Step by step setup to send form data to Google Sheets

May 28, 2015 / by Scott Olmsted

Static sites are ever so useful. Not everything needs Ruby on Rails, or Wordpress, or some other active website framework. I use Middleman (https://middlemanapp.com/) to create static sites, but there are several other tools available.

If one hosts one's static site at a server where PHP is available, then collecting information entered into a form is not difficult. But what if one hosts at Amazon S3, where no scripting language is available?

There exists commercial services for handling form data, such as FormKeep (https://robots.thoughtbot.com/introducing-formkeep) by Thoughtbot. But maybe you don't want to spend $7/month (personal) or $25/month (commercial) to collect form data.

Martin Hawksey showed us at Google Sheets as a Database - INSERT with Apps Script using POST/GET methods (with ajax example) (https://mashe.hawksey.info/2014/07/google-sheets-as-a-database-insert-with-apps-script-using-postget-methods-with-ajax-example/) how to use a combination of Javascript in the browswer and a Google App Script to send form data to a spreadsheet (Google Sheets).

But the steps to do this were not spelled out and it took me some fiddling and experimenting to get it to work. So I thought I would lay out the steps for you here.

The Form

Create a file named `index.html` and copy this into it:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='utf-8'>
    <meta content='IE=edge' http-equiv='X-UA-Compatible'>
    <meta content='width=device-width, initial-scale=1' name='viewport'>
  </head>
  <body>
    <!-- Contact Form - sent to a Google Sheet -->
    <form id='foo'>
      <p>
        <label>Name</label>
        <input id='name' name='name' type='text'>
      </p><p>
        <label>Email Address</label>
        <input id='email' name='email' type='email'>
      </p><p>
        <label>Phone Number</label>
        <input id='phone' name='phone' type='tel'>
      </p><p>
        <label>Message</label>
        <textarea id='message' name='message' rows='5'></textarea>
      </p>
        <div id='success'></div>
        <button type='submit'>Send</button>
    </form>

  <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
  <!-- Custom Theme JavaScript -->
  <script src='google-sheet.js'></script>
</html>
```

The Javascript

Create a file named `google-sheet.js` in the same directory and copy this into it:

```
// Variable to hold request
var request;

// Bind to the submit event of our form
$("#foo").submit(function(event){

    // Abort any pending request
    if (request) {
        request.abort();
    }
    // setup some local variables
    var $form = $(this);

    // Let's select and cache all the fields
    var $inputs = $form.find("input, select, button, textarea");

    // Serialize the data in the form
    var serializedData = $form.serialize();

    // Let's disable the inputs for the duration of the Ajax request.
    // Note: we disable elements AFTER the form data has been serialized.
    // Disabled form elements will not be serialized.
    $inputs.prop("disabled", true);

    // Fire off the request to /form.php
    request = $.ajax({
        url: "SCRIPT URL GOES HERE",
        type: "post",
        data: serializedData
    });

    // Callback handler that will be called on success
    request.done(function (response, textStatus, jqXHR){
        // Log a message to the console
        console.log("Hooray, it worked!");
        console.log(response);
        console.log(textStatus);
        console.log(jqXHR);
    });

    // Callback handler that will be called on failure
    request.fail(function (jqXHR, textStatus, errorThrown){
        // Log the error to the console
        console.error(
            "The following error occurred: "+
            textStatus, errorThrown
        );
    });

    // Callback handler that will be called regardless
    // if the request failed or succeeded
    request.always(function () {
        // Reenable the inputs
        $inputs.prop("disabled", false);
    });

    // Prevent default posting of form
    event.preventDefault();
});
```

The Sheet

Navigate to drive.google.com and click on NEW > Google Sheets to create a new Sheet. Give it a name, perhaps "Form Google Sheets". Put the following names into the first row of the first five columns:

```
Timestamp  name  email phone message
```

The Script

Click on Tools > Script Editor…, which should open a new window and a dialog called 'Google Apps Script'. Click on Create script for > Custom Functions in Sheets. This will create one script called 'Code.gs' containing functions such as SAY_HELLO.

Click on 'Untitled Project' at the top and give this project a name: 'Form Script'.

Highlight all of this script (we are going to replace it) and paste in the following:

```
//  1. Enter sheet name where data is to be written below
        var SHEET_NAME = "Sheet1";

//  2. Run > setup
//
//  3. Publish > Deploy as web app
//    - enter Project Version name and click 'Save New Version'
//    - set security level and enable service (most likely execute as 'me' and access 'anyone
, even anonymously)
//
//  4. Copy the 'Current web app URL' and post this in your form/script action
//
//  5. Insert column names on your destination sheet matching the parameter names of the data
you are passing in (exactly matching case)

var SCRIPT_PROP = PropertiesService.getScriptProperties(); // new property service

