Chapter No. 4
"Working with Controls"
In this package, you will find:

A Biography of the authors of the book
A preview chapter from the book, Chapter NO.4 "Working with Controls"
A synopsis of the book’s content
Information on where to buy this book

About the Authors

**Alper Dincer** is a civil engineer with an MS degree in Geographical Sciences. He has more than 10 years of experience in developing web and mobile GIS/LBS projects.

Since the beginning of his career, he was always passionate about maps and geo stuff. He started working as a research assistant at a university and then moved to a governmental agency to work with maps. He is also the co-founder of a tech company named Mekansal.

He also has some achievements relating to geo web projects. In 2009, he got the first place in the ESRI Developer Summit Mashup Challenge with his open source project ExtMap. ExtMap was based on the Google Maps JavaScript API v2. He is one of the Google Qualified Developers of the Google Maps JavaScript API program. In 2010, he also worked as a proctor in the same program as a volunteer.

As a developer and entrepreneur, he still likes coding with JavaScript, PHP, and Objective-C on different web and mobile projects. He completely agrees with Steve Jobs' quotes "Love what you do" and "Stay hungry, stay foolish". 

For More Information:

First, I would like to dedicate this book to my wife, Begum—the light of my life—for understanding my passion for coding and the digital world. I would like to thank God for having her in my life. Words cannot express my gratitude and love to her.

Second, I wish to acknowledge my loving family, who are always there for me.

Balkan Uraz is a city planner with an MS degree in Geographical Sciences. He has over 15 years of experience in the field of Geographic Information Systems (GIS).

Throughout his career, he has worked on several projects with one thing in common: GIS. In the early days of his career, he worked on projects involving municipal GIS and city information systems. He has also worked as a research assistant while he was conducting the tedious work on his thesis on routing systems.

He has worked on major LBS projects for mobile operators in Turkey that involve both software development and building the data inventory. He co-founded a tech company that specialized in navigation data collection and navigation products. He has also been a GIS consultant for major companies operating in the areas of field tracking and real estate.

In all his projects, he has worked around the same passion: building up the spatial infrastructure.

I would like to thank Esra for her love, support, and encouragement while writing this book. I would also like to thank my fellow colleagues for their enthusiasm and encouragement that lead to writing this book.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
Google Maps JavaScript API Cookbook

Currently, there are both open source and proprietary alternatives to the Google Maps JavaScript API, but what makes the API special for developers is that it is a complete solution with base maps, overlays, and technical capabilities.

The API has been especially exciting for developers because it is very easy to build up generic outcomes, and at the same time, it has its own tips and tricks and advanced functionalities within the same box. Therefore, you can swim afloat or dive deep when you are working with the API.

The Google Maps JavaScript API v3 enabled the quick and easy development of mobile scenarios, facilitating location-based solution developers to delve into the subject. Regarding the growth of mobile development, especially location-based applications, the Google Maps JavaScript API v3 has deserved rightful attention.

Last but not least, no mapping API has ever been as successful as the Google Maps API without the support of continuously updated and thoroughly handled vector and satellite data. Google has dedicated immense resources to maintaining the unified structure of the vector data and its cartographic quality, and this effort is paying off in terms of its API usage.

What This Book Covers

Chapter 1, Google Maps JavaScript API Basics, instructs you on how to create a simple Google Maps application centered around a main recipe. The map object and its primary options, including map types, will be introduced by adding details to the recipe.

Chapter 2, Adding Raster Layers, presents the addition of external raster data through a series of recipes alongside Google layers such as the Tile, Traffic, Transit, and Weather layers.

Chapter 3, Adding Vector Layers, introduces you to drawing vector features together with the display of external vector sources such as KML and GeoRSS.

Chapter 4, Working with Controls, explains controls in detail. Creating and customizing a custom user interface for both the Web and mobile will be introduced in this chapter.

Chapter 5, Understanding Google Maps JavaScript API Events, describes events in detail to react to map, layer, or marker's behaviors such as zoom end, layer changed, or marker added. Events will add more interactivity to mapping programming.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
Chapter 6, *Google Maps JavaScript Libraries*, explains the libraries that will extend the capabilities of the Google Maps JavaScript API in detail. These libraries have different abilities to increase the power of the Google Maps JavaScript API.

