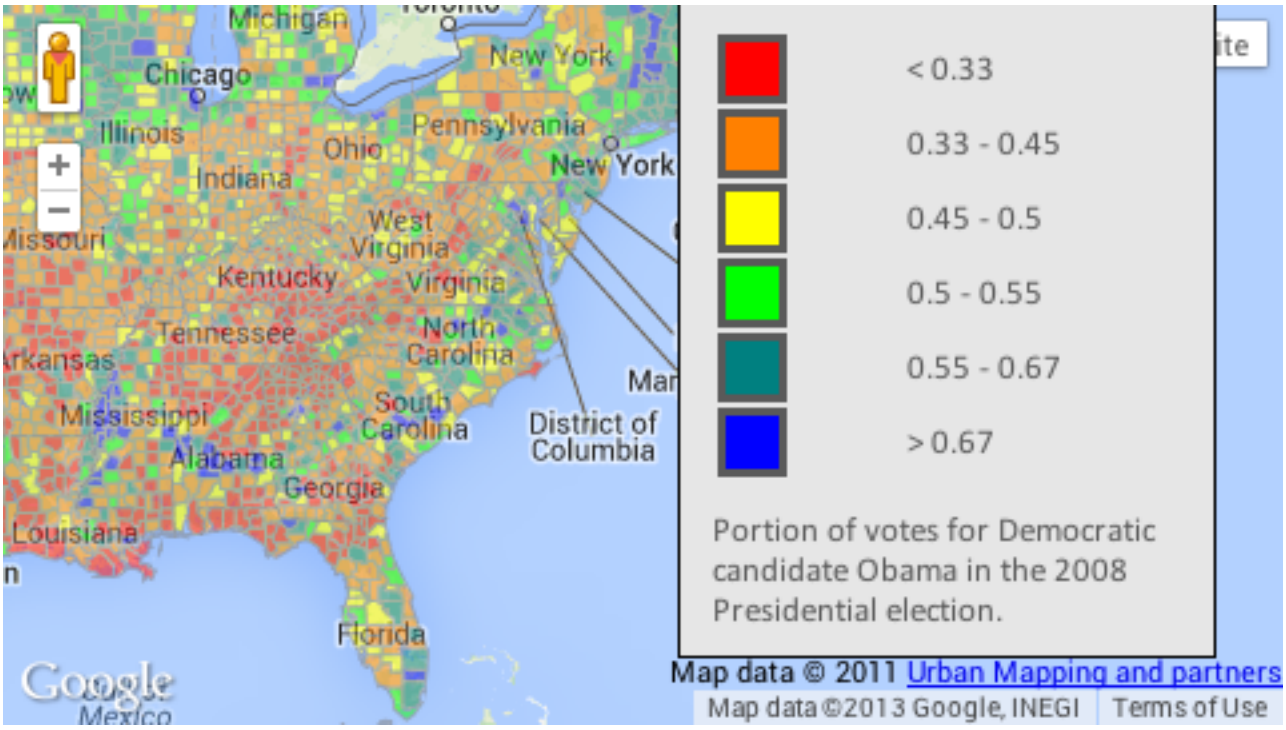
2. In the query string, change the value of the longitude and width parameters so that when the map is rendered the legend will not obscure any part of the United States:
   - shift the centerpoint East by increasing `lng`;
   - increase `width`.

```
/?lat=39&lng=-88&width=975&height=480&zoom=4&legend=1
```

4. Put it all together and you have the complete REST path for the map:

```
http://query.mapfluence.com/2.0/MFDOCS/map/mode=theme|select=umi.us_elections
.2008_pres.prt_dem_pty|from=umi.us_census00.county_geometry|breaks=0.33,0.45,
0.55,0.67|border=808080_1|styles=ff0000,ff8000,ffff00,00ff80,0000ff|opacity=0
.5/g/?lat=39&lng=-88&width=975&height=480&zoom=4&legend=1
```

Here's an excerpt of the map, showing a legend, that is returned from the above URL (to see the full map, open a browser window and enter the URL into the address field):



Note: For complete details on the purpose and valid values of legend layer properties, both required and optional, refer to the **/legend** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/Legend.

13. REST for Tiles

In some circumstances you may wish to return a static image of a portion of a map rather than a slippy map that can be zoomed and dragged (such as those built in the preceding sections). The process of specifying a tile is similar to that of building a regular map, but instead of ending the path with an optional query string (for zoom, size, centerpoint, etc.) you specify (non-optionally) the type, zoom, and coordinates as a continuation of the path.