// If you don't want to expose either GET or POST methods you can comment out the appropriate
function
function doGet(e){
  return handleResponse(e);
}

function doPost(e){
  return handleResponse(e);
}

function handleResponse(e) {
  // shortly after my original solution Google announced the LockService[1]
  // this prevents concurrent access overwritting data
  // [1] http://googleappsdeveloper.blogspot.co.uk/2011/10/concurrency-and-google-apps-script
.html
  // we want a public lock, one that locks for all invocations
  var lock = LockService.getPublicLock();
  lock.waitLock(30000);  // wait 30 seconds before conceding defeat.

  try {
    // next set where we write the data - you could write to multiple/alternate destinations
    var doc = SpreadsheetApp.openById(SCRIPT_PROP.getProperty("key"));
    var sheet = doc.getSheetByName(SHEET_NAME);

    // we'll assume header is in row 1 but you can override with header_row in GET/POST data
    var headRow = e.parameter.header_row || 1;
    var headers = sheet.getRange(1, 1, 1, sheet.getLastColumn()).getValues()[0];
    var nextRow = sheet.getLastRow()+1; // get next row
    var row = [];
    // loop through the header columns
    for (i in headers){
      if (headers[i] == "Timestamp"){ // special case if you include a 'Timestamp' column
        row.push(new Date());
      } else { // else use header name to get data
        row.push(e.parameter[headers[i]]);
      }
    }
    // more efficient to set values as [][] array than individually
    sheet.getRange(nextRow, 1, 1, row.length).setValues([row]);
    // return json success results
    return ContentService
          .createTextOutput(JSON.stringify({"result":"success", "row": nextRow}))
          .setMimeType(ContentService.MimeType.JSON);
  } catch(e){
    // if error return this
    return ContentService
          .createTextOutput(JSON.stringify({"result":"error", "error": e}))
          .setMimeType(ContentService.MimeType.JSON);
  } finally { //release lock
    lock.releaseLock();
  }
}

function setup() {
    var doc = SpreadsheetApp.getActiveSpreadsheet();
    SCRIPT_PROP.setProperty("key", doc.getId());
}
```

Click on the Save icon. Set the dropdown in the nav bar to 'setup' and click on the right-pointing triangle to its left to run this function. It should show 'Running function setup' and then put up a dialog 'Authorization Required'. Click on Continue. In the next dialog 'Request for permission - Formscript would like to' click on Accept.

In the menus click on File > Manage Versions… We must save a version of the script for it to be called. In the box labeled 'Describe what has changed' type 'Initial version' and click on 'Save New Version', then on 'OK'.

Back to the menus: click on Resources > Current roject's triggers. In this dialog click on 'No triggers set up. Click here to add one now'. In the dropdowns select 'doPost', 'From spreadsheet', and 'On form submit', then click on 'Save'.

Back to the menus: click on Publish > Deploy as web app… For 'Who has access to the app:' select 'Anyone, even anonymous'. Leave 'Execute the app as:' set to 'Me' and Project Version to '1'. Click the 'Deploy' button.

A dialog should appear announcing 'This project is now deployed as a web app'. Copy the Current web app URL from the dialog; it should look something like:

```
https://script.google.com/macros/s/AKfycbw6RTOxn5OT_BIw9Nl_3KoFSXEQEbiKSZCLyombb1YqkGfRKUSz/e
xec
```

Click OK.

Now go back to `google-sheet.js` and replace 'SCRIPT URL GOES HERE' with the URL copied from the dialog.

Display or refresh the `index.html` web page. Enter data into the four fields and click on the 'Send' button. Within a few seconds that data should appear in your Google sheet. If your browser is Google Chrome, right-click in the web page and click on Inspect Element > Console. It should show:

```
Hooray, it worked!
- Object
success
- Object
```

Well done! You can now modify this form and drop it into any web page to collect–at no cost–responses from those who visit your page. With a little research and effort you may be able to get Google Apps to email every time someone submits your form.

I plan to use it to collect names and email addresses of those who visit an MVP page, where the "product" is just a description of the SaaS site I have in mind. I may also use it on a stand-alone blog where I do not want to let readers post comments, but I would rather they send them to me so I can screen, remove spam, edit, and comment on them before putting them up.

--> Comments (http://railsrescue.com/blog/2015-05-28-step-by-step-setup-to-send-form-data-to-google-sheets/)

Ghostery blocked comments powered by Disqus.

comments powered by Disqus (http://disqus.com)

About Us

Rails Rescue makes web applications using Ruby on Rails at a reasonable cost and with a minimum of problems for our clients.

Learn more (/about_us)

Navigation

Projects (/projects)

How We Work (/how_we_work)

About Us (/about_us)

Contact (/contact)

Get in touch

843.779.9640

Click to email us (mailto:info@railsrescue.com)