Avoiding Conflicts with Other Libraries

The jQuery library and virtually all of its plugins are contained within the jQuery namespace. As a general rule, global objects are stored inside the jQuery namespace as well, so you shouldn't get a clash between jQuery and any other library (like prototype.js, MooTools, or YUI).

That said, there is one caveat: by default, jQuery uses $ as a shortcut for jQuery. Thus, if you are using another JavaScript library that uses the $ variable, you can run into conflicts with jQuery. In order to avoid these conflicts, you need to put jQuery in no-conflict mode immediately after it is loaded onto the page and before you attempt to use jQuery in your page.

Putting jQuery Into No-Conflict Mode

When you put jQuery into no-conflict mode, you have the option of assigning a new variable name to replace the $ alias.

<!-- Putting jQuery into no-conflict mode. -->

<script src="prototype.js"></script>

<script src="jquery.js"></script>

<script>

var $j = jQuery.noConflict();

// $j is now an alias to the jQuery function; creating the new alias is optional.

$j(document).ready(function() {

$j( "div" ).hide();

});

// The $ variable now has the prototype meaning, which is a shortcut for

// document.getElementById(). mainDiv below is a DOM element, not a jQuery object.

window.onload = function() {

var mainDiv = $( "main" );

}

</script>

In the code above, the $ will revert back to its meaning in original library. You'll still be able to use the full function name jQuery as well as the new alias $j in the rest of your application. The new alias can be named anything you'd like: jq, $J, awesomeQuery, etc.

Finally, if you don't want to define another alternative to the full jQuery function name (you really like to use $ and don't care about using the other library's $ method), then there's still another approach you might try: simply add the $ as an argument passed to your jQuery( document ).ready() function. This is most frequently used in the case where you still want the benefits of really concise jQuery code, but don't want to cause conflicts with other libraries.

<!-- Another way to put jQuery into no-conflict mode. -->

<script src="prototype.js"></script>

<script src="jquery.js"></script>

<script>

jQuery.noConflict();

jQuery( document ).ready(function( $ ) {

// You can use the locally-scoped $ in here as an alias to jQuery.

$( "div" ).hide();

});

// The $ variable in the global scope has the prototype.js meaning.

window.onload = function(){

var mainDiv = $( "main" );

}

</script>

This is probably the ideal solution for most of your code, considering that there'll be less code that you'll have to change in order to achieve complete compatibility.

Including jQuery Before Other Libraries

The code snippets above rely on jQuery being loaded after prototype.js is loaded. If you include jQuery before other libraries, you may use jQuery when you do some work with jQuery, but the $ will have the meaning defined in the other library. There is no need to relinquish the $ alias by calling jQuery.noConflict().

<!-- Loading jQuery before other libraries. -->

<script src="jquery.js"></script>

<script src="prototype.js"></script>

<script>

// Use full jQuery function name to reference jQuery.

jQuery( document ).ready(function() {

jQuery( "div" ).hide();

});

// Use the $ variable as defined in prototype.js

window.onload = function() {

var mainDiv = $( "main" );

};

</script>

Summary of Ways to Reference the jQuery Function

Here's a recap of ways you can reference the jQuery function when the presence of another library creates a conflict over the use of the $ variable:

Create a New Alias

The jQuery.noConflict() method returns a reference to the jQuery function, so you can capture it in whatever variable you'd like:

<script src="prototype.js"></script>

<script src="jquery.js"></script>

<script>

// Give $ back to prototype.js; create new alias to jQuery.

var $jq = jQuery.noConflict();

</script>

Use an Immediately Invoked Function Expression

You can continue to use the standard $ by wrapping your code in an immediately invoked function expression; this is also a standard pattern for jQuery plugin authoring, where the author cannot know whether another library will have taken over the $. See the Plugins section for more information about writing plugins.

<!-- Using the $ inside an immediately-invoked function expression. -->

<script src="prototype.js"></script>

<script src="jquery.js"></script>

<script>

jQuery.noConflict();

(function( $ ) {

// Your jQuery code here, using the $

})( jQuery );

</script>

Note that if you use this technique, you will not be able to use prototype.js methods inside the immediately invoked function that expect $ to be prototype.js's $.

Use the Argument That's Passed to the jQuery( document ).ready() Function

<script src="jquery.js"></script>

<script src="prototype.js"></script>

<script>

jQuery(document).ready(function( $ ) {

// Your jQuery code here, using $ to refer to jQuery.

});

</script>

Or using the more concise syntax for the DOM ready function:

<script src="jquery.js"></script>

<script src="prototype.js"></script>

<script>

jQuery(function($){

// Your jQuery code here, using the $

});

</script>