The coordinate and zoom specification for tiles is completely different than for maps. A full explanation is beyond the scope of this document (see http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames) but the basic concepts are as follows:

- A tile is a 256 × 256 pixel image (PNG file) showing one part of a map of the entire world.
- A tile is specified as /zoom (z) /horizontal coordinate (x) /vertical coordinate (y), e.g. `/8/16/32`.
- Zoom subdivides the map into a grid and thereby determines how much of the world map fits into a single tile.
- Coordinates identify the horizontal and vertical location of an individual tile within the overall grid. The upper left tile always has coordinates of `/0/0`. The higher the zoom, the more sections the map is subdivided into, and thus the higher the maximum valid value for coordinates.
- The valid zoom range is 0-18. The higher the zoom, the higher the granularity of the grid:
  - At zoom 0, the grid is 1 by 1, and a tile includes the entire world map. The specification for the one and only tile would be `/0/0/0`.
  - At zoom 1, the number of subdivisions is doubled in both the horizontal and vertical dimensions, resulting in a 2 by 2 grid with four tiles total. The specification for the bottom right tile would be `/1/1/1`.
  - At zoom 2, the number of subdivisions is doubled again, resulting in a 4 by 4 grid with 16 tiles total. The specification for the bottom right tile would be `/2/3/3`.
  - At zoom 3, the number of subdivisions is doubled again, resulting in an 8 by 8 grid with 64 tiles total. The specification for the bottom right tile would be `/3/7/7`.
  - And so on...

Note: An alternative approach to generating a static (non-slippy) map tile, using a query string with zoom, latitude, and longitude, is the Mapfluence static Map (see **/static** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/static-map).

The following steps illustrate how to specify a tile using a named layer (the base map is not specified so the default Mapfluence base map will be used):

1. Start with the host, version, API key, and endpoint. Since we are making a tile rather than a map, the endpoint will be `tile`:

```
http://query.mapfluence.com/2.0/MFDOCS/tile
```

2. Add a layer definition (separating each property value pair with a pipe) for a named layer, in this case the layer that shows all features of a Mapfluence map (essentially the base map, but used as a layer):

```
/mode=named|from=umi
```

3. Add a definition of the desired zoom:

```
/11
```

4. Add a definition of the desired coordinates:

```
/327/791.png
```

5. Put it all together and you have the complete REST path for the tile:

```
http://query.mapfluence.com/2.0/MFDOCS/tile/mode=named|from=umi/11/327/791.png
```

To experiment with different tiles, open a browser window, paste the URL into the address field, and change the layer, zoom, and/or coordinates values. Here's the tile that is returned from the above URL:

Note: For complete details on the purpose and valid values of tile properties, both required and optional, refer to the **/tile** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/tile/Tile.

14. REST for Spatial Queries

Spatial queries are used for operations that don't involve mapping; instead they return text, typically JSON. These queries are distinct from data catalog queries in that they are focused on returning the actual data within the Mapfluence data catalog (or, if enabled, a custom dataset) rather than returning metadata about the catalog.

Spatial queries retrieve data related to a feature, such as a county or state, that is identified with the `from` property. The data to retrieve about the feature is specified with the `select` property, which points to one or more data sources. Each source is one of the following:

- The feature fields of the geometry table in which the feature exists (see **Feature Fields** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/select/#guide-select-feature-fields).
- An attribute column in an attribute table in which there are attribute records related to features meeting the specified criteria. The field values may be returned individually or aggregated across rows (records) using an aggregate expression (sum, count, etc.; see **Expressions** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/expressions).

In the following sections we'll use Mapfluence REST to perform two different kinds of spatial queries:

- Expression Spatial Query
- Attribute and Field Query

14.1 Expression Spatial Query

The following steps illustrate how to specify a simple spatial query using an expression. In this case the query returns a count of the states whose name contained the string "al" as of the specified date:

1. Start with the host, version, and API key (substitute your own key):

   ```
   http://query.mapfluence.com/2.0/MFDOCS
   ```

2. Add the query statement:

   ```
   /spatialquery.json
   ```

3. Add the query definition in the form of a query string, in this case specifying the following properties:
   - `from=umi.us_census00.state_geometry`: the geometry table containing the features (in this case states) about which the data is to be retrieved.
   - `select=count(*)`: the data to retrieve, in this case a count (a form of aggregate expression)

of the rows in the attribute table that meet the criteria specified with the `where` property.
- where=name__icontains:'al': an optional filter that may be applied to narrow the results returned from the query. In this case we want the count to include only states with "al" in the field of the attribute table column "name."
- date=2010-08-01T07:00:00Z: The datetime for which the information is retrieved (if a new state had been added to the Union after August 1, 2009 at 7AM local time, it would not be counted in this query).

```
?from=umi.us_census00.state_geometry&select=count(*)&where=name__icontains:'al'&date=2010-08-01T07:00:00Z
```

