Data Analysis using Maps

in Logi Info v12.2

March 2017
# Table of Contents

Introduction .................................................................................................................. 1

Using Animated Maps .................................................................................................. 2
  The Animated Map Element ...................................................................................... 2
  Referencing Data ....................................................................................................... 5

Adding Color Ranges .................................................................................................... 8
  Using a Color Spectrum Legend ............................................................................. 10
  Available Maps ........................................................................................................ 11

Introducing Google Maps ............................................................................................ 15

About Web Services ...................................................................................................... 18
  API Licensing and Use .......................................................................................... 19

System Requirements ................................................................................................. 19

The Google Map Family of Elements ........................................................................... 19

Refreshing Google Maps ............................................................................................ 26

About Mapping Data ................................................................................................... 26

Google Map Tutorial ................................................................................................... 29

Create the Basic Application ......................................................................................... 29

Configure the Web Service Connection ....................................................................... 29

Create the Basic Google Map ....................................................................................... 30

Adding Interactive Map Features ................................................................................ 32
  Put Controls on the Map ......................................................................................... 32
  Switch Map Types .................................................................................................... 33
  Display a Custom Map Marker ................................................................................ 34

Geocoding Data ........................................................................................................... 35

Examine the Geocoded Data ....................................................................................... 37

The “Include Condition” Attribute ............................................................................ 39

Reverse Geocoding ..................................................................................................... 39

Adding an Info Panel .................................................................................................. 39

Adding Finishing Touches ........................................................................................... 42

Using Google Map Polygons ....................................................................................... 45
  Geographic Area Boundary Data ........................................................................... 46
  General Features ..................................................................................................... 46

Understanding the Data ............................................................................................... 47

Creating a Google Map with Polygons ....................................................................... 48

Adding Drill-down Capabilities .................................................................................. 52
Tools for GIS Data Conversion ................................................................. 54
SHP2KML .................................................................................................. 54
GPSBabel ................................................................................................. 55
GPS Visualizer ....................................................................................... 55
ExpertGPS .............................................................................................. 55
Using Data from SQL Server ................................................................. 55
GIS Data Resources ............................................................................... 55
United States Census Bureau ............................................................... 55
Texas State Data Center ....................................................................... 55
UC Berkeley ......................................................................................... 56
University of North Carolina .............................................................. 56
Shapefile Resources ............................................................................. 56
Using Google Map Polylines ................................................................. 57
Geographic Data .................................................................................. 58
Understanding the Data ....................................................................... 58
Creating a Google Map with Polylines ............................................... 59
Using the Logi Google Map API ............................................................ 63
Events and Objects .............................................................................. 63
Y ........................................................................................................... 63
rdCreate.................................................................................................. 63
rdCreated................................................................................................ 64
rdGetMapObject() ............................................................................... 64
Example Definition ............................................................................ 65
Using the ESRI ArcGIS API ................................................................. 66
Licensing ............................................................................................... 66
Requirements Check List .................................................................... 66
Adding a Simple ArcGIS Map ............................................................... 67
Adding a Polygon Overlay ................................................................. 71
Embedding Logi Visualizations ............................................................ 74
Resources and Support ....................................................................... 81
Introduction

Logi managed reporting tools offer developers a rich selection of data visualizations. One of the most interesting is the ability to display geographic maps that are related to data. This publication introduces developers to the use of maps in Logi Info and Logi Report.

Logi Info is a tool that allows developers to create and run comprehensive reporting solutions and web applications, using a full range of business analytics features and components.

Documentation for Logi Analytics products, available on the Logi Developer Network web site, will always present the most up-to-date information. Specifically, you can look there for the newest features; publications like this one are by their nature more difficult to keep current.

Unlike products based on client-server technology, Logi Analytics’ products were conceived to be web-based from the start. They output only HTML and script to the end-user’s browser and as a result achieve simplicity, cross-browser compatibility, and an ease-of-use that eludes other products. At the same time, this means that some of the flashier functionality of desktop applications or downloaded custom components is not available. However, the resulting minimal configuration management requirements and tighter security of Logi products provides ample proof of the correctness of its approach, especially in an enterprise setting.

The DevNet web site includes numerous free sample applications that can be quickly downloaded and that provide a great learning opportunity for new developers. Frequently the sample apps mirror the examples in the documents on DevNet. Logi developers are urged to take advantage of these sample apps.

What should the Logi developer already know? A prospective Logi developer should already have a good understanding of general web technologies, such as HTML and style sheets, and be familiar with their data sources and the techniques, such as SQL, needed to access that data. Experience with JavaScript is useful but not required.

What information is not presented here? Developers must have an understanding of the data they want to work with and the techniques, such as SQL queries, required to access that data. The assumption is made that developers have these skills themselves or have access to others within their organizations with them and the specifics of these topics are not included here.

We hope you enjoy working with these products and quickly become productive and successful with them. Comments about this publication are welcomed and can be addressed to DevNet@Logianalytics.com.
Using Animated Maps

Logi Info’s Animated Maps are animated, interactive, data-driven, JavaScript-based maps for Logi applications. Developers can use them to display geographical data distributed by category, regions, or entities. The best usage examples include statistical display of data, flight routes, office locations, election results, survey results, or business intelligence such as “Revenue by Regions” or “Revenue by States”.

They do not work with zip codes or geocoded (latitude- & longitude-based) data.

Like all Logi data visualizations, map portions can be interactive links that can be clicked in order to "drill-down" to other reports, charts, and URLs.

These maps do not require Adobe’s Flash Player browser-extension be installed in order to view them. Maps for 320 countries and regions are included with Logi Info; see the list of maps included in a later section.

In almost all of these maps, their subdivisions (regions, provinces, states, etc.) are referenced by an internal numeric ID. In order to relate these ID numbers to "real world" names more likely to be encountered in your data, such as "Montana" or "Belgium", each map has a companion XML data file that can be used to cross-reference a region name to the internal ID. This is discussed in detail later in the Referencing Data section.

Animated Maps can be exported to PDF.

A sample application can be found in the DevNet Sample Applications page

The Animated Map Element

Animated Maps are included in Logi Info applications by adding the Animated Map element to a report definition.

In the example above, an Animated Map element has been added and its attributes set as shown.
The resulting map is shown above. When the cursor passes over a state, its color changes and a "tooltip" appears with the full state name. The attributes for the Animated Map element are explained below. Unless otherwise noted, attribute values are optional.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>(Required) Specifies the height of the map, in pixels.</td>
</tr>
<tr>
<td>Map Type</td>
<td>(Required) Specifies the type of base map to be rendered; e.g. World, USA, China, USARegion etc. A complete list is provided in a later chapter. Specific map type values can be found by examining the .js files in the application folder's rdTemplate\rdFusionMap folder. The India map, for example, is called FusionCharts.HC.india.js and the country or state name portion &quot;india&quot; is the value that's entered in this attribute.</td>
</tr>
<tr>
<td>Regional Name Column</td>
<td>(Required) Specifies the name of a column in the datalayer, with data representing the name for each region to be mapped. If a datalayer is not being used, enter an arbitrary placeholder value.</td>
</tr>
<tr>
<td>Regional Value Column</td>
<td>(Required) Specifies the name of a column in the datalayer, with data representing the value for each region to be mapped. If a datalayer is not being used, enter an arbitrary placeholder value.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Width</td>
<td>(Required) Specifies the width of the map, in pixels.</td>
</tr>
<tr>
<td>Background Color</td>
<td>Specifies the color of the map background; either a color name, decimal RGB value, or hex RGB value prefixed with a pound sign (#112233). &quot;Transparent&quot; is a valid value.</td>
</tr>
<tr>
<td>Border Color</td>
<td>Specifies the color of the map border lines can be a color name, a decimal RGB value, or a hex RGB value prefixed with a pound sign (#112233). &quot;Transparent&quot; is a valid value.</td>
</tr>
<tr>
<td>Color</td>
<td>Specifies the default color of the map; can be a color name, a decimal RGB value, or a hex RGB value prefixed with a pound sign (#112233). &quot;Transparent&quot; is a valid value.</td>
</tr>
<tr>
<td>Font</td>
<td>Specifies the family name of font in the map. Can be followed by a space and &quot;bold&quot; or &quot;italic&quot;.</td>
</tr>
<tr>
<td>Font Color</td>
<td>Specifies the color of the font in the map; can be a color name, a decimal RGB value, or a hex RGB value prefixed with a pound sign (#112233).</td>
</tr>
<tr>
<td>Font Size</td>
<td>Specifies the size, in points, of the font in the map.</td>
</tr>
<tr>
<td>Hover Color</td>
<td>Specifies the color of map regions when the mouse pointer is hovered over it.</td>
</tr>
<tr>
<td>ID</td>
<td>Specifies the element's unique identifier.</td>
</tr>
<tr>
<td>Include Value In Labels</td>
<td>Specifies, when working with a datalayer, whether to have data values appear in the labels along with region identifiers. Enter 1 to show the values.</td>
</tr>
<tr>
<td>Outer Border Color</td>
<td>Specifies the color of the outer border of the map canvas. Can be a color name, a decimal RGB value, or a hex RGB value prefixed with a pound sign (#112233).</td>
</tr>
<tr>
<td>Show Labels</td>
<td>Specifies whether region labels are displayed. Default: True</td>
</tr>
<tr>
<td>Tooltip Column</td>
<td>An automatically-generated tooltip is displayed when the cursor is hovered over each map region, consisting of the data in the Regional Name Column and Regional Value Column attribute values specified earlier, separated by a comma and space. This attribute, Tooltip Column, specifies the name of a datalayer column containing the text for a custom tooltip, which will be shown instead of the automatic tooltip if a value is entered here. You can specify the name of a column returned from the datasource, or that of a Calculated Column you add to the datalayer to create more customized text.</td>
</tr>
</tbody>
</table>
Referencing Data

It's nice to be able to show maps, but the Animated Map element becomes really useful when it associates data with map regions. To do this, a datalayer element is added to the definition.

In the following example, a map of Canada will be produced, showing its provinces with their abbreviations. When the cursor passes over a province, a "tooltip" will appear showing the full province name and the number of members in a mythical book club chapter from that province.

In the example definition above, a DataLayer.Static element has been added as a child of the Animated Map element, and some Static Data Row elements have been added to provide data values. Naturally, depending on the datasource, other types of datalayers can be used here.

A fragment of this map's companion XML data file, Canada.xml, is shown above. Note its highlighted attributes, "InternalID" and "LongName". In order to relate the Province value in our data to the appropriate region in the map, a datalayer Join can be used between the data and the companion XML file.
First, as shown above, the Animated Map element’s **Region Name Column** attribute value is set to the InternalID in the companion XML file. The **Regional Value Column** attribute is set to the MemberCount value in the static data.

