What do your users think?

Glynn Naughton shows how to capture and report on user feedback from HTML-based documentation

1. Technical communicators want as much user feedback as possible on their documentation to help them improve it. Alas, they typically receive very little. Anything that makes it easier for users to submit feedback is therefore very welcome.

One approach in HTML-based documentation is to add a feedback widget on each page, which is what, Wikipedia, for example, does. At my organisation, we built our own solution based loosely on the Wikipedia approach. Figure 1 shows the widget we added to the bottom of each topic in our WebHelp systems.

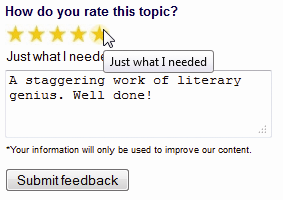


Figure 1. Feedback widget

We used HTML forms, JavaScript and Google spreadsheets to build our own feedback widget. This article describes its architecture and gives step-by-step instructions on creating a bare-bones implementation. You can elaborate and customise this implementation to create a production-quality widget.

Overview

1. The widget is built around an HTML form that links to a Google spreadsheet. When the user clicks the button, the form passes its data to the spreadsheet.
2. The steps involved in creating the widget are:

* Build a custom form with fields for passing data to the spreadsheet.
* Create the Google spreadsheet.
* Build a form on the Google spreadsheet with fields corresponding to the fields on the custom form.

Figure 2. Custom form

<html>

<head>

<title>Feedback Form</title>

<body>

<form method="POST">

<input type="hidden" value="Adding a User"/>

Rating: <input type="text"/><br/>

Comment: <input type="text"/><br/>

<input type="submit" name="submit"/>

</form>

</body>

</html>

* Copy generated code from the Google form to the custom form to link the custom form to the spreadsheet.

1. Two further steps are beyond the scope of this article:

* Format the custom form.
* Attach it to the topics for which you want feedback.

Once you have created your widget and attached it to your documentation pages, you will start collecting data, and the final part of this article suggests some approaches to creating reports from the data and dealing with the user feedback you receive.

Build the custom form

1. The custom form contains three fields: a hidden field to pass the topic name (hard-coded to “Adding a User” in this simplified example), a rating field (we used a scale from 1 to 5 in my organisation) and a comment field.

Figure 2 shows the form’s basic code within a web page, before you add generated code from the Google form. You can copy this HTML to a file and use it to follow along with the rest of this article.

Create the Google spreadsheet

1. Given that my organisation already made extensive use of Google Apps for email and documents, using a Google spreadsheet to collect user feedback was an obvious choice for us. If this is not your case, you will probably need to create a Google account first. When you have done so, and signed in to your account, you see the Google menu across the top of the page. Click **Drive** and create a new spreadsheet.

Build the Google form

1. Google spreadsheets allow you to create forms for data entry. When you create a form, the spreadsheet adds columns that correspond to the form fields, as well as a timestamp column that records when users submitted data. On data submission, Google adds a row to the spreadsheet and adds the data from each form field to the corresponding column.

Your help users never see the Google form. You build it simply to copy some of its generated code into your custom form. In effect, you replace the generated Google form with your own implementation.

To build your form, select Tools > Create a form from the spreadsheet menu to open the form designer.

If your organisation already uses Google Apps, make sure that the Require <organisation-name> sign-in to view this form option at the top of the form designer is unselected. If you leave this option selected, nobody from outside your organisation will be able to submit feedback. (If you are using a personal Google account, this option does not appear.)

When you first open the form designer, the form has two fields with Question Title values “Sample Question 1” and “Sample Question 2”, with Sample Question 1 in edit mode. (Question Title is in effect the field name.) Change these values to “Topic Name” and “Rating” (click the pencil icon to the right of a field to make it editable and Done when you have finished editing it) and leave the **Question Type** (field data type) as “Text”. Click Add Item > Text from the menu to add an additional text field, and set Question Title to “Comment”.

Link the custom form to the spreadsheet

1. Now click the link after You can view the published form here at the bottom of the form designer. You see the Google form as a user would see it, but you only want to harvest some of its code. To view the form code, right-click the form and select View page source, or the equivalent text in your browser.

First, you need to update the *<form>* element in your custom form. Locate the *<form>* element in the Google code and copy its action attribute. Your *<form>* element now looks something like this:

<form method="POST" action="https://docs.google.com/a/gnaughton.com/spreadsheet/formResponse?formkey=dEZoN0QyWjBMa2MtSDJ4UHluaWllOVE6MQ&amp;ifq">

Now, find the <input> elements in the generated code that correspond to the fields on your custom form. You can identify these elements because they are preceded by *<label>* elements with the field names. For example, the Topic Name input element appears something like this:

<div class="ss-item ss-text"><div class="ss-form-entry"><label class="ss-q-title" for="entry\_0">Topic Name

</label>

<label class="ss-q-help" for="entry\_0"></label>

<input type="text" name="entry.0.single" value="" class="ss-q-short" id="entry\_0"></div></div></div>

Figure 3. Custom form linked to Google spreadsheet

<html>

<head>

<title>Feedback Form</title>

<body>

<iframe name="hidden\_iframe" style="display:none"></iframe>

<form method="POST" action="..." target="hidden\_iframe">

<input type="hidden" value="Adding a User" name=”entry.0.single”/>

Rating: <input type="text" name=”entry.1.single”/><br/>

...