Chapter 7, *Working with Services*, elaborates on services that will extend the Google Maps JavaScript API. These services, including Geocoding and Street View, expose the real power of mapping with the Google Maps JavaScript API.

Chapter 8, *Mastering the Google Maps JavaScript API through Advanced Recipes*, explains the integration of external GIS servers and services with the Google Maps JavaScript API. These includes ArcGIS Server, GeoServer, CartoDB, and Google Fusion Tables with OGC services such as WMS.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
4

Working with Controls

In this chapter, we will cover:

- Adding and removing controls
- Changing the position of controls
- Creating and adding a geolocation control
- Creating a table of contents control for layers
- Adding your own logo as a control

Introduction

This chapter covers the controls that are found in the Google Maps JavaScript API. Generally speaking, controls are UI elements that interact with the user. At a very basic level, they comprise of simple HTML elements or a combination of them.

Controls enable the user to pan the map, zoom in or out, measure distances or areas, and so on. Complex controls involve the administration of multiple overlay layers (introduced in Chapter 2, Adding Raster Layers) in the form of a table of contents (ToC) control, or an editing toolbar for drawing vector features.

The Google Maps JavaScript API presents developers with the opportunity to use and customize built-in controls as well as build custom controls from the ground up.

This chapter will first focus on dealing with built-in controls and their configurations in detail, including the customization of the UI. Then, creating custom controls from very basic to complex ones (such as ToCs) will be covered.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
Adding and removing controls

The Google Maps default UI has several controls that are displayed by default or if certain conditions are met. These include:

- Zoom control
- Pan control
- MapType control
- Scale control
- Street View control
- Rotate control
- Overview Map control

The Google Maps JavaScript API presents the opportunity for developers to opt in or out of these controls or customize them in terms of functionality or look.

In this recipe, we will cover ways to alter the UI by adding or removing built-in controls and how to change their properties through presented options.

Getting ready

The first recipe of Chapter 1, Google Maps JavaScript API Basics, will do our work. We will alter it for this recipe.

How to do it...

You will opt for the appearance of the built-in controls if you perform the following step:

1. Alter the `mapOptions` object as follows:

   ```javascript
   var mapOptions = {
       center: new google.maps.LatLng(43.771094,11.25033),
       zoom: 13,
       mapTypeId: google.maps.MapTypeId.ROADMAP,
       panControl: true,
       scaleControl: false,
   }
   ```

For More Information:

www.packtpub.com/google-maps-javascript-api-cookbook/book
zoomControl: true,
zoomControlOptions: {
    style: google.maps.ZoomControlStyle.SMALL
},
overviewMapControl: true,
overviewMapControlOptions: {
    opened: true
},
mapTypeControl: false

You can have your built-in controls On or Off according to your taste as displayed in the preceding screenshot.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
How it works...

You might have observed that we have added a lot to the `mapOptions` object. This is because you can make the controls visible or invisible in the `google.maps.mapOptions` object:

```javascript
panControl: true,
scaleControl: false,
zoomControl: true,
mapTypeControl: false,
overviewMapControl: true
```

By assigning Boolean (true/false) values, you can display `panControl`, `zoomControl`, and `overviewMapControl`, while `scaleControl` and `mapTypeControl` are hidden.

Some controls appear by default. For instance, we have not mentioned `streetViewControl` in our `mapOptions` object; however, it is displayed in the interface because it is there by default. The built-in controls and their default presence in the UI are as follows:

<table>
<thead>
<tr>
<th>Control name</th>
<th>Default presence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom control</td>
<td>Yes</td>
</tr>
<tr>
<td>Pan control</td>
<td>Yes</td>
</tr>
<tr>
<td>Scale control</td>
<td>No</td>
</tr>
<tr>
<td>MapType control</td>
<td>Yes</td>
</tr>
<tr>
<td>Street View control</td>
<td>Yes</td>
</tr>
<tr>
<td>Rotate control</td>
<td>Yes (for oblique imagery)</td>
</tr>
<tr>
<td>Overview Map control</td>
<td>No</td>
</tr>
</tbody>
</table>

Although `rotateControl` is displayed by default, you might have noticed that it is not found in the interface because it appears only when oblique imagery is shown. Tweaking the `mapOptions` object as follows, we can view the control:

1. Enable `mapTypeControl` so that you can select satellite imagery in the UI, as shown in the screenshot following the code:

   ```javascript
   mapTypeControl: true
   ```

   ![Map Satellite](Map_Satellite)

You can see `mapTypeControl` in the preceding screenshot.