Next, a **Join** element is added, as shown above. The Join element performs the equivalent of a SQL JOIN on two datalayers. In this case, an Inner Join will be used between the static datalayer and...

...a new **DataLayer.XML File** element which is added. This datalayer retrieves the data from the map’s companion XML file and its **XML File** attribute value is shown but has been truncated to fit on this page. The real value should always begin with:

```
@Function.AppPhysicalPath~\rdTemplate\rdFusionMap\Canada.xml
```

The companion XML file names correspond to those of the map type .js files (without the "FusionCharts.HC." prefix and with an .xml file extension). So, for example, the file `FusionCharts.HC.canada.js` corresponds to the companion XML file `Canada.xml`.
Finally, a **Match Condition** element is used to provide the JOIN criteria: the Province column value in the static datalayer must match the LongName column value in the companion XML file. Note that, as in any JOIN, the choice of which column is on the left and which is on the right is significant. Generally, the Left Data Column comes from the datalayer *above* the join and the Right Data Column from the datalayer *below* it.

Now when the map is displayed, as shown above, and the cursor is placed over a region, the **LongName** and **MemberCount** information appear in a tooltip.
"Drill-down" capability can be added by placing an Action element, as shown above, beneath the Animated Map element. Region-specific data can be accessed using @Chart tokens and passed, as Link Parameters, to the next report or process.

### Adding Color Ranges

Once data values have been associated with them, individual regions in an Animated Map can be color-coded based on those values.

This is accomplished, as shown above, by adding one or more Map Color Range elements beneath the Animated Map element. Each element represents one color, which is set in its Color attribute. The value of this attribute can be set using @Session, @Request, and @Local tokens, but not @Data or @Chart tokens. A legend is automatically generated for the map and the text in the DisplayValue attribute is used in it.

The assignment of a color to a particular map region is controlled by the Max Value and Min Value attributes; the color will be assigned if the data value is within this range. These values correspond to the data value identified in the Animated Map element's Region Value Column attribute.

⚠️ Note this important distinction: in order to prevent overlapping ranges, the color is assigned if the data is equal to or greater than the Min Value attribute value, and less than the Max Value attribute value, and you need to provide your range values accordingly. For example,
<table>
<thead>
<tr>
<th>Data Values</th>
<th>Min Value ( =&gt; )</th>
<th>Max Value ( &lt; )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 5</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>6 - 10</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>11 - n</td>
<td>11</td>
<td>n - 1</td>
</tr>
</tbody>
</table>

will accurately handle a data value of 10.5

And the results can be seen in the example shown above. If no text is entered in the DisplayValue attributes, the numeric minimum and maximum values will be displayed in the legend instead.
Using a Color Spectrum Legend

Instead of using Map Color Range elements, you can add a "color spectrum" legend to your animated map; this legend uses colors to illustrate the distribution of the data values used to generate the map and assigns those colors to the appropriate map regions.

The first step, as shown above, is to add a Color Spectrum Column element beneath the datalayer.

Configure its attributes to identify the data column used as the Animated Map element’s Region Value Column, and the desired colors for the low, medium, and high values.

The next step is to add an Animated Map Color Spectrum Legend element, as shown above, beneath the Animated Map and to set its caption.

The legend appears by default below the map, but its Location attribute can be set to make it appear to the right of the map.
And the resulting map and legend are shown above.

**Available Maps**

Approximately 320 maps are distributed with Logi Info and they can all be seen in the

```
<yourApplication>\rdTemplate\rdFusionMap
```

folder. The following table provides a partial listing of the maps distributed with Logi Info:

<table>
<thead>
<tr>
<th>Map Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>World</td>
<td>The World, with six major regions: North and Central America, South America, Europe, Africa, Asia, and Australia</td>
</tr>
<tr>
<td>World8</td>
<td>The World, with eight major regions: North America, Central America, South America, Europe, Africa, Middle East, Asia, and Oceania</td>
</tr>
<tr>
<td>Region</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NorthAmerica</td>
<td>Includes Canada, United States, Mexico, Central America, and Caribbean Islands</td>
</tr>
<tr>
<td>NorthAmerica_WOCentral</td>
<td>Includes Canada, United States, and Mexico</td>
</tr>
<tr>
<td>SouthAmerica</td>
<td>South America</td>
</tr>
<tr>
<td>CentralAmerica</td>
<td>Central America</td>
</tr>
<tr>
<td>CentralAmerica_withCaribbean</td>
<td>Central America, with Caribbean Islands</td>
</tr>
<tr>
<td>Europe</td>
<td>Europe, with 46 countries</td>
</tr>
<tr>
<td>EuropeWithCountries</td>
<td>Europe, with 49 countries (adds England, Northern Ireland, and Wales)</td>
</tr>
<tr>
<td>EuropeRegion</td>
<td>Europe, with five major regions: Western, Central, Eastern, Northern, Southern</td>
</tr>
<tr>
<td>EastEuropeanRegion</td>
<td>Includes Bulgaria, Romania, Moldova, Ukraine, Belarus, and Russia</td>
</tr>
<tr>
<td>CentralEuropeanRegion</td>
<td>Includes Switzerland, Germany, Liechtenstein, Slovenia, Austria, Czech Republic, Hungary, Slovakia, and Poland</td>
</tr>
<tr>
<td>WestEuropeanRegion</td>
<td>Includes France, Monaco, Luxembourg, Belgium, and the Netherlands</td>
</tr>
<tr>
<td>NorthEuropeanRegion</td>
<td>Includes Iceland, Ireland, UK, Denmark, Norway, Sweden</td>
</tr>
<tr>
<td>SouthEuropeanRegion</td>
<td>Includes Portugal, Spain, Andorra, Italy, San Marino, Vatican City, Malta, Croatia, Bosnia, Montenegro, Albania, Greece, Macedonia, Serbia, Turkey, and Cyprus</td>
</tr>
<tr>
<td>Asia</td>
<td>Asia, including Iran, Pakistan, Afghanistan, Turkmenistan, Uzbekistan, Kyrgyzstan, and Tajikistan</td>
</tr>
<tr>
<td>Asia3</td>
<td>Asia, without the Middle-East countries in the previous map</td>
</tr>
<tr>
<td>Africa</td>
<td>Africa</td>
</tr>
<tr>
<td>MiddleEast</td>
<td>Middle East, including Saudi Arabia, Israel, Turkey, Syria, Iraq, Qatar, Oman, Yemen, Kuwait, Bahrain, Emirates, and countries listed as included in Asia map</td>
</tr>
<tr>
<td>Oceania</td>
<td>Australia, New Zealand, and nearby Pacific islands</td>
</tr>
<tr>
<td>USA</td>
<td>United States, including individual states</td>
</tr>
<tr>
<td>USARegion</td>
<td>United States, with five regions: Northwest, Southwest, Central, North-</td>
</tr>
<tr>
<td>Region</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USANorthWestRegion</td>
<td>Northwest states, including Alaska, Washington, Oregon, Idaho, Montana, and Wyoming</td>
</tr>
<tr>
<td>USASouthWestRegion</td>
<td>Southwest states, including Hawaii, California, Nevada, Utah, Arizona, Colorado, and New Mexico</td>
</tr>
<tr>
<td>USACentralRegion</td>
<td>Central states, including North and South Dakota, Nebraska, Kansas, Oklahoma, Texas, Minnesota, Iowa, Missouri, western Michigan, Wisconsin, and Illinois</td>
</tr>
<tr>
<td>USANorthEastRegion</td>
<td>Northeast states, including eastern Michigan, Indiana, Ohio, Kentucky, Virginia, West Virginia, Pennsylvania, Maryland, Delaware, New Jersey, New York, Connecticut, Rhode Island, Massachusetts, Vermont, New Hampshire, and Maine, and the District of Columbia</td>
</tr>
<tr>
<td>USASouthEastRegion</td>
<td>Southeast states, including Arkansas, Louisiana, Mississippi, Tennessee, Alabama, Georgia, Florida, South Carolina, and North Carolina</td>
</tr>
<tr>
<td>(US - Individual States)</td>
<td>Map Type value is the state name, without any spaces; maps show counties.</td>
</tr>
<tr>
<td>Canada</td>
<td>Canada, including thirteen provinces: Yukon Territory, British Columbia, Northwest Territories, Alberta, Saskatchewan, Nunavut, Manitoba, Ontario, Quebec, Newfoundland &amp; Labrador, New Brunswick, Nova Scotia, and Prince Edward Island</td>
</tr>
<tr>
<td>(Canadian Provinces)</td>
<td>Maps provide counties. Map Type value is Ontario or Quebec.</td>
</tr>
<tr>
<td>UK</td>
<td>United Kingdom, including England, Wales, Northern Ireland, and Scotland.</td>
</tr>
<tr>
<td>(UK Countries)</td>
<td>Map Type value is one of four country names listed above, maps show counties.</td>
</tr>
<tr>
<td>ScotlandRegion</td>
<td>Scotland, with ten regions: Dumfries &amp; Galloway, Borders, Strathclyde, Lothian, Central, Fife, Tayside, Grampian, Highland, and Western Isles</td>
</tr>
<tr>
<td>(Countries)</td>
<td>These maps provide counties or other regional divisions. The Map Type values are: Afghanistan, Albania, Andorra, Antigua, Argentina, Armenia, Asia, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Belarus, Belgium, Belize, Bolivia, Bosnia, Herzegovina, Brazil, British Colum-</td>
</tr>
<tr>
<td>(Special Maps)</td>
<td>These maps provide regions or provinces: Norway Region, Spain Provinces</td>
</tr>
</tbody>
</table>
Introducing Google Maps

The Logi Info Google Map element allows developers to include the geo-mapping features of the Google Maps web service in their reports and provides an opportunity to combine it with their own information to produce hybrid maps, sometimes called "map mashups". This section introduces Google Maps and the Google Map element.

A Google Map is an interactive geographic map that can be programmed to identify locations and have other features. The concept is simple: take two different data sources (your data and Google’s maps) and combine them into something that’s more useful than the sum of their parts.

The example above shows a Google Map of the area surrounding Washington, D.C. It's a typical map depicting roads, major land features, and significant points-of-interest. The example also identifies the available optional controls, the scale for indicating map distances, and the cursor shaped that appears when dragging the map within its viewing area.
The example above shows the same map with the **Terrain** map style selected.

And, shown above, is the same map with the **Satellite** map type selected.

In "Street View" mode, shown above, Google Maps provide a street-level photographic view of a map area.

