Map-based Input with Google Fusion Tables

Robert Shepard
University of Nebraska–Lincoln
shepard@huskers.unl.edu

This is the final of three Practical Cartographer's Corner pieces in this issue which come from a graduate course entitled "Cartographic Methods," taught by Michael Peterson (mpeterson@unomaha.edu) at the University of Nebraska at Omaha during the 2013 Spring semester.

INTRODUCTION

Google Fusion Tables has become increasingly popular for its data sharing potential as well as its automatic geocoding service, which converts tabular address data to latitude and longitude coordinates and plots them on a map. Understandably, there is substantial documentation on the web devoted to helping people and organizations take advantage of these features. However, one area that is not well documented is how to enter point data through a map.

There are potentially very many practical applications for entering point locations. For example, a digital history project team using an online GIS to display and store their data may want to receive requests and corrections to their data set from other historians; or perhaps a crowdsourced mapping project relies on input from many users simultaneously, none of whom are GIS technicians. It is possible for a web developer to create a MySQL database on a server and grant users the requisite permissions to write to that database using information drawn from the map application.

Google Fusion Tables provides a viable and simple alternative for entering point data. Outlined here is how to build a basic, cloud-based method of map input that uses a combination of free Google services to acquire, manage, and display spatial data. An intermediate understanding of HTML and JavaScript is required to create a map-based input application with Fusion Tables. A prospective developer will need a web host for displaying pages and a designated Google account, preferably not a personal account.

CREATING AND SYNCING THE FORM AND DATABASES

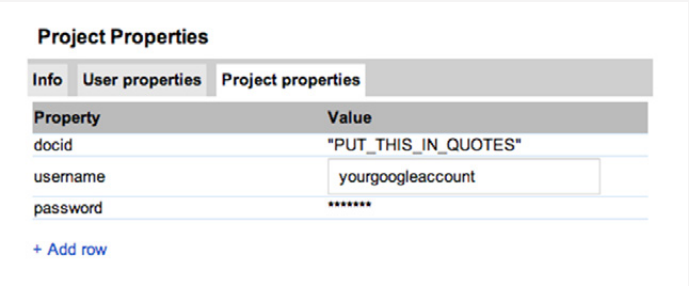
As with all database applications, the structure of the database must be planned in advance. The process will synchronize Google Forms, Google Spreadsheets, and Google Fusion Tables, and so it is desirable to have the fields for all three of these match to ensure that data are being transferred and copied properly. If a developer wants to add a field to a Fusion Table, for example, and expects user input for those fields from a map-based application, a change has to be made at all levels, and this becomes more difficult after data input begins.

The process begins with creating a Google Form in Google Drive and adding text-field questions for "latitude" and "longitude," in addition to any other attribute information that a developer might want to attach to mapped data points. For custom styling options in the final output, some question should involve a categorical variable that can be queried later (the example here asks the user to categorize the type of point, assigning a numerical value of 1, 2 or 3 from a radio button selector to fill the form field). Afterward, the developer should click *Choose response destination* in the menu bar and elect to create a new Google Spreadsheet in which the responses will be stored.

Once the spreadsheet is in place, a Google Fusion Table is created in Google Drive by choosing **Create > Fusion Table,** importing the newly-created spreadsheet that is already synchronized with the form. Because the latitude and longitude fields have to be submitted through the form as text, they come into the Fusion Table as text as well, and this property has to be changed for mapping data. Changes are made to columns in Fusion Tables by choosing **Edit > Change Columns** in the menu bar. Columns are selectable on the left, and, once highlighted, their properties are displayed on the right. Latitude and longitude should be changed to *location* types, *two column location* should be checked. Once changes are saved, and before returning to Google Drive, it is very important to record the unique ID for the Fusion table. The ID is a 40-character alphanumeric string accessed by selecting **File > About** in the menu of the Fusion Table, and this string is needed for accessing the table using the Google Maps API. Finally, the table should be shared.

A little bit of script editing can synchronize the Fusion Table with the form, but first a project has to be prepared and authorized to use scripts on a user's account. This work is done in the script editor, which is accessed in the response spreadsheet by selecting **Tools > Script Editor**. The very first task is setting the project properties for the script, accessed in the menu bar of the script editor, by selecting **File > Project Properties…** The developer has to enter his or her Google account username, and add rows to enter the account password and a "docid," which will be the variable used to identify the Fusion Table ID to which the spreadsheet will sync (Figure 1). It is very important to enter the "docid" field in quotation marks.