For More Information:

Oblique imagery (45 degree imagery) is served in certain locations, and wherever present, `mapTypeControl` updates itself to include a submenu toggle for displaying oblique imagery.

The `RotateControl` control is displayed between the Pan and Zoom controls. It allows users to rotate the oblique imagery at 90 degree intervals. Also, the Pan control is altered to have a ring, letting us change the heading of the oblique imagery when it is displayed:

---

There's more...

The Google Maps JavaScript API allows us to not only toggle the built-in controls between On and Off, but also customize their properties and styles. For instance:

```javascript
overviewMapControlOptions: {
  opened: true
}
```

---

For More Information:

www.packtpub.com/google-maps-javascript-api-cookbook/book
This sets the Overview Map control in the opened state. Please remember that the Overview Map control's default state is collapsed, and this setting sets the control to opened at the start of your application. You can collapse or open the control whenever you want by pressing the small arrow in the lower-right corner of the control.

For the Zoom control, the options presented in the recipe are as follows:

```javascript
zoomControlOptions: {
    style: google.maps.ZoomControlStyle.SMALL
}
```

This option sets the Zoom control to be styled as small, comprising of two small buttons, one for zoom in and the other for zoom out. Other options for the `style` property for `zoomControlOptions` are:

- `google.maps.ZoomControlStyle.LARGE`
- `google.maps.ZoomControlStyle.DEFAULT`

The `LARGE` option sets the Zoom control to be seen as a long stick where you can traverse between zoom levels. The `DEFAULT` option decides whether to display either the large Zoom controls or the small Zoom controls according to the screen size.

You might have noted that options for controls are handled by objects with the `Options` suffix. In the same manner, there are options for `mapTypeControl` as well within the `MapTypeControlOptions` object. Adding the following lines to the code will make some alterations:

```javascript
mapTypeControl: true,
mapTypeControlOptions: {
    mapTypeIds: [google.maps.MapTypeId.ROADMAP,
                 google.maps.MapTypeId.HYBRID],
    style: google.maps.MapTypeControlStyle.DROPDOWN_MENU
}
```

The settings in the `mapTypeControlOptions` property make `mapTypeControl` offer only the `ROADMAP` and `HYBRID` map types, so you will not be able to select the `SATellite` and `TERRAIN` map types. As you would recall from Chapter 2, Adding Raster Layers, the `mapTypeIds` property not only accepts built-in map types, but also styled map types through a `StyledMapType` object and any tiled image source—either as a base map or overlay map—through the `ImageMapType` object.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
The second property, *style*, sets `mapTypeControl` to be displayed as a drop-down menu instead of a standard horizontal bar. The other options for the *style* property of `mapTypeControlOptions` are:

```javascript
google.maps.MapTypeControlStyle.HORIZONTAL_BAR
google.maps.MapTypeControlStyle.DEFAULT
```

The *DEFAULT* option is for dynamically picking the Zoom control as either a horizontal bar or a drop-down menu according to the screen estate.

**Changing the position of controls**

Google Maps controls have their default positions, and the Google Maps JavaScript API offers a level of flexibility over changing these default positions. You can also position your custom controls according to the places in the UI that are offered as in the following screenshot:

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
The preceding screenshot depicts the possible locations where you can place your controls. It is noteworthy that TOP_LEFT is not equal to LEFT_TOP, with TOP_LEFT the first one on the top.

In this recipe, we will describe how to specify the corresponding positions of controls in the Google Maps UI.

**Getting ready**

This recipe is based on the previous recipe's code; therefore, having that will do most of our work.