Street View is engaged by dragging the "Peg Man" icon, as shown above, onto an enabled street (shown outlined in blue when the icon is moved). Once dropped, the view switches to street-level and controls are presented allowing you to navigate and turn within the "virtual world".
Now the basic map example includes a "mashup" of data. When identifying data is fed to the web service, the resulting output can pinpoint locations on the map. The example above shows how a geographic Map Marker is placed on the map to identify a specific location.

Map Markers can use the default icon (shown) or a custom image, gauge, or even a chart. Optionally, the map can be configured to display a Map Marker Info window, containing additional location-specific data, when the marker is clicked.

The example shown above includes data-driven, colored regions (in this case, representing postal codes), known as Google Map Polygons, overlaid on a map of Washington, D.C. Logi Info can work with GIS boundary data to produce polygon-shaped overlays on for states, counties, cities, school districts, and other areas. Like the map marker, polygons can be clicked to display a popup information panel with detail data. They’re discussed in detail in a later chapter.
Similarly, **Google Map Polylines** can be plotted from data to show a route, in this case from the Orsay Museum in Paris to the Louvre Museum. Polyline color, width, and transparency level are all configurable and can be set from data values. They’re discussed in detail in a later chapter.

### About Web Services

The formal definition of a web service is "a software system designed to support interoperable, machine-to-machine interaction over a network". In plain terms, developers can think of a web service as a standard programming "function" that happens to be hosted on an external machine that your program reaches over the Internet. You send it parameters and the web service sends you results, with the benefit that the whole process is language neutral.

![Diagram](image)

The diagram above illustrates how this fits into the Logi application architecture. Usually, the "parameters" that you send to the web service come from a database that your application queries. The results are used in the HTML pages that are output to a browser. Logi Studio elements make the process of connecting to, and communicating with, the web service very easy.
API Licensing and Use

Web Service brokers, such as Google, make their web services available to Internet users, often for a fee. This means consumers of the services are obliged to sign up and pay, usually on a transaction-by-transaction basis.

We're making the following license-related information and links available as a courtesy but, as Google's licensing plans may change without any notice to us, please understand that we don't guarantee that the information to be currently accurate. You, as the developer, are responsible for reviewing the current Google licensing requirements and for taking whatever actions are legally necessary with regard to licensing.

Our Google Map elements use the Google Maps JavaScript API v3.

As of January 6, 2016, Google offers two license plans: Google Maps APIs Standard Plan and Google Maps APIs Premium Plan.

A Google Maps API Key of some kind is required after June 22, 2016. Logi applications that used the API without a key prior to June 22nd were "grandfathered in" to allow them to continue to function during a transition period. However, the exception expired on October 12, 2016 and an API key is now required.

Here are a number of Google links that may help you sort out the licensing changes and requirements:

- Google Maps API License Pricing and Plans
- Google Maps APIs FAQ
- Google Maps API Get A Key

Generally, to get a key in order to do development work, you'll need to have a Standard Plan key, which requires a personal Google account and registration of your application.

System Requirements

Your Logi application needs to be able to access the Internet in order to interact with the Google web service, which means there may be firewall and security implications for you to consider.

Google Maps can be exported to PDF.

The Google Map Family of Elements

A number of Logi Studio elements make up the family used to create Google Maps. Each element, its function, and its special attributes are discussed below:
Connection.GoogleMaps
This special type of Connection element is required for communicating with the web service. Like all Connection elements, it resides in your _Settings definition.

See our online document Google Connections for detailed information about getting an API Key and configuring this element.

Google Map
The root element for implementing a Google Map in your report. Attributes include:

- **ID** - (Required) A unique ID for this element.
- **Height & Width** - (Required) The dimensions, in pixels, of the map.
- **Connection ID** - The ID of the Connection.GoogleMaps element you added to your _Settings definition. If blank, the first (top-most) element of this type in _Settings will be used by default.
- **Google Map Pan Control** - Specifies whether and which type of pan controls appears on the map. This control allows the user to "slide" the map left, right, up, or down within the viewing window. This is similar to clicking and dragging the map, however the sliding movement occurs in fixed increments and in straight lines. Default value: True
- **Google Map Street View** - Specifies whether or not the "Street View" control appears on the map. This control contains a "peg man" icon that can be dragged onto the map to enable Street View mode, which provides a street-level photographic view of the surroundings of the icon. Default value: True
- **Google Map Type Control** - Specifies whether or not the Map Type controls appear on the map. These controls allow the user to select the type of map to be shown ("Map", "Satellite", etc.). The available map types the user can select will vary depending on the map size and the value of the Google Map Types attribute. The default value is "Auto", which selects the best control types based on map size and other factors.
- **Google Map Types** - Specifies the type of map available for selection by the user when Google Map Type Control value is "True" or "Auto". One or more map types may be entered, separated by commas, making the corresponding map types available to the user. The first map type entered will be the default type shown when the map is first displayed. Available map type values include "Map", "Satellite", etc.
Data Analysis using Maps


- **Google Map Zoom Control** - Specifies whether a zoom control appears on the map and its size. The value "Large" displays it as a slider, while "Small" displays it as "+/-" buttons. A value of "False" hides it altogether. The default is "Auto", which selects the best control type based on map size and other factors.

- **Map Scale** - Specifies whether a scale legend appears on the map. Default value: *False*

- **Security Right ID** - Specifies which security rights can view the map. If blank, then access is unrestricted.

**Google Map Initial View**

Allows you to optionally set the initial Zoom Level and geographic location the map will open to when the page is first displayed. Attributes include:

- **Google Map Zoom Level** - Specifies the resolution of the initial map. Values can range from 0 (the lowest zoom level, in which the entire world appears on one map) to 21+ (down to individual buildings).

- **Latitude & Longitude** - Specifies the positioning values, in the data returned in the associated datalayer, for each marker. If left blank, the defaults are "@Data.Latitude~" and "@Data.Longitude~". Like all tokens, the column names are case-sensitive.

**Google Map Markers**

Map Markers appear on the map to pinpoint a location, based on latitude and longitude positioning values. The graphic image for the marker can consist of an image, chart, or gauge; a default image is provided. Attributes include:

- **ID** - (Required) A unique ID for this element.

- **Latitude & Longitude** - Specifies the positioning values, in the data returned in the associated datalayer, for each marker. If left blank, the defaults are "@Data.Latitude~" and "@Data.Longitude~". Like all tokens, the column names are case-sensitive.

- **Security Right ID** - Specifies which security rights can view the map. If blank, then access is unrestricted.

You can add **Action** elements below this element so that when a user clicks a marker, an information panel appears or another report is shown.
### Map Marker Image

This *optional* element is a container for one Image, Gauge, or Chart element that provides the image for the Map Marker element. If this element is not included in the definition, the default marker image is used. The default image is:

![Default marker image](image)

Data can be used here in interesting ways. For example, an image caption (image file name) and the image size can be data from the datalayer (their values can be @Data tokens), so locations can be differentiated visually based on their data.

The marker image can also be an actual gauge or chart which would also immediately differentiate locations based on their data.

### Map Marker Label

This *optional* element allows you to define a label that will appear under the marker. The label can be styled most easily by defining a CSS class. Its attributes are Caption and Class.

### Map Marker Clustering

When present, this *optional* element will group markers into clusters according to their distance from a cluster's center.

When a marker is added, the marker cluster will find a position in all the clusters or, if it fails to find one, will create a new cluster with the marker.

The number of markers in a cluster will be displayed on the cluster marker. Clusters will break apart into individual markers when the mapped is zoomed-in sufficiently. This element has no attributes.
Google Map Polygons

Google Map Polygons are regions plotted onto a map and are optional. They can be semi-transparent and have colors that are based on data values. Polygons are plotted from sets of latitude and longitude points. These typically come from a DataLayer.Gpx File or DataLayer.Kml File element.

- **ID** - (Required) A unique ID for this element.
- **Border Color and Thickness** - Specifies the polygon border color and thickness in pixels. Default color: Red
- **Border Transparency** - Specifies a level of transparency for the border, where 0 = opaque and 15 = completely transparent. Default: 4
- **Fill Color** - Specifies the color that fills the polygon interior. Tokens can be used here to set the color based on data. Default: Red
- **Fill Transparency** - Specifies a level of transparency for the interior, where 0 = opaque and 15 = completely transparent. Default: 4

You can add Action elements below this element so that when a user clicks a polygon, an information panel appears or another report is shown.

Google Map Polylines

Google Map Polylines are lines plotted onto a map and are optional. They can be semi-transparent and have colors that are based on data values. Polylines are plotted from sets of latitude and longitude points. These typically come from a DataLayer.Gpx File or DataLayer.Kml File element.

- **ID** - (Required) A unique ID for this element.
- **Border Color and Thickness** - Specifies the polygon border color and thickness in pixels. Default color: Red
- **Border Transparency** - Specifies a level of transparency for the border, where 0 = opaque and 15 = completely transparent. Default: 4

You can add Action elements below this element so that when a user clicks a polygon, an information panel appears or another report is shown.
KML Overlay
The optional KML Overlay element allows display of one or more KML files as overlays on the Google Map. When an overlay is used, the boundary viewport (location and zoom) are set according to the last KML file in the list.

- **ID** - (Required) A unique ID for this element.
- **KML URL** - The URL of a KML or KMZ file. The URL must be publicly-accessible; local intranet URLs will not work.

DataLayer Element
Datalayer elements are used as usual for any data retrieval operation and can have Group Filters, etc. as child elements to shape the data. Any type of datalayer element can be used.

The data for this purpose should include at least some combination of the following (see section About Mapping Data, below):

- Street address
- City
- State
- Postal Code

Geocode Columns
This optional element is used if your database does not already include positioning data. It accepts address data and, via the Google web service, attempts to retrieve latitude and longitude values ("geocoding") for each record in the datalayer. The values are placed into two columns, named "Longitude" and "Latitude", that are added to the DataLayer.

Without a specific license, this element will work for a time- and rate-restricted volume of requests. Customers can secure a "Google Maps API for Business" license to alter or remove these restrictions (see the API Licensing section of this document) when using the Connection.Google Maps element's Google Maps Client ID and Private Key attributes.

Follow this link for a list of the countries for which Google currently provides geocoding. Other web service brokers also provide geocoding in other countries. Element Attributes include:

- **City Data Column** - Name of a datalayer column that has the city name.
- **Connection ID** - The ID of the Connection. Google Maps element you added to your _Settings definition. If blank, the first (top-most) element of this type in _Settings will be used by default.

- **Country Data Column** - Name of a datalayer column that has the country name.

- **ID** - A unique ID for this element.

- **Include Condition** - (v10.0.319) An expression that evaluates to a value of True or False. If the attribute is blank or evaluates to True, geocoding will occur; if the value evaluates to False, the element is skipped.