</html>

The only thing you need from all this is the name attribute of the *<input>* element. Copy this to the corresponding *<input>* element on your custom form, as follows:

<input value="Adding a User" name="entry.0.single">

Repeat this for the other two *<input>* elements to map them to the corresponding fields on the Google form.

Your form can now send data to the Google spreadsheet. Try it out by opening the HTML file that contains the form in a browser, entering values in the fields and clicking Submit. The values appear in a new row in the Google spreadsheet, together with "Adding a User" in the Topic Name column and the datetime when you submitted the data in the Timestamp column.

The final touch

1. One problem remains: the browser displays Google's standard "Your response has been recorded" page after you submit the data, and the **Submit another response** link on this page takes you back to the Google form that users should not see..

You can bypass this page by adding an invisible inline frame (<iframe>) to the HTML page containing your form, and pointing the form's target attribute to it. (The target attribute tells the form where to display its response after a user submits data.) Figure 3 shows the form code with these final components (shown in bold) in place.

Your form now remains as the current page after you submit data. In practice, you would add an onload event to the inline frame and use it to implement your own post-submit behaviour.

Build a production widget

1. Of course, the widget is not close to production-ready in this crude state. For a production implementation, you need to add JavaScript and CSS to populate invisible fields, define form behaviour and data validation rules, and style the form.

My organisation's form uses JavaScript and CSS to implement ratings stars. The appearance of stars changes when the user hovers over or clicks them. When a user clicks a star, the JavaScript populates a hidden rating field with the corresponding rating from 1 to 5, and the form passes this value to the spreadsheet. The form also uses JavaScript to populate a hidden field that passes the URL of the page, which we use to identify the help topic. (Remember that you hard-coded the topic name in this simplified example.) Nevertheless, we added all that polish and functionality on top of the basic architecture described here.

Deploy the widget

1. Now you need to attach the widget to the topics in your WebHelp systems. We used a Ruby script to add its code just above the closing </body> tag of each topic. Our form has a hidden field with a placeholder value, and the script replaces this placeholder with the name of the help system, which the writer configures in the script settings file. This allows us to identify which help system the feedback comes from.

That kind of processing is beyond the scope of this article, but you can use many programming or scripting languages to accomplish something similar.

Make sense of the feedback

1. You can view users’ feedback directly in the spreadsheet, but when the responses begin to mount up you need some way of reporting on the data to make sense of it.

You can use the Google Visualization API for reporting on a Google spreadsheet. (A quick query on Google will take you to its reference page.) This API implements a simple SQL-like query language, which we used to construct a basic but useful ad hoc query builder (see Figure 4).

The API also implements numerous charts, and you can use these to visualise the results returned by the query language to create graphical summaries of your spreadsheet data.

Act on the feedback

1. So far, we have talked solely about technical solutions for collecting and reporting on feedback, but all this work is pointless if we do not act on what our users tell us. To do that, we must have processes in place alongside the technology.

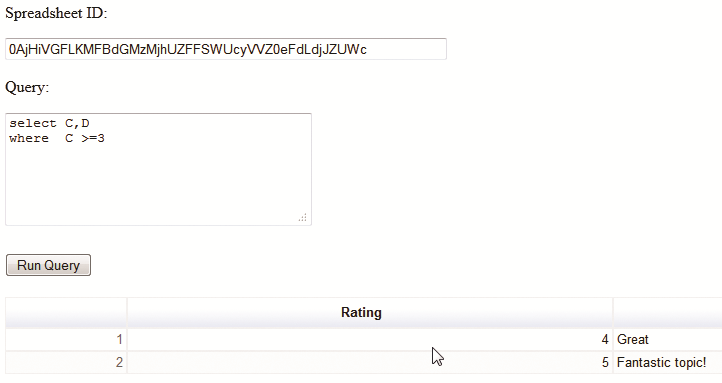


Figure 4. Query builder

In our organisation we had regular meetings with writers to track the status of their feedback items, and we used the Visualization API to report on progress. We could, for example, see the number of items that resulted in changes to the documentation, or the number of items that were more than n days old but where the assigned writer had not yet taken action, or the number of user responses by month.

To make this kind of reporting possible, we added a script to our Google spreadsheet to copy the user responses to a second spreadsheet. (We also restricted access to the initial spreadsheet, which became our pristine production database.) We added columns to the second spreadsheet, such as the name of the writer assigned to that help system, and the status of the feedback item (such as ‘Update in process’, ‘No action necessary’). These additional columns allowed us to track our response to the feedback.

Conclusion

1. This article has described how to build a simple widget that captures user feedback from an HTML page using an HTML form and Google spreadsheet. It has also described how to report on the feedback received, and suggested processes for acting on that feedback.

This solution proved very effective in my organisation. We received over a hundred user responses in the months following its rollout. That is about a hundred more than in the same period before we implemented it. Some of the resulting documentation updates are already in production; others are in the pipeline.

Glynn Naughton has worked for 15 years in the software industry, mainly as a developer and technical communicator.

E: glynn.istc@gnaughton.com