**How to do it...**

You can flush the positioning of the controls with the following step:

2. Completely renew the `mapOptions` object as follows:

```javascript
var mapOptions = {
    center: new google.maps.LatLng(43.771094, 11.25033),
    zoom: 13,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    panControl: true,
    panControlOptions: {
        position: google.maps.ControlPosition.TOP_RIGHT
    },
    zoomControl: true,
    zoomControlOptions: {
        style: google.maps.ZoomControlStyle.SMALL,
        position: google.maps.ControlPosition.BOTTOM_CENTER
    },
    mapTypeControl: true,
    mapTypeControlOptions: {
        position: google.maps.ControlPosition.LEFT_TOP
    },
    streetViewControlOptions: {
        position: google.maps.ControlPosition.LEFT_CENTER
    }
};
```

For More Information:

www.packtpub.com/google-maps-javascript-api-cookbook/book
You can change your controls' positions in the map UI as per your taste as you can observe in the preceding screenshot.

**How it works...**

In the `mapOptions` object, we have specified each control's position by its `position` property in its own options as you can see in the following code snippet:

```javascript
mapTypeControlOptions: {
    position: google.maps.ControlPosition.LEFT_TOP
}
```
The following code directs mapTypeControl to be placed in the left-top corner of the map's div element, while panControl is placed in the top-right corner:

```javascript
    panControlOptions:
    {
        position: google.maps.ControlPosition.TOP_RIGHT
    },
```

### Complete listing for control positions

The complete listing for control positions can be found at the Google Maps JavaScript API Reference documentation URL (https://developers.google.com/maps/documentation/javascript/reference#ControlPosition).

---

**Creating and adding a geolocation control**

The Google Maps UI has many built-in controls introduced in previous recipes. These controls serve numerous needs, such as panning, zooming, and changing the map type. However, the user needs are infinite, and the user might be very creative. It is impossible to present a built-in control for every need.

Instead, the Google Maps JavaScript API has presented a path for creating custom controls for every specific need. Custom controls are basically simple HTML elements wrapped in a single element, mostly the `<div>` element.

In this recipe, we will go over the basics of creating a custom control, placing it on the Google Maps UI, and using it through event-handling routines.

### Getting ready

This recipe will be based on the Moving from Web to mobile devices recipe introduced in Chapter 1, Google Maps JavaScript API Basics. Our recipe will utilize the geolocation code extract from this recipe; therefore, it will be helpful to revisit this recipe.

### How to do it...

You will have a brand new custom geolocation control if you perform the following steps:

1. First, create a JavaScript object that will be our custom control at the end (the constructor will take two parameters that will be explained in the later steps):

   ```javascript
   function GeoLocationControl(geoLocControlDiv, map){
   }
   ```

---

**For More Information:**

www.packtpub.com/google-maps-javascript-api-cookbook/book
2. Inside the GeoLocationControl class, set the class property to contain the div element referenced as the first argument in the constructor:
   
   ```javascript
   geoLocControlDiv.className = 'controlContainer';
   ```

3. Inside the GeoLocationControl class, set the internal HTML div element details, including its class attribute, so that this element looks like a button:
   
   ```javascript
   var controlButton = document.createElement('div');
   controlButton.className = 'controlButton';
   controlButton.innerHTML = 'Geolocate';
   ```

4. Add this internal div element (controlButton) to the container div element as follows:
   
   ```javascript
   geoLocControlDiv.appendChild(controlButton);
   ```

5. Add the click event listener for controlButton inside the GeoLocationControl class:
   
   ```javascript
   google.maps.event.addListener(controlButton, 'click', function() {
       if (navigator.geolocation) {
           navigator.geolocation.
               getCurrentPosition(function(position) {
                   var lat = position.coords.latitude;
                   var lng = position.coords.longitude;
                   var devCenter = new google.maps.LatLng(lat, lng);
                   map.setCenter(devCenter);
                   map.setZoom(15);

                   var marker = new google.maps.Marker({
                       position: devCenter,
                       map: map,
                   });
               });
       }
   });
   ```

6. Now, in the ordinary initMap() function that we have used over all the recipes, add the container HTML div element in addition to the standard definition of the map and mapOptions objects:

   ```javascript
   var geoLocationControlDiv = document.createElement('div');
   ```

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
7. Instantiate the custom control class, the GeoLocationControl class inside initMap(), supplying two arguments: the container div element created in the previous step and the map object itself:

```javascript
var geoLocationControl = new GeoLocationControl(geoLocationControlDiv, map);
```

8. Place the custom control among other controls in the map UI:

```javascript
map.controls[google.maps.ControlPosition.RIGHT_CENTER].push (geoLocationControlDiv);
```

You should have a custom control functioning as a geolocation control as seen in the preceding screenshot.