- **House Number Data Column** - The name of a datalayer column that contains the street number.

- **Latitude Column ID & Longitude Column ID** - The names of the columns that the web service will add to the datalayer and fill-in with the Latitude and Longitude values. Default column names are "Latitude" and "Longitude".

- **Place Data Column** - The name of a datalayer column that has the entire address information or a place name in a single string. Use this attribute when the address data is not broken up into separate columns or when naming a point-of-interest, such as "Washington Monument" or "Grand Canyon". Use of this attribute disables the other location data columns.

- **Postal Code Data Column** - The name of a datalayer column that has the postal code data (the "Zip Code" in the U.S.)

- **State Province Data Column** - The name of a datalayer column that has the state or province name.

- **Street Data Column** - Name of a datalayer column that contains the street name.

---

**Reverse Geocode Columns**

This *optional* element is used to produce address data from geographic coordinates. Values returned are put into columns that are added to the datalayer, using these column names:

- "StreetNumber",
- "StreetName",
- "Locality",
- "Sublocality",
- "AreaLevel1",
- "AreaLevel2",
- "AreaLevel3",
- "Country",
- "Latitude",
- "Longitude",
- "PostalCode".

Attributes include:

- **ID** - (Required) A unique ID for this element.

- **Latitude Data Column** - (Required) Name of the existing datalayer column that contains the latitude data.

- **Longitude Data Column** - (Required) Name of the existing datalayer column that contains the longi-
Data Analysis using Maps

**Action.Map Marker Info**
This *optional* element adds click event processing to the Google Map Markers element. When a map marker is clicked, processing flow continues with this element's child elements. Attributes are:

- **ID** - (Required) A unique ID for this element.
- **Security Right ID** - Specifies which security rights can click the map and invoke an action. If blank, then access is unrestricted.

**Map Marker Info**
This *optional* element represents the pop-up "balloon" that appears over the map when the parent Map Marker is clicked. A wide variety of elements, forming a virtual sub-report, can be placed as children below this element. These can include images, charts, links, data, and more. The data from the row associated with the clicked marker is available to the children of this element. (see the example image below).

---

**Refreshing Google Maps**

Developers may want to refresh their Google Maps with new data, either automatically or in response to user input. To do this, you can, of course, use an **Action.Report** element to refresh the entire report page.

Or if you want to refresh just the map, not the entire page, you can place your Google Map in a separate report definition and include it in your original report as a subreport using the **SubReport** element. Make the SubReport element the target of an **Action.Refresh Element** or **Refresh Element Timer** element and don't forget to use Link Parameters under the SubReport to pass any necessary values to the Google Map.

A Google Map element is usable as a direct target of an Action.Refresh Element or Refresh Element Timer element, however, this refreshes only the Google Map Markers and not the entire map.

---

**About Mapping Data**

Data used for Google Maps typically contains, at the least, address information. When this information is sent to the web service, the map generated will be scaled to fit the data into the dimensions you set for the map (in the Google Map element).
For example, if your data only includes state/province information for a single state, the map generated will be scaled to show as much of the state as necessary to include all of the map markers (above, left). This could be the result, for example, of a SQL query statement that includes a `WHERE State = 'MT'` clause.

However, if your data includes street address, city, and state/province information, the map generated will be scaled as a street-level map (above, right). This could be the result, for example, of a SQL query statement that includes a `WHERE City = 'New York'` clause.

As shown above, one of the benefits of the interactive nature of these maps is that you can display additional information when the map marker is clicked. So, your data might include business intelligence data in addition to the address information.

For example, revenue figures or employee counts. This data can be displayed, as in the example, in the Info Window that pops up when a marker is clicked. The Info Window can also contain images, charts, gauges, sub-reports, and links, all driven by data.
If your data already includes latitude and longitude positioning data, so much the better. The use of Geocode Column elements to provide that data involves additional "trips" to the web service and, depending on the service broker's agreement, that could mean additional transaction fees.

In addition, at the time of this writing, Google imposes a geocoding limit of 2,500/day (using free service) or 100,000 day limit (Premiere API license) or enterprise users may have other contractual limits. See the *Google Maps Geocoding Usage Limits* on their website for more information.

Support for data that includes "MultiGeometry" tags is included.
Google Map Tutorial

This section walks developers through building a sample application using the Logi Info Google Map element. Before starting this tutorial you should be sure that:

- You’re using Logi Info v12 or later.
- You’ve reviewed the Google Maps Terms of Use and have a Google Maps API Premiere license if needed.
- You’ve downloaded the sample data files and have unzipped them to a temporary location. The download URL is: http://devnet.logianalytics.com/downloads/googlemapdata.zip and the files are:
  
  HD_Dealers.xml
  HD_BarShield.gif
  BikeShops.xml
  BikeShops.gif

- You can connect to the Internet (outbound and inbound) from your development web server.
- You’re familiar with the general steps for creating a new Logi application and for accessing databases to retrieve data.

If any of the above items are incomplete or missing, please take care of them before proceeding.

Create the Basic Application

Get started by building your Google Map application and placing the sample data appropriately:

1. In Logi Studio, create a new application and register it with IIS. Test it to ensure that it works.
2. Extract all four files from the sample data .zip file and save them in the _SupportFiles folder under your Logi application project folder.

Configure the Web Service Connection

Assuming that have a Google Maps API Key, as described in the About Web Services chapter, you can configure a connection.
1. In Logi Studio, go to your _Settings_ definition and select the _Connections_ element.
2. Beneath it add a _Connection.GoogleMaps_ element and configure its attributes, as shown above. Other configurations are possible depending on the nature of your API Key.

We don't need any other connections for this tutorial: you're going to create sample Google Maps using data from two different XML data files, which do not require them.

**Create the Basic Google Map**

Now, with just a few quick steps, you'll produce a functional Google Map. The map will show you the locations of _Harley-Davidson Motorcycle Dealerships_ in the state of Illinois:

1. In your report definition, add a _Google Map_ element and configure its attributes, as shown above. If you're using a _Connection.Google Maps_ element, enter its ID in the _Connection ID_ attribute.

2. Add a _Google Map Markers_ element below the Google Map element and configure its attributes as shown above. The sample database you're working with includes columns named "Latitude" and "Longitude", which contain positioning data. Later on, you'll discover how to get that data from Google if you don't have it.
3. Add a **DataLayer.XML File** element below the Google Map Marker and and set its attributes as shown above, to filter out all records except those where the State column equals "IL", the postal abbreviation for Illinois.

4. Finally, add a **Condition Filter** element beneath the datalayer, as shown above, and set its **Condition** attribute as shown, to filter out all records except those where the State column equals the postal abbreviation for Virginia.

Now you're ready to create your first Google Map. **Save** and **browse** your application. The results should look something like this:
At this point, take some time to play with this map:

- Change the Compare Filter element’s attributes to get data for a different state or a city, Data Column = City, Compare Value = Chicago, for example, and notice how the map scales to the data.
- Note that your cursor is a "hand" when it’s over the map, meaning that you can pan the map by dragging the cursor. Give it a try!
- Change your Google Map element's Height and Width attributes and see what happens to the map scale.

### Adding Interactive Map Features

There are several ways to customize your map to make it more interactive. First, ensure that you change the Condition Filter element’s Condition attribute to get data for Chicago only (Data Column = City, Compare Value = Chicago).

#### Put Controls on the Map

A Google Map can be configured to include a number of useful controls:

Revisit your Google Map element and set its Google Map Pan Control, Google Map Zoom Control, and Map Scale attributes as shown above.
Save and browse your application. The map should now include **Pan** and **Zoom** controls and a **Scale legend**, as shown above. Try out the Zoom and Pan controls - they're actually a lot of fun to use!

**Switch Map Types**

The Google Map has other map types available: a **terrain** map, a **satellite** image, and a **hybrid** combination of the satellite image overlaid with the street map. These can be selected at runtime by the user if you add the proper controls:
Go back to your Google Map element again and set its attribute as shown above.

![Map and Satellite buttons](image)

Save and browse your application. The map should now include the **buttons** shown above, which allow the user to switch between map types. Click the buttons and see what happens. You’ll see that some of the map types are selected using checkboxes that appear when you click a button.

Note that the **order** of the buttons on the map matches the order of the values in the Google Map Types attribute.

**Display a Custom Map Marker**

The map of Harley-Davidson dealerships we created earlier can be enhanced by **changing the graphic** used to indicate locations on the map.

1. In your definition, beneath the Google Map Markers element ("mmDealerships"), add a **Map Marker Image** element. This element has no attributes.
2. Beneath it, add an **Image** element and set its attributes as shown. The image in from the data .zip file you downloaded earlier. A **Height** and **Width** value is required when using an Image element in a Google Map.
Save and browse your application, and see that the stock map marker has been replaced by the custom image, as shown above. Naturally, you have to exercise some caution when designing the size of alternate marker images so that you don't completely hide the map behind them.

Speaking of size, you can cause the marker image itself to convey relative data by setting its Height and Width attributes to some value from your database. In the example above, the token for a sales revenue data column (just an example here, it's not in the sample data) is used in the attribute values. The images on your map will then appear larger or smaller based on the data, allowing easy visual comparison.

**Geocoding Data**

So far, you've been working with sample data that includes address information and positioning data (latitude and longitude).

But what if you're working with a database that doesn't have that positioning data? Most location-oriented databases only have address information such as street address, city, state, and zip code. Well, if you have ad-
dress information or place names, you can use a Google Map "helper" web service to get the positioning data. Even partial address data, such as zip codes alone, can result in useful positioning data.

The way this works is: you send your address information to the helper service and it will return it with the positioning data (if it can find it) appended. Practically, what happens is that your datalayer gets two columns added to it, with default names "Longitude" and "Latitude". This process is called "geocoding". The following exercise will show you how it's done.

1. In Studio, first go to your _Settings definition and select the General element. Set the value for the Debugger Style attribute to DebuggerLinks.

2. In your main report definition, collapse and remark your “gmHarleyDealers” Google Map element and add a new Google Map element to your definition, as shown above. Set its attributes to match those of the remarked Google Map element.

3. Beneath the new Google Map element, add a Google Map Markers element, as shown above. Set its ID attribute to mmBikeShops and set its other attributes to match those of the remarked Google Map Markers element.

4. Beneath that element, add a Datalayer.XML File element and set its attributes as shown above. It will use one of the .xml from the data .zip file you downloaded earlier.

5. Add a Geocode Columns element beneath the datalayer and set its attributes as shown above. If you're using a Connection.Google Maps element, enter its ID in the Connection ID attribute.

Also, notice that there are attributes for the Latitude and Longitude Column IDs; these default to "Lati-
tude" and "Longitude", which also happen to be the names of the columns that the Google service appends to your data. Therefore, you can leave them blank.

