# How to Create Your Own jQuery Plugin

## Why Create a jQuery Plugin?

Here is a brief list of reasons you might want to create a jQuery plugin:

* Reuse, reuse, reuse
* Encapsulation
* Easy to write
* Maintain chainability
* Public distribution
* Prevent namspace clashing
* They are just plain fun

One of the biggest reasons I think one should make a plugin is to encapsulate your code for reuse across your project. Since plugins are relatively easy to write, there shouldn't be much holding you back from cleaning up your code and making it easier to maintain.

If at some point you bundle up your plugin for public distribution that is great too, but organizing your own personal code is of highest value.

If you do distribute your jQuery plugin to the public, it is a great idea to namespace your plugin so that it doesn't collide with one of the many other plugins that are out there.

If none of the above reasons hit home with you, then just know that jQuery plugins are just plain fun to write, use, and reuse.

## How Does a jQuery Plugin Work?

That might sound like an oversimplied explanation, but when it comes down to it the basic concept of a jQuery plugin is

* Passing a set of DOM elements from a jQuery selection
* Manipulate the DOM elements
* Return the jQuery object so that you maintain chainability

Before we can even start addressing the above three items, we first need to wire-up the framework for our jQuery plugin.

## Plugin Declaration

* jQuery.prototype.tooltip = function(options) {
* $.fn.tooltip = function(options) {
* Self Invoking Anonymous Function

(function($){

$.fn.tooltip = function(options) { ... }

})(jQuery);

## Plugin Parameters

* + You could pass individual parameters to your jQuery Plugin

$.fn.tooltip =

function(text, color) {};

$('#helloWorld')

.tooltip('Hello World!', '#cccccc');

* + However, I am more a proponent of passing a JSON object...

* + Provide a set of Public Accessible Default Plugin Settings for User to Manipulate
  + Accept Custom Settings from the User
  + Merge the Default Settings and Custom Settings and use the Union within the Plugin

$.fn.tooltip = function(options){};

$('#helloWorld').tooltip({

text: 'Hello World!',

color: '#cccccc'

});

## For Each

* + Note: The Plugin may have been passed one or more elements

* + A common technique is to use jQuery's each method to enumerate through the elements

this.each(function() {

var $this = $(this);

//etc...

}

## Public Functions

$.fn.tooltip = function(options) {

return this.each(function() {

var $this = $(this);

var message =

$.fn.tooltip.public();

});

};

$.fn.tooltip.public = function(){

return 'Hello World!';

};

## Private Functions

(function($) {

$.fn.tooltip = function(options) {

var message = private();

};

function private() {

return 'Hello World!';

};

})(jQuery);

## Support the Metadata Plugin

* + Provide Additional Information to a particular element…

Etc…

<li

class="someclass {some: 'data'}">

</li>

<li data="{some:'random',

json: 'data'}"></li>

$.fn.tooltip = function(options) {

var opts = $.extend({},

$.fn.tooltip.defaults, options);

return this.each(function() {

var $this = $(this);

//Element Specific Options

var o = $.meta ? $.extend({},

opts, $this.data()): opts;

//Plugin Logic Here…

}

}

## Allow for Chaining

* + As with most jQuery functions and Plugins, it is desirable to return the jQuery object to facilitate chaining

$(‘a’).css(‘color’, ‘red’).fadeIn();

(function($) {

$.fn.tooltip = function(options) {

return this; //chainability

}

})(jQuery);