For More Information:

This recipe might seem confusing compared to the preceding recipes, but in essence, there is just one important point to create custom controls in the Google Maps JavaScript API; you can utilize any HTML element to be used as a custom control. In fact, the following simple code extract is sufficient to have a custom control:

```javascript
var controlDiv = document.createElement('div');
map.controls[google.maps.ControlPosition.RIGHT_CENTER].push(controlDiv);
```

This code creates an HTML div element and then adds it to the controls array of the map object. The controls array is a two-dimensional array, the first dimension being the available positions defined in the google.maps.ControlPosition class and the second dimension being the controls. This transparent control with no label inside will do anything as there is no event-handling code for the div element; however, this reality does not change the fact that this is a custom control.

Other details, such as CSS styling, filling in attributes, and event handling, are necessary for a professional custom control to be used for map UI users.

In our recipe, we have chosen to create a JavaScript class to wrap all these details in order to be structural:

```javascript
function GeoLocationControl(geoLocControlDiv, map) {
}
```

Our class constructor makes use of two elements: the container div element and the map object. It needs the reference for the container div element to add the child element controlButton to it:

```javascript
geoLocControlDiv.appendChild(controlButton);
```

The controlButton object (an HTML div element) has to respond to some user-originated events for the custom control to be useful and meaningful:

```javascript
google.maps.event.addDomListener(controlButton, 'click', function() {
});
```

The google.maps.event.addDomListener method acts as an event handler registration, and it works in the same way on every browser. These method- and event-related subjects will be covered in Chapter 5, Understanding Google Maps JavaScript API Events. For now, it is alright to be aware of the click event, which will be listened to by the controlButton object.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
The geolocation code extract from Chapter 1, Google Maps JavaScript API Basics, resides inside the addDomListener method, making use of the Geolocation API of the browser. If there is support for the Geolocation API and if the location is retrieved, a marker is added to the map for this location:

```javascript
var marker = new google.maps.Marker({
    position: devCenter,
    map: map,
});
```

This whole creation of child elements and event-handling logic is enveloped in one JavaScript class constructor, which is called by the following:

```javascript
var geoLocationControl = new GeoLocationControl(geoLocationControlDiv, map);
```

The following is the only other code snippet required to accomplish this task:

```javascript
map.controls[google.maps.ControlPosition.RIGHT_CENTER].push(geoLocationControlDiv);
```

It is worth noting that the controls array takes the container div element as the custom control. Also, bear in mind that controls[google.maps.ControlPosition.RIGHT_CENTER] might already have other controls in other scenarios. We are using push so that the existing controls are not replaced.

## Creating a table of contents control for layers

Table of Contents (ToC) controls such as UI elements are very common in desktop GIS software, such as ArcGIS Desktop, Mapinfo, and Geomedia. Also, their web counterparts make use of ToCs intensively in their UI, including ArcGIS and .Net web components.

The main use of ToCs is to turn On and Off the various raster or vector layers so as to overlay and view multiple strata of data. For vector layers, the options might be enriched by allowing the users to change the symbology of the vector layer with respect to ToCs.

The Google Maps UI does not have a built-in ToC control; however, with the flexibility of building up a custom control, there are virtually infinite possibilities.

The Google Maps JavaScript API allows developers to utilize the third-party base maps such as OpenStreetMaps or display the overlay raster layers on top of base maps (discussed in detail in Chapter 2, Adding Raster Layers). Also, in Chapter 3, Adding Vector Layers, various kinds of vector data has been overlaid in the respective recipes.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
In this recipe, we will only take base maps to be shown on our ToC in order to have an understanding of the structure, including keeping the state of the control and having multiple event handlers for multiple HTML elements wrapped in one control. This structure might be, of course, enriched with the addition of overlay and vector layers.