There's also a **Place Data Column** attribute. This is used if you don't have address data but instead have the names of points-of-interest, such as "Washington Monument" or "Brooklyn Bridge".

Save your changes and browse the application. Your map of bicycle shops in Washington State should look like the image shown above.

**Examine the Geocoded Data**

Let's look at what the geocoding process actually did: click the Debug icon at the bottom-right corner of your map page. Your map page will be replaced with the **Debugger Trace** page.

Look down the page and, in the Event column, find the **Get DataLayer.XML File** entry, as shown above. A little further down, in the right-hand column, look for and click the **View File Stream Data** link, as shown above (the number of bytes may vary).
You should see the data in a table that was retrieved into your datalayer from the data file. Notice that two new columns, *Latitude* and *Longitude*, with positioning data have been added to each record by the Google Map service.

Return to the Debugger Trace page again and look down the Event column for **WARNING** messages. You can see here what happens in the trace when an address *can’t be geocoded*. This maybe because of something extraneous in the data or you may have reached your allowed Google Map query limit.

Geocoding is a slick way of getting positioning data based on your address information! However, keep the following in mind:

- *Each data record* submitted for geocoding is considered a "transaction" and counts against your daily allotment as described in the terms of your license with Google.

- Geocoding takes *time*; the more records to be processed, the longer a delay the user faces so, for example, geocoding the London telephone book in real time is not recommended.

- *Errors* in your address data not only take geocoding time but the failed attempts still count as transactions, so it’s in your interest to have your address data be as *clean* and *accurate* as possible before geocoding.

- The Google service also "throttles" geocoding requests to keep itself from being overwhelmed and generally returns an Error 620 if too many requests are received too quickly. The threshold is described in the terms of your license with Google.
The “Include Condition” Attribute

The Geocode Columns element has an **Include Condition** attribute. If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is utilized with the map. If the value evaluates to *False*, the element is ignored. This feature allows developers to dynamically switch between different sets of geocoding data.

Reverse Geocoding

Suppose you have the opposite location data situation from the one discussed above: you have the positioning data but not the address information. A typical use-case would be a mobile device app that “knows” its own GPS coordinates and needs to display the address.

The **Reverse Geocode Columns** element allows you to feed positioning data to the Google web service and retrieve address information. The web service will attempt to append and return data for the following columns into the datalayer: StreetNumber, StreetName, Locality, Sublocality, AreaLevel1, AreaLevel2, AreaLevel3, Country, Latitude, Longitude, and PostalCode.

Adding an Info Panel

You’re now ready to add even more interactivity to your Google Map. The **Info Panel** allows you to provide detailed information when users click map markers. **Disclaimer: While the stores used in this database are real, the data about them is fictional.**
1. In your definition, add an **Action. Map Marker Info** element beneath your Google Map Marker element ("mmBikeShops"). It has no attributes that need to be set.

2. Beneath it, add a **Map Marker Info** element. It, too, has no attributes that need to be set.

3. Beneath the MapMarkerInfo element, as shown above, first add a **Division** element and, assuming you have a Theme in use in your definition, set its **Class** attribute to **ThemeAlignCenter** and its **Output HTML Div Tag** attribute to **True**.

   Then, beneath it, add a **Label** element and a **New Line** element, as shown above. Set the Label's attributes as shown above; no attributes need to be set for the New Line element.

<table>
<thead>
<tr>
<th>ID</th>
<th>Caption Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lblAddress</td>
<td>@Data.Address~</td>
</tr>
<tr>
<td>lblCityStateZip</td>
<td>=&quot;@Data.City~&quot; +&quot;, &quot; + &quot;@Data.State~&quot; + &quot; + &quot;@Data.Zip~&quot;</td>
</tr>
<tr>
<td>lblPhone</td>
<td>@Data.Phone~</td>
</tr>
<tr>
<td>lblEmployees</td>
<td># of Employees: @Data.Employees~</td>
</tr>
<tr>
<td>lblURL</td>
<td>@Data.URL~</td>
</tr>
</tbody>
</table>

4. Add five more **Label** elements and four **New Line** elements to separate them. Use the information shown above to set the Label elements' attributes (but leave their Format attributes blank this time).
5. Finally, beneath the last Label element (lblURL) add an <Action.Link> and <Target.Link> element as shown above. Set the Target.Link element's attributes as shown above.

Save your application and browse or preview it. Now, when your map appears, click one of the map markers. You should see something like this:

Click on different map markers. As you can see, the Info Window is displayed with data from the database for each store when you click on its marker. You can also click on the store URL in the Info Window to go to their web site in another browser window.

Other Action-type elements could be used in your definition to redirect users to Logi application detail reports, processes, etc.
Adding Finishing Touches

To finish your map of bicycle shops, let's add two final flourishes. First, add a custom map marker image:

Beneath the Google Map Markers element ("mmBikeShops"), add a **Map Marker Image** element and, beneath it, an **Image** element. The Map Marker Image element has no attributes; set the Image attributes as shown above. Note that you're setting the **size** of the custom image to the employee **count** data from the database, so shops with more employees will have a larger image.

Let's test that concept. Save your application and browse or preview it. You should see something like this:

Click one of the map marker images and compare the number of employees reported in the Info Panel. Do the marker image sizes reflect the relative employee counts?

Finally, let's add a **Pie Chart** to the Info Panel. **Note that this operation requires use of a separate database and cannot be achieved using the sample XML data used in this tutorial.**
Beneath the MapMarkerInfo element, add a **New Line** element, a **Label** element, and a **New Line** element as shown above. Set the New Line elements' **Line Count** attribute to 2. Set the attributes for the Label element as shown above.

Add a **Chart Canvas** element and a **Series.Pie** element and set their attributes appropriately. Add a datalayer element beneath either of them retrieve data, and use an @Data token, representing a value from the original map datalayer ("dlBikeShopsXML"), to filter your query. An example SQL query is:

```
SELECT Category, PercentOfSales FROM SalesData Where ID = @Data.ID~
```

Save your application and browse or preview it. Click on a map marker and you should see something like this:
Congratulations, you've successfully completed this Google Map tutorial. You've seen how to connect to the Google Maps web service, how to plot data on the map, how to add interactive map controls, how to geocode address data, and how to use the Info Window.
Using Google Map Polygons

Google Map Polygons, plotted onto the map, are overlays or "regions" that can be color-coded based on data values. Here's an example:

The map above shows the northern half of Washington, D.C. with polygons representing different postal codes overlaid on it. The color of the polygon denotes the number of banks located within the region. The polygons have been configured to be semi-transparent, so that geographic features, such as streets, can be seen through them.
Polygons function like traditional Google Map Markers and can be related to other data. In the example above, they’ve been configured so that when they’re clicked, a pop-up information panel is displayed containing more detailed information. Polygons and Map Markers can be used in the same map.

Other examples of geographic polygons include political boundaries (such as states, counties, and cities), school districts, voting districts, and water management districts, to name but a few.

**Geographic Area Boundary Data**

Polygons are plotted using sets of *latitude* and *longitude points* that describe the boundaries of geographic areas. For use with Logi elements, these typically come from GPX or KML files, which are GIS industry standard XML data files. Special datalayers are provided in Logi Info to read these files. Because of the public nature of many of these geographic areas, their boundary data is often freely available on the Internet.

Area boundaries are also widely available in *shapefiles*, a popular geospatial vector format for GIS data. Free or inexpensive software tools, discussed in a later section, are available for converting shapefiles into GPX or KML data files.

Geographic data can also be retrieved from SQL database tables using DataLayer.SQL - see a later chapter for more information about that.

**General Features**

The following describe additional features of Google Map polygons:

- Borders, fill colors, and transparency level are all configurable and may be set from data values.
- Colors may be set in steps or as smooth color spectrums based on minimum and maximum data ranges.
- Polygons may be *included* or *excluded* based on data values.
- Polygon resolution is adjustable, and is dynamically sharpened as the user zooms the map in or out.
- Maps are initially displayed at a location and zoom level that shows all polygons.
• A special color spectrum legend element can be displayed below or alongside the map.

The examples discussed in this document are available on DevNet for download as the Google Map Regions Sample Application.

Understanding the Data

The following example demonstrates how to implement Google Map Polygons. The purpose of the example is to create a map of the state of Florida, with an overlay of all its public school districts, in order to easily compare "dropout rates" - the percentage of students who have dropped out of the school system - in the 2007-2008 school year. As is often the case, the first step is to have a look at the data.

For this example, relevant annual dropout rate data from the State of Florida's Department of Education has been downloaded and stored as an XML data file.

```
<?xml version="1.0" encoding="utf-8"?>
<dropOuts District="ALACHUA" Y1999="5.7" Y2000="6.3" Y2001="6.1" Y2002="5.2"
Y2003="5.1" Y2004="5.1" Y2005="5" Y2006="6.1" Y2007="6.6" Y2008="3.6" />
<dropOuts District="BAKER" Y1999="9.7" Y2000="3" Y2001="4.2" Y2002="3.5" Y2003="3.7"
Y2004="4" Y2005="4.3" Y2006="3.7" Y2007="2.8" Y2008="1.8" />
<dropOuts District="BAY" Y1999="2.5" Y2000="3.5" Y2001="1.6" Y2002="1.5" Y2003="1.1"
Y2004="1.8" Y2005="1.2" Y2006="2" Y2007="2.5" Y2008="1.7" />
```