Figure 1: Entering project properties to authorize scripts.



After project properties have been set, a custom script can be developed to synchronize the spreadsheet with the Fusion Table. The process can be extremely complicated, even for those who have experience writing scripts. Fortunately, Kathryn Hurley of the Google Fusion Tables team has provided a very useful script specifically intended to synchronize Google Spreadsheets and Google Fusion Tables. The script can be accessed from **http://kh-samples.googlecode.com/svn/trunk/code/appsscript.js** and it should be copied from its source and pasted into the script editor window, replacing all of the placeholder text that may be present. Again, Google has made some minor adjustments to their online documents system since the creation of the script. In particular, the Google Fusion Table ID field is now a string rather than a numeric ID, so it important that the value for docid is contained within quotation marks in the project properties window. That way, the script will properly identify the Fusion Table to which it is supposed to send data. With the script copied into the editor window, the project should then be saved.

The last steps of syncing the form and databases involve automating and authorizing the sync process. In the script editor menu, the developer should choose **Select Function > onFormSubmit**. Then, the steps should be run by selecting the small triangle *Run* button from the menu. The program prompts the developer to authorize. After authorizing the behavior, the developer should select the *Triggers* (clock icon) from the menu and click to add the trigger mechanism: *Run: onFormSumit, Events: From spreadsheet, "on form submit."* After the steps have been completed, input generated from the form will automatically write to the Fusion Table as well as the response spreadsheet.

CREATING THE MAP INPUT SYSTEM

Google Forms automatically generates a webpage for form submissions. Clicking on *View live form* from the spreadsheet and viewing the source code for the form, or grabbing the *embed form in a webpage* link allows a developer to view the HTML code for the standard input form. Within this code are several items of interest. First of all, the URL listed in the form action section is the target spreadsheet to which the form is sending data. Each segment of the form lists a variable name that represents a column in the database to which records are added (e.g., "name = entry_1739129090"). These fields precede the text that indicates the title for each input box. There is also an

```
map = new google.maps.Map(document.getElementById('map-canvas'),
     mapOptions);

google.maps.event.addListener(map, 'click', function(event) {
     addMarker(event.latLng, map);
});

google.maps.event.addListener(map, 'click', function(event) {
     var lat = document.getElementById("latbox").value = event.latLng.lat()
     var lng = document.getElementById("lngbox").value = event.latLng.lng()
```

Example 1: Setting listeners for map click events and acquiring values for the variables.

ID field that identifies each form box relative to other elements on the page. Lastly, an important item to note is how the HTTP POST command is used to submit records to the database. Whether data are being sent using this combination of Google Maps API and Google Forms, or whether a developer is attempting to send data to a Fusion Table directly, the POST command is the standard format for inserting data into Google Drive documents.

The map application input system for the Fusion Tables database therefore is focused on capturing user-defined information in the browser, most importantly latitude and longitude, and routing that information to the form input boxes by way of creating user-defined variables that link up with the ID names of form input variables.

Once the map has been called in an initialize function, listeners have to be set, waiting for users to commit a single click event and capture the coordinates of that click. The code follows a simple `document.getElementByID(xxx).value =` format. Examples 1 and 2 document working excerpts of this procedure using Google Maps API v.3.

With listeners and events set to capture coordinates to variables, the next step is copying the code from a Google Form into the same code as the map (alternatively, it is possible to create a custom HTML form that uses the same POST command to write to the same location). The *id* fields of the form that are intended to represent latitude and longitude should be reset to the event variables that are used to capture latitude and longitude, respectively (Example 2).

