Build your custom JQuery

…………………………………………………….

http://www.ibm.com/developerworks/library/wa-jqplugin/index.html

…………………………………………………

jQuery is a library that extends the JavaScript language. When creating a jQuery plug-in, you're essentially extending the jQuery library, which in turn is extending JavaScript itself. Truly understanding how your plug-in extends the jQuery library requires an understanding of the JavaScript prototype property. Although it is not used directly, the JavaScript prototype property is used behind the scenes through the jQuery property fn, which is a jQuery alias for the native JavaScript prototype property.

To create a new jQuery plug-in using the fn property, simply assign a plug-in name to the fn property and point it to a new function that will act as the constructor function, similar to plain JavaScript. The code in Listing 1 shows how to define a new jQuery plug-in named accordion by using the jQuery object and the fn property and assigning it to a new constructor function.

Fig.1

jQuery.fn.accordion = function() {

// Add plugin code here

};

Listing 1 shows one way to create a jQuery plug-in; there is nothing functionally wrong with the example. However, the recommended way to create a jQuery plug-in is to first create a wrapper function that lets you use the dollar sign ($). By default, the dollar sign can cause conflicts with other JavaScript frameworks. If you wrap your plug-in in a function, conflicts won't occur with other JavaScript frameworks and the use of the dollar sign. The example code in Listing 2 shows how to apply a wrapper function to a jQuery plug-in definition.

(function($) {

$.fn.accordion = function() {

// Add plugin code here

};

})(jQuery);

In above the jQuery keyword is applied to the wrapper function, which lets you use the dollar sign within the plug-in as you do when using the fn property. With the wrapper function in place, you can use the dollar sign in lieu of the jQuery keyword anywhere throughout the plug-in without interfering with other third-party plug-ins. This option provides a way to write less code throughout the plug-in and helps to keep your plug-in code cleaner and easier to maintain.

Maintaining chainability

A benefit of jQuery is that it lets you use any type of selector. However, you must keep in mind that your plug-in can be dealing with several different element types. Using the this keyword lets your plug-in apply the associated functions by accessing each element in a loop regardless of the element type. If you use the return keyword in front of the each loop, you can maintain chainability with your plug-in. Listing 3 shows the each loop being assigned to a function handler and combined with the return keyword.

**Using the return keyword in front of the each loop**

|  |
| --- |
| (function($) {  $.fn.accordion = function() {  return this.each(function() {  // Using return allows for chainability  });  };  })(jQuery); |

With the code above, the example accordion plug-in can be used in a chain of method calls. With chainability—another great jQuery feature—your plug-in can be used in a chain of method calls. For example, the following code shows how an HTML element is faded out and then removed from the document object model (DOM) in a single chain of method calls.

|  |
| --- |
| $("#my-div").fadeOut().remove(); |

**Structuring an accordion**

A typical accordion design includes title bars and related content areas. Definition lists are a great HTML structure for accordions; dt elements are used for titles and dd elements are used for content areas. The HTML structure in Listing 4 is a definition list with four titles and their corresponding content areas.

<dl class="accordion" id="my-accordion">

<dt>Section 1</dt>

<dd>Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam.</dd>

<dt>Section 2</dt>

<dd>Vestibulum a velit eu ante scelerisque vulputate.</dd>

<dt>Section 3</dt>

<dd>Nam mi. Proin viverra leo ut odio. Curabitur malesuada.</dd>

<dt>Section 4</dt>

<dd>Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc.</dd>

</dl>

The accordion class is used to apply styles to the overall definition list, the titles, and the content areas. In the example in Listing 5, the accordion class itself applies a width, border, font family, and font size. You can modify any of the proceeding CSS examples to include your own custom styles, such as color, font family, sizes, and spacing.

**CSS Settings;**

**Accordion CSS class used to define styles for the overall definition list**

|  |
| --- |
| .accordion {  width: 500px;  border: 1px solid #ccc;  border-bottom: none;  font-family: Arial, Helvetica, sans-serif;  font-size: 12px;  } |

You then use the accordion CSS class to define the styles for the titles (dt) and content (dd). The titles and content both include shared styles that define a bottom border and set the margin to 0, which allows the title bars and content areas to rest snuggly against each other, as in Listing 6.

**Shared styles associated with titles and content areas of the accordion**

|  |
| --- |
| .accordion dt,  .accordion dd {  border-bottom: 1px solid #ccc;  margin: 0px;  } |

To make the dt element look more like a title bar, set a background color and add a pointer cursor so it's apparent to users that the title bar is clickable. Various other styles are included in these classes, such as padding, a font size, and a font weight. The dd element has added padding to space out the description a bit from the titles. Listing 7 shows an example.

**CSS classes associated with title and content areas of the accordion**

|  |
| --- |
| .accordion dt {  background: #eaeaea;  cursor: pointer;  padding: 8px 4px;  font-size: 13px;  font-weight: bold;  }  .accordion dd {  padding: 12px 8px;  } |

**Custom coding your plug-in**

To make a functional accordion, you must apply custom code to the jQuery plug-in function that you started creating in the previous section. The accordion plug-in starts by looping through all defined accordions. To define an accordion, use the following jQuery within the HTML document or within an externally embedded JavaScript file.

|  |
| --- |
| $('dl#my-accordion').accordion(); |

For each accordion, you access the associated definition titles using jQuery's children method, which returns an array or dt elements. Apply a click event to the dt elements, then apply a method named reset to each dt. The reset method collapses all dd elements when the accordion first loads. The click event triggers a custom method named onClick when a dt element or title bar is clicked. The custom onClick method looks for all the dt elements within the accordion. It calls a custom hide method, which hides every associated dd element by using the next method to find the dd element next to the dt element, and then slides it up to animate it closed.

After all dd elements are hidden, the dd element associated with the clicked dt element becomes visible using the slideDown method and creates an expanding and contracting animation, as in Listing 8. The final line of code in the onClick method is return false, which ensures that any title bar that is clicked does not exhibit its usual behavior. For example, if you used an anchor element as the title bar, you would want to return false so the user isn't directed to another page or portion of the existing page.

**Custom accordion functions used to create a jQuery plug-in**

|  |
| --- |
| (function($) {  $.fn.accordion = function(options) {  return this.each(function() {  var dts = $(this).children('dt');  dts.click(onClick);  dts.each(reset);  });    function onClick() {  $(this).siblings('dt').each(hide);  $(this).next().slideDown('fast');  return false;  }  function hide() {  $(this).next().slideUp('fast');  }  function reset() {  $(this).next().hide();  }  }  })(jQuery); |

When this accordion plug-in is associated with an HTML definition list structure like the one you previously created, the accordion function will be applied. With accordion functions, when one title bar or dt element is clicked, its content area is opened and any other open content areas are closed. In other words, only one content area can be open at a time.

**Defaults and options**

A jQuery plug-in can include defaults and options. *Options* are essentially arguments that you can pass to your plug-in. Rather than sending several arguments, with options you can send one argument as an *object literal,* which is a standard jQuery practice. When allowing options in your plug-in, it is a best practice to set default options using the defaults object. Like options, *defaults* are an object literal that should include the properties you are allowing to be passed to your plug-in.

For example, if you're allowing a property that can be used to open the first content area of the accordion when it is first loaded, you should include a default for the open property within your plug-in. Use defaults within your plug-in to determine default functions and use options to override default values. When the plug-in receives the options, you can use the $.extend method to do the actual overriding. jQuery's $.extend method merges two or more objects. The example in Listing 9 shows the common practice of using the $.extend method to merge user-defined options with the default options in a custom jQuery plug-in.