4. Put all the pieces together and you have the complete REST path for the query:

```
http://query.mapfluence.com/2.0/MFDOCS/spatialquery.json?from=umi.us_census00.state_geometry&select=count(*)&where=name__icontains:'al'&date=2010-08-01T07:00:00Z
```

This query returns the following JSON key/value pair (to see the complete response body, open a browser window and enter the URL into the address field):

```
{
   "count(*)": 3
}
```

Notes:
» For information on selecting data to retrieve from the three types of sources see **Retrieve data with select statements** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/select/.
» In a where statement (filter), two underscores "__" separate the attribute or field (e.g. `name`) and the operator (e.g. `icontains`).
» For information on the filters that may be used in queries see **Filter data using where statements** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/where.
» For information on using dates in queries, see **Dealing with dates** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/guides/dates.
» For complete details on the purpose and valid values of spatial query properties, both required and optional, refer to the **/spatialquery.json** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/query/SpatialQuery.

14.2 Attribute and Field Query

The following steps illustrate how to specify a spatial query using an attribute and two feature fields. In this case the query returns the names and centroid coordinates of U.S. counties in which average annual snowfall exceeded 100 inches as of the specified date:

1. Start with the host, version, API key (substitute your own key), and query statement:

```
http://query.mapfluence.com/2.0/MFDOCS/spatialquery.json
```

2. Add the query definition in the form of a query string, in this case specifying the following properties:
   - from=umi.us_census00.county_geometry: the geometry table containing the features (in this case counties) about which the data is to be retrieved.
   - select=umi.snowfall_monthly.attributes.annualavg, centroid, name: the data to retrieve about the features (counties), in this case the average annual snowfall attribute (from an attribute table with data on snowfall), and the feature fields `name` and `centroid` (coordinates of the geographical center of the county).
   - where=umi.snowfall_monthly.attributes.annualavg_gt:100: an optional filter that may be applied to narrow the results returned from the query (thereby avoiding the error `TooManyResultsError`). In this case we want the count to include only counties where average annual snowfall is greater than 100 inches.
   - date=2010-08-01T07:00:00Z: The datetime for which the information is to be retrieved (August 1, 2009 at 7AM local time).

```
?from=umi.us_census00.county_geometry&select=umi.snowfall_monthly.attributes.annualavg,centroid,name&where=umi.snowfall_monthly.attributes.annualavg__gt:100&date=2010-08-01T07:00:00Z
```

3. Put the pieces together and you have the complete REST path for the query:

```
http://query.mapfluence.com/2.0/MFDOCS/spatialquery.json?from=umi.us_census00.county_geometry&select=umi.snowfall_monthly.attributes.annualavg, centroid, name&where=umi.snowfall_monthly.attributes.annualavg__gt:100&date=2010-08-01T07:00:00Z
```

This query returns a FeatureCollection JSON array of 44 features (counties) meeting the specified criteria. The following example shows the array member for the first county in the list, which gives the county's centroid (a point geometry), its average annual snowfall, and its name (to see the complete response body, open a browser window and enter the URL into the address field):


```
{
   "geometry": {
      "type": "Point",
      "coordinates": [
         -89.514786,
         46.0529009
      ]
   },
   "type": "Feature",
   "properties": {
      "umi.snowfall_monthly.attributes.annualavg": 144.1,
      "name": "Vilas"
   }
}
```


Note: See query-related notes above.

15. REST for Estimate Queries

An estimate query computes an estimated value for a feature attribute whose actual value is not known. The estimate is based on the known values of the same attribute in the areas surrounding the feature for which the estimate is requested. An appropriate estimation method is chosen depending on the type of the attribute that is being estimated.

Consider, for example, the case of estimating the number of households within a given ZIP code when the attributes of the ZIP code geometry table do not include the number of households within the area of each ZIP code. One solution is to get an estimate calculated from the number-of-households attribute of the census block groups whose geometries fall within the target ZIP code.

The following steps illustrate how to specify a simple estimate operation, in this case estimating the number of households within a given ZIP code:

1. Start with the host, version, and API key (substitute your own key):

   ```
   http://query.mapfluence.com/2.0/MFDOCS
   ```

2. Add the query statement:

   ```
   /estimate.json
   ```

3. Add the query definition in the form of a query string, in this case specifying the following properties:
   - select=umi.us_census00.stats.cnt_hh: the attribute table column for which values are to be estimated, in this case the number of households from the US Census 2000 Data Tables.
   - in=umi.us_census00.zcta_geometry.07932@2010-01-01T01:00:00Z: the geometry of the area for which the estimate is to be calculated, in this case ZIP code 07932.
   - using=umi.us_census00.blkgrp_geometry: the geometry table that will be used to find what features are spatially contained within the geometry specified with the `in` property.
   - date=2010-01-01T01:00:00Z: A filter that narrows the attribute records used in the estimation to those whose time span includes the specified datetime.