A sample of the XML dropout data is shown above; note that the field District contains the school district name values.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" version="1.1" creator="ExpertGPS 3.03"
xschema="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd"
http://www.topografix.com/GPX/1/1/gpx_modified/0/1 http://www.topografix.com/GPX/gpx_modified/0/1/gpx_modified.xsd">
<metadata>
<bounds minlat="24.54470100" minlon="-87.63493800" maxlat="31.00088800" maxlon="-80.03136200"/>
<extensions>
<time xmlns="http://www.topografix.com/GPX/gpx_modified/0/1">2009-04-20T19:40:01.535Z</time>
</extensions>
</metadata>
<extensions>
<polyline xmlns="http://www.topografix.com/GPX/gpx_overlay/0/3">
<desc>ALACHUA COUNTY SCHOOL DISTRICT</desc>
<label>ALACHUA COUNTY SCHOOL DISTRICT</label>
</polyline>
<points>
<pt lat="30.99692800" lon="-85.49827200"/>
<pt lat="31.00088400" lon="-85.24363200"/>
<pt lat="31.00083500" lon="-85.15445200"/>
<pt lat="31.00083400" lon="-85.15221800"/>
```
Similarly, GIS data defining the boundaries of the state's school districts has been downloaded and stored as a GPX file and a sample of it is shown above. Note that the school district name is part of the data in the `desc` field. The beginning of the latitude and longitude data for the district boundaries is at the bottom of the sample.

In our Logi application, we'll relate the *school district names* in these two files with each other in order to make our map overlay functional.

The format of a KML file is somewhat different but it's still XML data and you can look at it easily enough to determine relevant field names.

**Creating a Google Map with Polygons**

Before starting to create an application that uses Google Maps, you must get a Google Maps API Key, in order to access the Google web service. See the section *About Web Services* in this document for information about getting an API Key. The example begins below with the assumptions that an API Key has been secured and that a `Connection.GoogleMaps` element has been configured with it in the `_Settings` definition.

1. In your report definition, add and configure a *Google Map* element to use your Google Map connection.
2. Beneath it, add a *Google Map Polygons* element, as shown above. This configures the regions that will overlay the map and its *Border* and *Fill* attributes affect the appearance of the regions. We'll come back later to set the *Fill Color* attribute.

3. Beneath the polygon element, add a *DataLayer.GpxFile* element, as shown above. It directly reads the GPX file we saw earlier to get the GIS data defining the boundaries of the state's school districts. If you're working with KML files instead, there's also a *DataLayer.Kml File* element for reading them. Tokens like
@Function.AppPhysicalPath~ can make it easier to identify a file stored within your application folder.

4. Beneath the datalayer, add a **Calculated Column**, as shown above. This column is used to compensate for the fact that the school district names are formatted differently in the two data files examined earlier. The complete formula attribute value is:

   \[
   \text{Replace("@Data.desc~", " COUNTY SCHOOL DISTRICT", "")}
   \]

   This truncates the data, creating an additional datalayer column that contains only the district name.

5. Next, add a **Join** element, a **DataLayer.XML File** element and a **Match Condition** element, as shown above. The Join element is configured as an **Inner Join** and the datalayer retrieves data from the XML file with the dropout data we examined earlier.

   The Match Condition element is configured as shown above, relating the **District** column in the XML file with the calculated column **calcDesc** created in the previous step from the GPX file.
6. Select the DataLayer.GpxFile element ("dlFloridaGPX") and, beneath it, add a **Color Spectrum Column** element, as shown above. This column will contain the color value for each row in the datalayer, determining the color for each region on the map.

   The **Data Column** attribute value will be the name of the column with the data for the year we want to compare, 2008, and the color attributes are set to provide a color that ranges between **Red** and **Green** based on the dropout rate. Make a note of its **ID** attribute value, which will be used in the next step.

7. Reselect the Google Map Polygons element ("polyMap1"), as shown above, and set its **Fill Color** attribute value to the data token for the Color Spectrum Column added in the previous step.
8. Finally, beneath the Google Map Polygons element, add a **Polygon Color Spectrum Legend** element, as shown above. This will create a color gradient legend beneath the map.

The resulting map looks like the example shown above. The map behaves like any Google Map and can be zoomed and re-centered. This document continues on page two, with an example showing how to add drill-down capabilities to the regions.
Adding Drill-down Capabilities

One of the most attractive features of Google Maps is their *drill-down* capabilities: clicking a Map Marker in a map causes a popup info panel to open, which can contain detail information. The same is true for Google Map Polygons - they can also be clicked in order to drill into the data. Continuing with our previous example, here's how this capability is added:

1. In the report definition, start by selecting the Google Map Polygons element. Beneath it add an **Action.Map Marker Info** element, and then beneath it, a **Map Marker Info** element, as shown above. Neither needs any configuration beyond being given an ID.

2. Beneath the Map Marker Info element, add a **Division** element and beneath it add several **Label** and **New Line** elements, as shown above. The Division allows a uniform style to be applied to the Labels, and the Labels display detail data from the datalayer, which is still in scope. Note the use of an @Data token to display the school district name.
3. Next select the Map Marker Info element again and beneath it add `SubReport` and `Target.Report` elements, as shown above. This will allow you to add a chart or table, defined in a separate definition, to the popup info panel. Note that the `Frame Border` and `Scrolling` attributes are set to `False` so that the chart or table will embed smoothly into the popup panel.

Finally, a `Link Parameters` element can be added beneath the SubReport element, as shown above, to pass identifying information to the detail report.
The resulting pop-up information panel displayed when a region is clicked is shown above.

**Tools for GIS Data Conversion**

As mentioned at the beginning of this document, GIS data describing region boundaries is used in Logi Info as either a **GPX** or **KML** file. In addition, data is also widely available in shapefiles (SHP). Note that these files need to use the standard Mercator projection, based on the WGS 1984 Ellipsoid projection, that Google Maps requires; "conic" projections, for example, will not work.

A search of the Internet will yield many hits for software tools, free and paid, that can convert GIS data into these formats. Logi Analytics does not specifically endorse or recommend any particular tool but we have experimented with the following:

- **SHP2KML**
  The SHP2KML program (freeware) available at [http://www.zonums.com/shp2kml.html](http://www.zonums.com/shp2kml.html) will convert a SHP file into a Google KML file.

---

*The following resources are provided to you as a courtesy. They're external resources not associated with Logi Analytics and we cannot guarantee that the links provided will remain viable. Developers are encouraged to search the Internet for their own resources.*
GPSBabel
The GPSBabel program (licensed under GNU—but should be fine for just about every use except redistribution) is available at http://www.gpsbabel.org/ and will convert KML to GPX format.

GPS Visualizer
This is a free online conversion tool which is probably good for small files but not larger ones: http://www.gpsvisualizer.com/convert_input?convert_output=gpz

ExpertGPS
ExpertGPS makes it very easy to convert SHP files into GPX or KML formats: http://www.expertgps.com/

The program isn’t free, but it’s low cost ($74.95 at this writing) and is available in a 30-day trial version. The program makes it very easy to convert between SHP and other file formats.

When you download a .zip file containing SHP files, it will contain three files. You'll need to unzip all three files to a folder and then use the "Import" function of the ExpertGPS program to generate the GPX file (you'll only select the .shp file for import but the process reads from the other two files to get the appropriate labels and other metadata). To create the appropriate GPX file for Logi Info to work you need to make sure that you're populating the "Description" field with the boundary name from the SHP file (e.g. zip code, county, etc); it's suggested that you leave all the other fields empty to create the smallest possible GPX file.

Using Data from SQL Server

The Google Map Polygons element can also use DataLayer.SQL to retrieve data from a SQL database for use in plotting geographic areas. The required data format and techniques for loading and retrieving data for this purpose are discussed in this blog post, Logi Analytics Choropleth Maps using SHP Files and SQL Data Layer, by Logi Analytics customer Robert Horvick.

Note that this link is provided as a courtesy and questions regarding the techniques described in this post should be directed to its author, not to Logi Customer Support. Mr. Horvick is not a Logi employee.

GIS Data Resources

The following resources are provided to you as a courtesy. They're external resources not associated with Logi Analytics and we cannot guarantee that the links provided will remain viable. Developers are encouraged to search the Internet for their own resources.

These web sites provide free, public GIS boundary data in a variety of formats:

United States Census Bureau
These SHP files are perfect for creating the boundaries for any U.S. state or territory. http://www.census.gov/geo/maps-data/data/tiger-cart-boundary.html

Texas State Data Center
Most U.S. states will have information like this available: http://osd.texas.gov/Data/Decennial/2010/
UC Berkley
Geographic data for most countries worldwide:  [http://gif.berkeley.edu/resources/data_subject.html](http://gif.berkeley.edu/resources/data_subject.html)

University of North Carolina
Geographic data for most countries worldwide:

Shapefile Resources
These are pretty good SHP file resources for European and world data files:
[http://www.cdc.gov/epiinfo/pc/shapefiles.htm](http://www.cdc.gov/epiinfo/pc/shapefiles.htm)

...and there many other resources are available on the Internet.
Using Google Map Polylines

Google Map Polylines, plotted onto the map, are overlays that are drawn, and can be color-coded, based on data values. Here’s an example:

The map above uses polylines to show a route from the Orsay Museum in Paris to the Louvre Museum. Polyline color, width, and transparency level are all configurable and can be set from data values. The line above has been configured to be semi-transparent, so that geographic features can be seen through it.

Polylines can be configured with Google Map Marker Info pop-ups and can be related to other data. In the example above, they've been configured so that when they're clicked, an information panel is displayed containing more detailed information about the immediate location. Other use-case examples include subway maps, traffic density indicators, and pipeline route mapping, to name but a few.
Geographic Data

Polylines are plotted using sets of **latitude** and **longitude points** that describe line segments. Multiple segments are used to draw a continuous line. For use with Logi elements, these typically come from GPX or KML files, which are GIS industry-standard XML data files. Special datalayers are provided in Logi Info to read these files. Because of their public nature, this data is often freely available on the Internet.

Data is also widely available in *shapefiles*, a popular geospatial vector format for GIS data. Software tools, discussed in the previous chapter *Using Google Map Polygons* for converting shapefiles into GPX or KML data files. Geographic data can also be retrieved from SQL database tables using DataLayer.SQL; see the Polygons chapter for more information.

Understanding the Data

The following example will demonstrate how to implement Google Map Polylines. The purpose of the example is to create the example we've seen above: routes between the Louvre Museum in Paris and several other primary tourist attractions. As is often the case, the first step is to have a look at the data.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx version="1.0" xmlns="http://www.topografix.com/GPX/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/0 http://www.topografix.com/GPX/1/0/gpx.xsd">
  <trk>
    <name>LouvreToCafeMetropole</name>
    <trkseg>
      <trkpt lat="48.8610022" lon="2.3352277"></trkpt>
      <trkpt lat="48.8607410" lon="2.3350883"></trkpt>
      <trkpt lat="48.8605857" lon="2.3356140"></trkpt>
      <trkpt lat="48.8598023" lon="2.3351634"></trkpt>
      <trkpt lat="48.8594352" lon="2.3366225"></trkpt>
      <trkpt lat="48.8593152" lon="2.3379850"></trkpt>
      <trkpt lat="48.8591670" lon="2.3390150"></trkpt>
      <trkpt lat="48.8589482" lon="2.3402274"></trkpt>
      <trkpt lat="48.8590470" lon="2.3402917"></trkpt>
    </trkseg>
  </trk>
  <trk>
    <name>LouvreToOrsay</name>
    <trkseg>
      <trkpt lat="48.8611857" lon="2.3351204"></trkpt>
      <trkpt lat="48.8613198" lon="2.3345196"></trkpt>
      ...
  </trk>
</gpx>
```
The GIS data defining the line segments for each route has been downloaded and stored as a GPX file, and a small part of it is shown above. You can see that each route has a name and a set of latitude and longitude records, organized in a hierarchy.