**Getting ready**

This recipe will make use of the *Using different tile sources as base maps* recipe in Chapter 2, *Adding Raster Layers*. It would be extremely helpful to review this recipe before beginning our current recipe. Also, to understand how a simple custom control is created, the previous recipe will be key.

**How to do it...**

The following are the steps to create a working ToC control inside the Google Maps UI:

1. Create a JavaScript class that will contain all our child controls and event handlers (up to step 12, all code will be embedded in this class constructor):
   
   ```javascript
   function TableOfContentsControl(tocControlDiv, map){
   }
   ```

2. Have this as a variable as it will be out of scope in the event handlers:
   ```javascript
   var tocControl = this;
   ```

3. Set the CSS properties of the container div element inside the class constructor:
   ```javascript
   tocControlDiv.className = 'tocControl';
   ```

4. Set the title of the ToC:
   ```javascript
   var tocLabel = document.createElement('label');
tocLabel.appendChild(document.createTextNode('Base Layers'));
tocControlDiv.appendChild(tocLabel);
   ```

5. Create a radio button for the *OpenStreetMap Base Map*:
   ```javascript
   var osmStuffDiv = document.createElement('div');
   var osmRadioButton = document.createElement('input');
osmRadioButton.type = 'radio';
osmRadioButton.name = 'BaseMaps';
osmRadioButton.id = 'OSM';
osmRadioButton.checked = false;
   ```

For More Information:

Working with Controls

```javascript
var osmLabel = document.createElement('label');
osmLabel.htmlFor = osmRadioButton.id;
osmLabel.appendChild(document.createTextNode('OpenStreetMap Base Map'));

osmStuffDiv.appendChild(osmRadioButton);
osmStuffDiv.appendChild(osmLabel);

6. Create a radio button for the Google Roadmap base map:
```javascript
var roadmapStuffDiv = document.createElement('div');

var roadmapRadioButton = document.createElement('input');
roadmapRadioButton.type = 'radio';
roadmapRadioButton.name = 'BaseMaps';
roadmapRadioButton.id = 'Roadmap';
roadmapRadioButton.checked = true;

var roadmapLabel = document.createElement('label');
roadmapLabel.htmlFor = roadmapRadioButton.id;
roadmapLabel.appendChild(document.createTextNode('Google Roadmap'));

roadmapStuffDiv.appendChild(roadmapRadioButton);
roadmapStuffDiv.appendChild(roadmapLabel);

7. Create a radio button for the Google Satellite base map:
```javascript
var satelliteStuffDiv = document.createElement('div');

var satelliteRadioButton = document.createElement('input');
satelliteRadioButton.type = 'radio';
satelliteRadioButton.name = 'BaseMaps';
satelliteRadioButton.id = 'Satellite';
satelliteRadioButton.checked = false;

var satelliteLabel = document.createElement('label');
satelliteLabel.htmlFor = roadmapRadioButton.id;
satelliteLabel.appendChild(document.createTextNode('Google Satellite'));

satelliteStuffDiv.appendChild(satelliteRadioButton);
satelliteStuffDiv.appendChild(satelliteLabel);
```
8. Put all the radio buttons and their labels in the parent div element:

tocControlDiv.appendChild(osmStuffDiv);
tocControlDiv.appendChild(roadmapStuffDiv);
tocControlDiv.appendChild(satelliteStuffDiv);

9. Create the click event handler for osmRadioButton (the setActiveBasemap and getActiveBasemap methods will be clarified in the following code):

google.maps.event.addDomListener(osmRadioButton, 'click', function() {
    if (osmRadioButton.checked) {
        tocControl.setActiveBasemap('OSM');
        map.setMapTypeId(tocControl.getActiveBasemap());
    }
});

10. Create the click event handler for roadmapRadioButton as follows:

google.maps.event.addDomListener(roadmapRadioButton, 'click', function() {
    if (roadmapRadioButton.checked) {
        tocControl.setActiveBasemap(google.maps.MapTypeId.ROADMAP);
        map.setMapTypeId(tocControl.getActiveBasemap());
    }
});

11. Create the click event handler for satelliteRadioButton:

google.maps.event.addDomListener(satelliteRadioButton, 'click', function() {
    if (satelliteRadioButton.checked) {
        tocControl.setActiveBasemap(google.maps.MapTypeId.SATELLITE);
        map.setMapTypeId(tocControl.getActiveBasemap());
    }
});