   ```
   ?select=umi.us_census00.stats.cnt_hh&in=umi.us_census00.zcta_geometry.07932@2010-01-01T01:00:00Z&using=umi.us_census00.blkgrp_geometry&date=2010-01-01T01:00:00Z
   ```

4. Put all the pieces together and you have the complete REST path for the query:

   ```
   http://query.mapfluence.com/2.0/MFDOCS/estimate.json?select=umi.us_census00.stats.cnt_hh&in=umi.us_census00.zcta_geometry.07932@2010-01-01T01:00:00Z&using=umi.us_census00.blkgrp_geometry&date=2010-01-01T01:00:00Z
   ```

This query returns the following JSON key/value pair (to see the complete response body, open a browser window and enter the URL into the address field):

```
{
   "umi.us_census00.stats.cnt_hh": 2936
}
```

Notes:

» An alternative to specifying the `in` geometry with GeoJSON is to use a feature ID (see **/catalog/feature/<ID>** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/catalog/Feature).

» For complete details on the purpose and valid values of spatial query properties, both required and optional, refer to the **/estimate.json** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/query/EstimateQuery.

16. REST for Spatial Operations

A spatial operation query is a utility query that creates a new geometry by combining two or more geometries using a spatial operator (intersection, union or extent).

The following steps illustrate how to specify a simple spatial operation, in this case taking two GeoJSON points and combining them with the union operator to return a new GeoJSON multipoint object:

1. Start with the host, version, and API key (substitute your own key):

```
http://query.mapfluence.com/2.0/MFDOCS
```

2. Add the query statement:

```
/spatialops.json
```

3. Add the query definition in the form of a query string, in this case specifying the following properties:
   - **select={"type": "point", "coordinates": [-122, 37]},{"type": "point", "coordinates": [-123, 36]}:** the geometries on which to perform the operation, in this case a set of two points at the specified latitude and longitude coordinates. .
   - **using=union:** the type of spatial operation to perform, in this case a union.

```
?select={"type": "point", "coordinates": [-122, 37]},{"type": "point",
"coordinates": [-123, 36]}&using=union
```

4. Put all the pieces together and you have the complete REST path for the query:

```
http://query.mapfluence.com/2.0/MFDOCS/spatialops.json?select={"type":
"point", "coordinates": [-122, 37]},{"type": "point", "coordinates": [-123,
36]}&using=union
```

This query returns the following GeoJSON structure representing the newly-created Multipoint geometry (to see the complete response body, open a browser window and enter the URL into the address field):

```
{
   "type": "MultiPoint",
   "coordinates": [
      [
         -123,
         36
      ],
      [
         -122,
         37
```

```
        ]
    ]
}
```

Notes:

» An alternative to specifying points with GeoJSON is to use feature IDs (see **/catalog/feature/<ID>** at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/catalog/Feature).

» For complete details on the purpose and valid values of spatial query properties, both required and optional, refer to the **/spatialops.json** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/query/SpatialOperation.

17. REST for Geocoding

A geocoding query is a utility query that takes an address, or part of an address, and returns estimated latitude and longitude coordinates of the specified location. Latitude and longitude are returned for the following types of geocoding query requests:

- Full address
- Intersection (two cross streets)
- Place by name (i.e. Tokyo): returns the place's centroid.
- Street: returns a series of coordinates.

The following steps illustrate how to specify a geocoding operation, in this case for a complete street address:

1. Start with the host, version, and API key (substitute your own key):

```
http://query.mapfluence.com/2.0/MFDOCS
```

2. Add the query statement:

```
/spatialquery.json
```

3. Add the query definition in the form of a query string, in this case specifying the following properties (with "+" substituted for spaces):
   - **street=690+Fifth+St.:** the street address.
   - **city=San+Francisco:** the city.
   - **state=CA:** the state code.
   - **postalcode=94107:** the postal code.

```
?street=690+Fifth+St.&city=San+Francisco&state=CA&postalcode=94107
```

4. Put all the pieces together and you have the complete REST path for the query:

```
http://query.mapfluence.com/2.0/MFDOCS/geocoder.json?street=690+Fifth+St.&cit
y=San+Francisco&state=CA&postalcode=94107
```

This query returns the following JSON structure with the lattitude and longitude (to see the complete response body, open a browser window and enter the URL into the address field):

```
[
   {
      "lat": 37.7753486633,
      "lon": -122.397674561
   }
]
```

Note: For complete details on the purpose and valid values of spatial query properties, both required and optional, refer to the **/geocoder.json** page of the Mapfluence REST API Reference at http://developer.urbanmapping.com/docs/mapfluence/rest/2.0/reference/query/GeocodeQuery.