The format of a KML file is somewhat different but it's still XML data and you can look at it easily enough to determine relevant field names.

**Creating a Google Map with Polylines**

Before starting to create an application that uses Google Maps, you must get a **Google Maps API Key**, in order to access the Google web service. The chapter *About Web Services* provides information about getting an API Key.

The example begins below with the assumption that you've gotten an API Key and that you've configured a **Connection.Google Maps** element in your _Settings definition with it.

1. In your report definition, add and configure a **Google Map** element to use your Google Map connection.
2. Beneath it, add a **Google Map Polylines** element, as shown above. This configures the lines that will overlay the map and its **Border** attributes affect their appearance.

3. Beneath the Polylines element, add a **DataLayer.Gpx File** element, as shown above. It directly reads the GPX file we saw earlier to get the GIS data defining the routes. If you're working with KML files instead, there's also a **DataLayer.Kml File** element for reading them. Tokens such as `@Function.AppPhysicalPath~` can make it easier to identify a file stored within your application folder. The full Filename attribute value above is: `@Function.AppPhysicalPath~\_SupportFiles\ParisTouristRoutes.gpx`
4. We want to dynamically select which individual route to display, so add a **Condition Filter** beneath the datalayer, as shown above. We'll use compare a data column value to a User Input selection we'll add later. The complete Condition attribute value is: 

```
"@Data.name~" = "@Request.inpRoute~"
```

5. Let's give the filter something to work with from the initial page load: a Default Request Parameter. Add the element, as shown above, and configure a request variable and value as shown. Remember that the spelling and case of the default parameter *must* match that of the Request token in the Condition Filter element's Condition attribute.

You should be able to test the application now and see one set of Polylines overlaid on the map.

6. Now we'll pick up the pace a bit. As you can see above, we've added **New Line** and **Label** elements to give the page a title.

And we've added an **Input Select List** element, which lets the user select which route he'd like to see.
drawn on the map. The datalayer reads an XML file (shown later) that contains correlates user-friendly route names to the names in the data.

Note that the element ID of the input element matches the spelling and name of the default request parameter we configured in the previous step.

Finally, in the example code, an Event Handler is added that refreshes the page when an input selection is made.

The resulting map looks like the example shown above. The map behaves like any Google Map and can be zoomed and re-centered. Select a different route in the Input Select List to re-draw the map and show the route.

Here's that XML data file used with the Input Select List:
<Paris>
    <Routes Name="The Orsay Museum" Route="LouvreToOrsay" />
    <Routes Name="Cafe Metropole" Route="LouvreToCafeMetropole" />
    <Routes Name="Tuileries Gardens" Route="LouvreToTuileries" />
    <Routes Name="Angelina Chocolates" Route="LouvreToAngelina" />
    <Routes Name="Hotel Palais Royal" Route="LouvreToPalaisRoyal" />
    <Routes Name="Cafe de l'Epoque" Route="LouvreToCafeDelEpoque" />
</Paris>
Using the Logi Google Map API

The Logi Google Map API is a JavaScript-based API that lets you handle Google Map-related events and objects. It depends on other JavaScript objects that are automatically created by the standard Logi Engine code. These objects are clearly identified in the example provided.


The Logi Google Map API is contained in \rdTemplate\rdGoogleMap\rdGmap.js, which is automatically included in your Logi application when you use the Google Map element. You do not need to include this script file independently.

You can include the JavaScript code you write that uses Logi Google Map API calls using the standard elements for that purpose, such as Include Script or Include Script File. If you use the latter, it must be a child of the Body or Report Footer elements, not the Report (root) or Report Header elements.

Events and Objects

The API provides the following events and objects:

Y

This object is an underlying Logi application script object, automatically created. It’s used in your code to identify the next two events.

rdCreate

This event fires before the Google Map object is created. It has three properties:

- **id** - Gets the ID of the Google Map object about to be created.
- **mapOptions** - Sets the Google Map's MapOptions object properties, ranging from background color to zoom control options. A complete list of these is available from Google on their specifications page.
- **container** - Gets the ID of the internal Division created as a container for the Google Map object.

The mapOptions property is the most useful as it allows you to configure the map in a variety of ways before it's created:
The example code shown above refers to a Google Map element with an element ID of "gmap". The code configures the map to disable the mouse scroll wheel (used to zoom in and out) when it's created.

**rdCreated**

This event fires after the Google Map object is created. It has four properties:

- **map** - Gets a reference to the newly-created Google Map object.
- **id** - Gets the ID of the newly-created Google Map object.
- **mapOptions** - Gets the Google Map's MapOptions object properties, ranging from background color to zoom control options. A complete list of these is available from Google on their specifications page.
- **container** - Gets the ID of the internal division created as a container for the Google Map object.

This event and its properties allow you to customize the map after it's been created:

```javascript
var mapDiv = Y.one('#gmap');
mapDiv.on('rdCreated', function(e) {
  var latLng = new google.maps.LatLng(38.924622, -77.216779);
  var marker = new google.maps.Marker({
    position: latLng,
    map: e.map
  });
});
```

The example code shown above refers to a Google Map element with an element ID of "gmap". It adds a Map Marker at a specific location (Logi World HQ) after the map is created, using the `LatLng` and `Marker` classes from the Google Map API.

```javascript
var latLng = new google.maps.LatLng(@Request.LatHQ~, @Request.LngHQ~);
```

Of course, the code can be made dynamic through the use of tokens, as shown above.

**rdGetMapObject()**

This Logi function returns a reference to the specified Google Map object and can be used to customize an existing Google Map based on events related to the Logi page, such as a user-initiated mouse click. Imagine a button or link on a page, with Action.JavaScript as its child with this code:
The example code shown above refers to a Google Map element with an element ID of "gmap". It adds a Map Marker at a random location, using the **LatLng** and **Marker** classes from the Google Map API, when the button or link is clicked.

### Example Definition

Here's a complete report definition that includes all of the previous code examples and some embellishments. [Download it](#) from DevNet and when it opens in your browser, save it, with an .lgx file extension, to the _Definitions_\_Reports folder in one of your Logi applications. The application version must be Logi Info v12.0.036 SP2 or later for it to work.

After you save the file, in Studio you may need to right-click the Reports folder in the Application panel and select **Refresh Application** in order to see the new definition.

```javascript
var myMap = rdGetGMapObject('gmap');
var randomNumber = Math.random();
var latLng = new google.maps.LatLng(38.924622 + randomNumber, -77.216779 - randomNumber);
var marker = new google.maps.Marker({
    position: latLng,
    map: myMap
});
```
Using the ESRI ArcGIS API

ESRI ArcGIS is a geographic information system for working with maps and geographic information, similar to Google Maps. When integrated into a Logi application, ArcGIS maps can driven by data from the wide range of datasources supported by Logi Info and contained within the rich Logi interactive framework.

Integration is achieved using ESRI's ArcGIS API for JavaScript and Logi script-related elements. Developers intent on doing this integration should be very experienced with JavaScript, JSON, and Logi Info development techniques and should review the ESRI API documentation before proceeding.

There are multiple techniques that can be used to include ArcGIS maps in a Logi application and several examples are shown below. The examples use Asynchronous Module Definition and Modern Dojo.

Licensing

The following information is provided as a courtesy by Logi Analytics and may change without notice. It's your responsibility as a developer to understand and comply with the ESRI ArcGIS licensing requirements.

Free use of the ArcGIS API is allowed for development and evaluations, for non-commercial external use, for educational purposes (teaching only), for NGOs and non-profit businesses, and when used in conjunction with other ArcGIS licenses and services.

Otherwise, commercial use on a publicly-accessible web site requires you to purchase an ArcGIS license.

For complete information about licensing, see the ArcGIS for Developers - Terms of Use page.

Requirements Check List

Before starting to integrate ArcGIS maps into a Logi application, be sure that:

- You're using Logi Info v11 or later
- You have a proper ArcGIS license, if necessary
- You can connect to the Internet from your development server
- You've examined ESRI's ArcGIS API for JavaScript documentation
- You're familiar with the general steps for creating a new Logi application and using JavaScript

If any of the above items are incomplete or missing, please take care of them before proceeding.
Adding a Simple ArcGIS Map

The following assumes an existing Logi application, with a report definition ready to have ArcGIS maps added to it.

We'll begin by adding two Style elements to your report definition, as shown above, and configuring them to access style sheets available from ArcGIS, using API v3.4. The style sheet URLs are:

http://serverapi.arcgisonline.com/jsapi/arcgis/3.4/js/esri/css/esri.css
http://serverapi.arcgisonline.com/jsapi/arcgis/3.4/js/dojo/dijit/themes/claro/claro.css

The second style sheet is for one of four Dojo Dijit themes available: claro, tundra, soria and nihilo. Feel free to try out different themes.

Next, we need to include the ArcGIS API library, using an Include Script File element, as shown above. The complete URL for the v3.4 API is:

http://serverapi.arcgisonline.com/jsapi/arcgis/3.4/

Note that this URL doesn't include a filename with a .js extension as you might expect.

Next, we add a Default Request Parameters element to provide some default values to control the initial map appearance and specify the icon image that will be used to pinpoint locations on the map. The names of the parameters are arbitrary and will be used later in our JavaScript code.
Now, we add a **Division** element, as shown above, which will be the ArcGIS map container. Its attributes are set to give it a unique ID, to set its Class to one from the Dojo themes, and to ensure that it's rendered as a DIV tag. You can control the size of the map by using CSS to set the height and width of this container element.