12. Outside the TableOfContentsControl class constructor, define a property for keeping the active base map:

TableOfContentsControl.prototype._activeBasemap = null;

For More Information:

www.packtpub.com/google-maps-javascript-api-cookbook/book
13. Define the getter and setter methods for the _activeBasemap property:

```javascript
TableOfContentsControl.prototype.getActiveBasemap =
function() {
    return this._activeBasemap;
};

TableOfContentsControl.prototype.setActiveBasemap =
function(basemap) {
    this._activeBasemap = basemap;
};
```

14. In the initMap() function, define the mapOptions object as follows:

```javascript
var mapOptions = {
    center: new google.maps.LatLng(39.9078, 32.8252),
    zoom: 10,
    mapTypeControlOptions: {
        mapTypeIds: [google.maps.MapTypeId.ROADMAP,
                     google.maps.MapTypeId.SATELLITE, 'OSM']
    },
    mapTypeControl: false
};
```

15. Define the osmMapType object as ImageMapType:

```javascript
var osmMapType = new google.maps.ImageMapType({
    getTileUrl: function(coord, zoom) {
        return 'http://tile.openstreetmap.org/' + zoom + '/' +
                coord.x + '/' + coord.y + '.png';
    },
    tileSize: new google.maps.Size(256, 256),
    name: 'OpenStreetMap',
    maxZoom: 18
});
```

16. Relate the 'OSM' mapTypeId object to the osmMapType object:

```javascript
map.mapTypes.set('OSM', osmMapType);
```

17. Set mapTypeId for startup:

```javascript
map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
```
18. Create the container div element, instantiate the TableOfContentsControl class, and position the container div element as a custom control:

```javascript
var tableOfContentsControlDiv =
document.createElement('div');

var tableOfContentsControl = new
  TableOfContentsControl(tableOfContentsControlDiv, map);

map.controls[google.maps.ControlPosition.TOP_RIGHT].push
  (tableOfContentsControlDiv);
```

You should have your own ToC control as a custom control in your map's UI as observed in the preceding screenshot.
How it works...

This recipe actually carries the same structure as the previous recipe; however, there are HTML elements in the custom control that make it seem more complex. We will take a look at the details bit by bit so that things will become clearer. As in the previous recipe, we have started by creating a JavaScript class constructor that embeds all the details, including the necessary radio buttons and their event handlers:

```javascript
function TableOfContentsControl(tocControlDiv, map){
    // Constructor code...
}
```

The radio button section for `osmRadioButton` embedded in `TableOfContentsControl` is as follows:

```javascript
var osmRadioButton = document.createElement('input');
osmRadioButton.type = 'radio';
osmRadioButton.name = 'BaseMaps';
osmRadioButton.id = 'OSM';
osmRadioButton.checked = false;
var osmLabel = document.createElement('label');
osmLabel.htmlFor = osmRadioButton.id;
osmLabel.appendChild(document.createTextNode('OpenStreetMap Base Map'));
tocControlDiv.appendChild(osmRadioButton);
tocControlDiv.appendChild(osmLabel);
```

```javascript
google.maps.event.addDomListener(osmRadioButton, 'click', function() {
    if (osmRadioButton.checked) {
        tocControl.setActiveBasemap('OSM');
        map.setMapTypeId(tocControl.getActiveBasemap());
    }
});
```

The preceding code extract for `osmRadioButton` is the same for `roadmapRadioButton` and `satelliteRadioButton`. The code creates the radio button and its associated label, adds it to the container `div` element (that is referenced as the first argument of the constructor), and then registers the `click` event for the radio button.

The `click` event checks whether the radio button is checked or not, then—if checked—it sets the active base map as an OSM base map. Then, it uses the active base map information to set `mapTypeId` for the map; this is referenced as the second argument of the constructor.

