# Supporting users of jQuery plugins: when usability meets coding

Sooner or later we will publish our first jQuery plugin. If our plugin is well-designed and solves one or more coding problems, it is very likely that many users will start using our code in their web projects. Supporting users in this later stage of the development process is crucial for many aspects. Most of all, supporting users and troubleshooting their problems will make the difference between a top-rated jQuery plugin and a plugin that will be soon forgot.

## jQuery Twitter Feed: a (bad) case study

To properly understand what supporting users really implies, we are going to use jQuery Twitter Feed as a case study. My intention is not to popularize my own plugin but to emphasize its many design and usability flaws which create several difficulties to end-users.

This plugin was originally created to “"*fetch and format your latest tweets from Twitter*"”. It was just one of the many jQuery plugins for Twitter with a really plain design (not to say trivial).

When Twitter dropped its support to the version 1.0 of its APIs (June 11, 2013), Twitter Feed stopped working just like many other similar plugins. Since I used this plugin on my websites, I had to find a solution to fix this problem.

So I used the Abraham's *Twitter OAuth* class, created a client PHP class and delegated the JSONP fetching to PHP. It worked, but the entire plugin's design become even clunkier. Now the jQuery code didn't fetch the feed from Twitter anymore but it simply made an AJAX call to the local PHP file. To make the plugin work you had to create a Twitter app, get your access credentials, insert your credentials into the PHP client class and specify your Twitter username and the number of tweets in the instance of the class.

In other words, today we have a quick-and-dirty solution rather than a usable jQuery plugin.

## Who are our users?

According to my personal experience, users roughly fall into two categories:

1. Web designers / developers
2. Common users

Generally speaking, the former category knows what jQuery is and how to write jQuery code. The problem with this category of users is that their programming skills vary greatly. Some users are able to solve their doubts about your plugin by themselves, simply because they've also a firm knowledge of JavaScript so they can dissect your code and modify it accordingly. Unfortunately, this category of users will rarely contact you so that you can't know for sure what solution they've found.

Other users, instead, did literally jump on the jQuery's bandwagon without studying JavaScript properly. These users will probably contact you when it comes to modify your plugin to fit the needs of a particular web project. In this case you should consider if the new feature they request is eligible to be added to the next releases of your plugin or if it's just a special and unique use case.

The latter category is the most troublesome to manage. This category encompasses bloggers, site owners and more broadly any user who simply copy and paste code to add new features to his/her site. Don't be surprised if the most frequently message you'll receive from these users starts with "plugin broken" or "plugin doesn't work".

These persons simply don't know web programming. Be patient.

## Where to publish our plugins

Twitter Feed is hosted on GitHub only. Though this choice seems reasonable to web developers and designers, it's simply unusable for common users because:

- Code examples don't work.
- Users don't know how to directly contact the plugin's author if they've problems with the plugin.

So the best solution is to use both your own web space and GitHub for your plugins. You can put live examples and documentation on your site and link back to your GitHub repository. This approach is followed by many top-rated jQuery plugins, such as jQuery Masonry.

## Documentation and examples

Documentation and examples come together and they're closely connected to each other. If you take a look at the jQuery Twitter Feed's documentation, you'll probably notice some evident errors when it comes to give the proper informations:

- It doesn't explain the necessary steps to create a Twitter app
- It doesn't provide screenshots or any other visual helps to users who don't know the Twitter's developer network
- It doesn't say where to find exactly the consumer key, consumer secret, access token and access token secret strings
- It doesn't specify what the URL chosen during the creation of the app really implies
- It doesn't specify how to include the jQuery's plugin into a web document
- It doesn't show the HTML structure generated by the jQuery code

- It doesn't specify that it's up to the user to create the main CSS styles for the HTML output
- It doesn't explain clearly what the `cache` option really does.

Any missing information in the documentation corresponds to a possible question by the end user. When writing your documentation, forget your skills and knowledge and try to explain the features of your plugin as simply as possible.

Remember the last time you felt frustrated by reading some tutorial or documentation page which was obscure and didn't solve your problem: don't make your users fell the same way.

Examples should cover all the features of your plugin, including all possible combination of your plugin's options. Every example should come with the corresponding code snippet on the same page. Never assume that all users will look at the source code of the page: think defensively and always be prepared for the worst-case scenario.

Usability is made of simplicity, not of assumptions. "*Don't make me think*' is the Steve Krug's motto and also the title of his most famous book. Never forget it.'"

## Answering user's questions

The first question I was asked about jQuery Twitter Feed was whether the plugin worked with WordPress. This question was asked by a web designer who was creating a theme for Theme Forest and needed to create a Twitter user stream.

Since I've successfully inserted Twitter Feed on my WordPress themes, this question looked trivial. Unfortunately, this was not the case. The web designer had problems with PHP and JavaScript and it took two days before he managed to insert the plugin successfully.

What we can learn from this?

- User's questions are sometimes unpredictable
- User's questions sometimes have nothing to do with the original design of our plugin
- User's questions often cover special or unique use cases which our plugin is not able to satisfy without completely changing its design.

The problem is how to respond to these questions. Not responding should be avoided, especially when we think about our web reputation. Trying to satisfy all requests is practically impossible, so we should try to answer in the most polite way when we receive these requests and explain to users that our plugin is designed only to add certain features.

These requests should not be neglected but they can serve as an hint for future enhancements to our plugin.

## Conclusion

Supporting users is the best way we have to create jQuery plugins which not only work smoothly but are also accessible and usable to people.