Next, we add an **Include Script** element with JavaScript code, using the API, to create and initialize the map. Here's that code:

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

var map;

function init() {
  map = new esri.Map("divMap",{
    basemap: "topo",
    center: [@Request.longitude~, @Request.latitude~], //longitude, latitude
    zoom: @Request.zoom~,
    infoWindow: new esri.dijit.Popup(null, dojo.create("div"))
  });
}
dojo.ready(init);
```

Note these items in the code:
The "divMap" identifier of our map container Division element is used in the Map constructor.

The API offers several basemap types, including: topo, streets, satellite, hybrid, national-geographic, and more.

@Request tokens, for the default request parameters, are used to center the map and set its initial zoom factor.

If we preview the definition, we'll see the map shown above. The zoom controls and ESRI logo can be controlled using different API options.

Now let's add some data-driven points to the map:

In this example, we'll access data in a .csv data file using Json Data and DataLayer.CSV elements, as shown above. The .csv file data looks like this:
The first row in the .csv file provides the data column names we'll use in our code. Notice that HTML is included right in the data, to provide links. We could, of course, get the data from any of the many other datasources supported by Logi Info.

In order to use the JSON data retrieved from the file, we need to add some new JavaScript code in our Include Script element, as follows:

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

var map;

function init() {
    map = new esri.Map("divMap", { 
        basemap: "topo",
        center: [@Request.longitude~, @Request.latitude~], //longitude, latitude
        zoom: @Request.zoom~,
        infoWindow: new esri.dijit.Popup(null, dojo.create("div"))
    });
    dojo.connect(map, "onLoad", addWells);
}

function addWells(data) {
    var symbol = new esri.symbol.PictureMarkerSymbol("@Request.iconFile~", 24, 24);
    var template = new esri.dijit.PopupTemplate({
        title: "@dataTitle~",
        description: "@dataDescription~"
    });
    for (var w in jsonVal) {
        var loc = new esri.geometry.Point(jsonVal[w].longitude, jsonVal[w].latitude);
        map.graphics.add(new esri.Graphic(loc, symbol, jsonVal[w], template));
    }
}

dojo.ready(init);
```
New code is shown above with in blue. Note these items in the code:

- The @Request token for the default request parameter for the map marker icon image is used.
- Data column names "dataTitle" and "dataDescription" are used to insert the data where needed for the pop-up that appears when a map marker is clicked.
- Latitude and longitude data for positioning the map markers is inserted using the array of values, json-Val, identified in the Json Data element.

Now when the map is displayed (and we zoom in a couple of clicks), the three map markers from our data file appear. If a marker is clicked, as shown above, a pop-up panel provides the data for that location. You may need to drag the map southward to see the complete pop-up panel. The pop-up panel can be styled and customized using the API.

**Adding a Polygon Overlay**

In this example, we'll draw a state map and add an overlay layer of polygons, representing the state's counties and reflecting their population size. This will be driven by U.S. Census data from an online source and we'll use the JavaScript from the ArcGIS Sample web site to render it, using API v3.9.
Once again, we'll begin by adding a Style element, as shown above, to a new definition. The style sheet for this example is: [http://js.arcgis.com/3.9/js/esri/css/esri.css](http://js.arcgis.com/3.9/js/esri/css/esri.css)

Next, we add the JavaScript code to create and initialize the map. The code has been taken from the example at [https://developers.arcgis.com/javascript/jssamples/renderer_class_breaks.html](https://developers.arcgis.com/javascript/jssamples/renderer_class_breaks.html) and saved into a file, ArcGISExample.js, in the application's _SupportFiles folder.

And, once again, we add a **Division** element, as shown above, which will be the ArcGIS map container. Its attributes are set to give it a unique ID and to ensure that it's rendered as a DIV tag. You can control the size of the map by using CSS to set the height and width of this container element.

Let's have a look at the JavaScript in the ArcGISExample.js file:

```javascript
var map;

require(["esri/map", "esri/layers/FeatureLayer", "esri/InfoTemplate", "esri/symbols/SimpleFillSymbol", "esri/renderers/ClassBreaksRenderer", "esri/Color", "dojo/dom-style", "dojo/domReady!"], function(
    Map, FeatureLayer, InfoTemplate, SimpleFillSymbol, ClassBreaksRenderer, Color, domStyle
) {

```


map = new Map("divMap", {
    basemap: "streets",
    center: [-98.215, 38.382],
    zoom: 7,
    slider: false
});
var symbol = new SimpleFillSymbol();
symbol.setColor(new Color([150, 150, 150, 0.5]));

// Add five breaks to the renderer
// If you have ESRI's ArcMap available, this can be a good way to determine break values
// You can also copy the RGB values from the color schemes ArcMap applies, or use colors
// from a site like www.colorbrewer.org
// alternatively, ArcGIS Server's generate renderer task could be used
var renderer = new ClassBreaksRenderer(symbol, "POP07_SQMI");
renderer.addBreak(0, 25, new SimpleFillSymbol().setColor(new Color([56, 168, 0, 0.5])));
renderer.addBreak(25, 75, new SimpleFillSymbol().setColor(new Color([139, 209, 0, 0.5])));
renderer.addBreak(75, 175, new SimpleFillSymbol().setColor(new Color([255, 255, 0, 0.5])));
renderer.addBreak(175, 400, new SimpleFillSymbol().setColor(new Color([255, 128, 0, 0.5])));
renderer.addBreak(400, Infinity, new SimpleFillSymbol().setColor(new Color([255, 0, 0, 0.5])));

var infoTemplate = new InfoTemplate("${NAME}", "${*}");
var featureLayer = new FeatureLayer("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Demographics/ESRI_Census_USA/MapServer/3", {
    mode: FeatureLayer.MODE_SNAPSHOT,
    outFields: ["*"],
    infoTemplate: infoTemplate
});

featureLayer.setDefinitionExpression("STATE_NAME = 'Kansas'");
featureLayer.setRenderer(renderer);
map.addLayer(featureLayer);
});

Note these items in the code:

- The "divMap" identifier of our map container Division element is used in the Map constructor.
- In this example, values for the map center point and zoom factor are hard-coded, but they could come from tokens, as in the earlier example.
• Renderer "breaks" relate population data to colors in the polygon overlay (the "FeatureLayer").
• Data for the overlay comes from an ArcGIS online sample source.

Now when the map is displayed the counties are shown in a polygon overlay and the polygons are colored based on their population density. If a polygon is clicked, as shown above, a pop-up panel provides the data for that county. You may need to drag the map southward to see the complete pop-up panel. The pop-up panel can be styled and customized using the API.

**Embedding Logi Visualizations**

In this final example, we'll draw a state map and add an overlay layer of polygons, representing the state's counties, as before. This time, however, when a county is clicked we'll display a pop-up panel that contains a Logi visualization (a bar chart) displaying the county's population size in two different years. This will be driven again by U.S. Census data from an online source and we'll use the JavaScript from the ArcGIS Sample web site to render it, using API v3.9.
Once again, we’ll begin by adding three **Style** elements, as shown above, to a new definition. The style sheets for this example are:


The third style sheet is for one of four Dojo Dijit themes available: *claro*, *tundra*, *soria* and *nihilo*. Feel free to try out different themes.

Next, we need to include the ArcGIS API library, using an **Include Script File** element, as shown above. The complete URL for the v3.9 API is: [http://serverapi.arcgisonline.com/hsapi/arcgis/3.9/](http://serverapi.arcgisonline.com/hsapi/arcgis/3.9/)

And, once again, we add a **Division** element, as shown above, which will be the ArcGIS map container. Its attributes are set to give it a unique ID and to ensure that it's rendered as a DIV tag. You can control the size of the map by using CSS to set the height and width of this container element.

Next, using an **Include Script** element, we add the JavaScript code to create and initialize the map. Our code looks like this:
var map;

require([
    "esri/map",
    "esri/InfoTemplate",
    "esri/layers/FeatureLayer",
    "esri/renderers/SimpleRenderer",
    "esri/symbols/SimpleFillSymbol",
    "esri/symbols/SimpleLineSymbol",
    "dojo/dom",
    "dojo/number",
    "dojo/on",
    "dojo/parser",
    "esri/Color",
    "dijit/layout/BorderContainer",
    "dijit/layout/ContentPane",
    "dojox/layout/ExpandoPane",
    "dojo/domReady!"
],

function (Map, InfoTemplate, FeatureLayer, SimpleRenderer, SimpleFillSymbol, SimpleLineSymbol, dom, number, on, parser, Color)
{

parser.parse();

map = new Map("divMap", {
    basemap: "streets",
    center: [-75.18, 42.34],
    zoom: 7
});

var infoTemplate = new InfoTemplate();
infoTemplate.setTitle("County Census Data");

var counties = new FeatureLayer("http://sampleserver1.arcgisonline.com/ArcGIS/rest//services/Demographics/ESRI_Census_USA/MapServer/3", {
    mode: FeatureLayer.MODE_SNAPSHOT,
    infoTemplate: infoTemplate,
```javascript
outFields: [  
    "NAME",  "STATE_NAME",  "POP2000",  "POP2007",  "POP00_SQMI",  
    "POP07_SQMI"
  ]
});

counties.setDefinitionExpression("STATE_NAME = 'New York'");

//apply a renderer
var symbol = new SimpleFillSymbol(SimpleFillSymbol.STYLE_SOLID,
  new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID,
    new Color([255, 255, 255, 0.35]), 1),
    new Color([109, 146, 155, 0.35]));
counties.setRenderer(new SimpleRenderer(symbol));

map.addLayer(counties);
});
```

Note these items in the code:

- The "divMap" identifier of our map container Division element is used in the Map constructor.
- In this example, values for the map center point and zoom factor are hard-coded, but they could come from tokens, as in the earlier example.
- An "InfoTemplate" is created for the pop-up panel that appears in response to clicks on the polygons. It includes the HTML to embed a different Logi report in an iFrame.
- The polygon overlay (the "FeatureLayer") reads data from an ArcGIS online sample source. Data retrieved is passed to the Logi report in the Info Template as a request variables in iFrame URL.
- A "SimpleFillSymbol" object is used to change a polygon's outline color when it's clicked.
If we preview the definition, we'll see the map shown above. The zoom controls and ESRI logo can be controlled using different API options.

Now let's add the separate Logi report definition that's going to appear in an iFrame in the pop-up panel.

Create a new definition, give it the name "CountyChart" (which matches the iFrame source URL in the JavaScript code above) and add some element for style, structure, and information, as shown above.

The Label element shown above uses one of the request variables passed in from the iFrame URL and it also has a Action.Link element (not shown) which uses the same request token to create a link to search for the county in Google.
And, finally, we add a set of elements for a Chart Canvas Chart, with a Bar series, as shown above. We have a fixed number (two) years of data, so we can use a DataLayer.Static element to receive the request variable data passed in the iFrame URL and place it in a datalayer to be used by the chart.

Now when the map is displayed the counties are shown in a polygon overlay, as before, and if a polygon is clicked, as shown above, a pop-up panel provides the data for that county, as a chart. The title is shown as a live link and all of the Chart Canvas Chart features, including hover-highlighting and Quicktips are present.
We've seen how easy it is to include ArcGIS maps into a Logi application, using static and online data. You can, of course, use any of the datasources Logi Info can access to produce JSON data to be consumed by a map.
Resources and Support

The following resources are available if you need additional information or support:

Corporate Headquarters

Phone: 1-888-564-4965
       (703) 752-9700
Fax: (703) 995-4811
Email: info@logianalytics.com
Address: 7900 Westpark Drive, Suite A200
         McLean, VA 22102
Web site: www.logianalytics.com
Developer Network: http://devnet.logianalytics.com

Sales Department

Email: Sales@logianalytics.com

Customer Service and Support

Email: CustomerService@logianalytics.com