Once the changes to the code have been made and saved, input to the form posts to the Fusion Table. Each submission represents one new row in the Fusion Table, and all form fields are likewise attached to the coordinates, much like a true desktop GIS stores data to locations. If desired, the rows can be edited in the Fusion Table without affecting the records in the spreadsheet.

```
<form action="https://docs.google.com/forms/d/ldzJ3x3GcOqWzHKmtCPX859mvQvnAcx0-PD6AH8CzaCA/formResponse"
method="POST">
     <spanclass="ss-q-title">Description</span>
     <input type="text" class="ss-q-short" name="entry_180163717" />
     Latitude:
          <input size="19" type="text" id="latbox" class="ss-q-short" name="entry_1739129090" value="">
     Longitude:
          <input size="19" type="text" id="lngbox" class="ss-q-short" name="entry_1554287385" value="">
```

Example 2: Changing the `id=` *field to correspond to the variables for latitude and longitude.*

```
layer = new google.maps.FusionTablesLayer({
     query: {
          select: 'Number',
          from: 'li_puliVU7S4PwTrdmmYkNsJWUk1B10BxlEhD8'
},

     styles: [
     {where: "'Code' = 1", markerOptions: { iconName: "red_blank"}},
     {where: "'Code' = 2", markerOptions: { iconName: "blu_blank"}},
     {where: "'Code' = 3", markerOptions: { iconName: "grn_blank"}},
     ]});

layer.setMap(map);
```

Example 3: Google Maps API Styling from Fusion Table data.

STYLING THE INPUT

Recent issues of *Cartographic Perspectives* have addressed Fusion Tables styling options, focused on setting the styles in the Fusion Table interface and using "buckets" to establish choropleth map color schemes (Peterson et al. 2012). Adding styles directly to records in the Fusion Table, however, saves the style and associates it directly with the data. While this gives significant control to the developer about how features are displayed, it also restricts the range of potential map applications that can be built by limiting the way that the data can be used. Google Maps API allows application creators to address the issue of styles based upon their own needs. Just as the previous steps have outlined the development of a crowd-sourced database input system, this section anticipates that multiple individuals also want to access the data in their own ways.

Fusion Tables layers are styled using Google Maps API by expanding the layer definition section. Rather than calling the table in one line of code, it is queried and styles are set by defining the symbol for each feature. Point data markers are defined in the marker options by using `iconName:` to reference an icon by name from Google's list of markers and colors. For example, `red_blank` is a plain red marker symbol, while `blu_blank` is a plain blue marker symbol. The process of styling polygon data is similar, except

Figure 2: Markers styled according to Google Fusion Table input.

that `polygonOptions` is used instead of `markerOptions`, and, rather than calling a specific icon, the developer defines a fill color using `fillColor:` and `fillOpacity:` to set an HTML color code and opacity level (from 0 to 1), respectively.

In the example, the `Code` field is the Fusion Table column containing descriptions that users selected for point data when they submitted to the database (Example 3).

Because the previous Google Form input steps provided a radio button with distinct and mutually exclusive categories, a user adding data through the map interface is storing information about feature classes to the database. The end result is a map that effectively recognizes user-defined classes (Figure 2).

SUMMARY

Exploiting Google's services for their robust data management properties does constrain the developer to Google's limitations, such as the exclusive use of a web Mercator projection in all mapping, or the fact that icons come from a pre-selected marker list. On the other hand, a crowd-sourced geospatial data entry project can be established quickly and deployed in the cloud using only the free tools made available by Google. The combination of Google Fusion Tables, Google Forms, and the Google Maps API provides a map input tool that circumvents issues associated with setting up and giving user access to a MySQL database that stores data. Between Google Fusion Tables and Google Maps API, styling options allow a developer to query user inputs and customize symbols automatically based on any set of parameters, potentially minimizing time demands associated with regularly updating and managing data appearances on a map. The methods outlined here are intended to serve as guidance for all parties, although more advanced developers may seek more control over the database and styles. Ultimately, small organizations and individual hobbyists working with crowd-sourced data input are those most likely to benefit from the process described here.

Working code from the input example is available from: **http://robshepard.hostzi.com/samplecode.htm**, and the finished map showing the styled Fusion Table data is available at **http://robshepard.hostzi.com/samplecode2.htm**.

REFERENCES

Peterson, Michael, Kelly Koepsell, Gabriel Pereda, and Spencer Trowbridge. 2012. "Cloud Mapping: Google Fusion Tables." *Cartographic Perspectives* 71:77–90. **http://www.cartographicperspectives.org/index.php/journal/article/view/cp71-peterson-et-al/html**.