For More Information:

To set and get the active base map information, two methods are used:

```javascript
setActiveBasemap('OSM')
getActiveBasemap()
```

These methods are defined outside the constructor as:

```javascript
TableOfContentsControl.prototype.getActiveBasemap = function() {
  return this._activeBasemap;
};

TableOfContentsControl.prototype.setActiveBasemap = function(basemap) {
  this._activeBasemap = basemap;
};
```

Here, the _activeBasemap local variable is defined as:

```javascript
TableOfContentsControl.prototype._activeBasemap = null;
```

There is just one tiny but important detail here. For the click event handler to see getter and setter methods of the TableOfContentsControl object, we have added a single line:

```javascript
var tocControl = this;
```

Here, this would be out of scope inside the event handler.

The OpenStreetMap base map section is located in the initMap() function. The details of how to display external base maps are covered in Chapter 2, Adding Raster Layers, so there is no need to go over specific bits and pieces on this.

The final piece of work is actually running the control in the UI. As we do not call the constructor of TableOfContentsControl, nothing will be shown as a custom ToC control. But, before having the ToC control, we have to reserve some estate in the mapOptions object:

```javascript
mapTypeControlOptions: {
  mapTypeIds: [google.maps.MapTypeId.ROADMAP,
               google.maps.MapTypeId.SATELLITE, 'OSM']
},
mapTypeControl=false
```

In mapTypeControlOptions, we list the possible map type IDs for the map in the mapTypeIds property.

However, we do not need maptypeControl anymore as we would have a ToC control instead; therefore, we set the mapTypeControl property to false.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
Then the last phase comes: placing the custom ToC control:

```javascript
var tableOfContentsControlDiv = document.createElement('div');

var tableOfContentsControl = new TableOfContentsControl(tableOfContentsControlDiv, map);
map.controls[google.maps.ControlPosition.TOP_RIGHT].push (tableOfContentsControlDiv);
```

First, we create an arbitrary `div` that will act as a container `div` element for our custom control. Then, we call the constructor of the `TableOfContentsControl` class supplying the container `div` element and the `map` object as arguments. After that, the curtain closes with adding the container `div` element to the two-dimensional controls array that controls the `map` object in its default place in `mapTypeControl`; that is, `google.maps.ControlPosition.TOP_RIGHT`.

### Adding your own logo as a control

The Google Maps JavaScript API has designed the addition of custom controls in a very flexible manner so that you can have a variable type of HTML elements in one HTML `div` element.

Adding your own logo of choice, such as adding your company's logo on top of the map UI in your own application, is a good sign for customization and shows off your work.

In this recipe, we will show a logo as a control in the map UI using the Google Maps JavaScript API.

### Getting ready

This recipe will make use of the very first recipe of Chapter 1, *Google Maps JavaScript API Basics*, as we only need the basics to develop this recipe.

### How to do it...

The following are the steps to display a logo as a custom control in the Google Maps UI:

1. After creating the `map` object in the `initMap()` function, create the container `div` element:
   ```javascript
   var logoDiv = document.createElement("div");
   ```

2. Then, create the HTML `img` element that contains your logo of preference:
   ```javascript
   var logoPic = document.createElement("img");
   logoPic.src = "ch04_logo.PNG";
   logoPic.id = "CompanyLogo";
   ```

For More Information:

3. Insert the `img` element into the container `div` element:

   ```javascript
   logoDiv.appendChild(logoPic);
   ```

4. Add the container `div` element to the `controls` array of the `map` object:

   ```javascript
   map.controls[google.maps.ControlPosition.LEFT_BOTTOM].push(logoDiv);
   ```

You can have the logo of your taste as a custom control in your map’s UI as seen in the preceding screenshot.

**How it works...**

The code of this recipe is actually the simplest form of custom controls in the Google Maps JavaScript API. There is no event handler for the control, and there is no state information in conjunction with the control. The only thing that exists is the control itself, which is the container `div` `logoDiv` element.

For More Information:  
The `logoPic` element and the `img` element keep a reference to the logo file and are embedded in `logoDiv`:

```javascript
var logoPic = document.createElement("img");
logoPic.src = "ch04_logo.PNG";
logoDiv.appendChild(logoPic);
```

Lastly, `logoDiv` is added to the `controls` array in the **LEFT_BOTTOM** position. When you open your application, you can see your logo in your map UI in its designated position.

For More Information:
www.packtpub.com/google-maps-javascript-api-cookbook/book
Where to buy this book


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

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