jQuery: creating an event-driven, object-oriented accordion

Creating an accordion in jQuery is trivial. The problem with accordions, and other UI elements in general, is that they tend to require procedural code to accomplish a very simple task. Instead, we can actually create an event-driven, fully object-oriented accordion with jQuery by using event data.

On event data

Event data allow us to store objects and other JavaScript data types within the global jQuery `event` object. An example:

```
$('#test').bind('click', {property: 'Test'},
function(event) {
        var str = event.data.property; // Object's
property stored in data
        $(this).text(str);
});
```

The interesting thing to note here is that event data can contain objects which, not surprisingly, may include jQuery's elements.

Custom events

jQuery allows us to define custom events. Custom events are jQuery events tailored to particular needs. For example, you can bind a custom event to an object just to notify a method of the state's change occurred to a property.

The syntax is practically the same as above, with the only exception of the event's name:

```
$('#test').bind('inviewport', function(event) {
        //...
});
```

Both custom and standard event handlers can be invoked by using the `trigger()` and `triggerHandler()` methods.

How an accordion works

When you click on an accordion's heading, three actions must be taken:

1. If the accordion's content is hidden, then slide it down.
2. If the accordion's content is visible, then slide it up.
3. All other accordion's contents must be hidden except the current one.

Usually we use the following procedural approach:

```
$('h3', 'div.accordion').each(function() {

        var $header = $(this);
        var $content = $header.next();

        $header.click(function() {

                $('div.accordion-
content').slideUp(600); // hide all contents

                if($content.is(':hidden')) {
```

```
                    $content.slideDown(600);

                } else {

                    $content.slideUp(600);

                }


        });

});
```

The problem with this approach lies in the fact that is not reusable nor flexible. What we actually need is a reusable object that we can use and reuse in many different contexts and projects with a few minor changes.

The Accordion object

Here's our event-driven, object-oriented implementation:

```
var Accordion = {
    Elements: {
        titles: $('h3', 'div.accordion')
    },
    Events: {
        open: function(evt) {
            if (evt.data.element.is(':hidden')) {
                evt.data.element.slideDown(600);
            } else {

```

```
                this.close();

            }
        },
        close: function(evt) {

            if (evt.data.element.is(':visible')) {
                evt.data.element.slideUp(600);

            } else {

                this.open();

            }
        },
        bind: function() {

            Accordion.Elements.titles.each(function()
{

                var $header = $(this);
                var $content = $header.next();

                $header.bind('open', {
                    element: $content
                }, Accordion.Events.open);
                $header.bind('close', {
                    element: $content
                }, Accordion.Events.close);

            });

        }
    },
```

```
    fn: {

        attachEvents: function() {

            Accordion.Events.bind();

            Accordion.Elements.titles.on('click',
function() {

                $('div.accordion-
content').slideUp(600);
                $(this).trigger('open');

            });


        }
    },

    init: function() {

        Accordion.fn.attachEvents();
    }
};


Accordion.init();
```

We've attached each accordion content to the `event` object of the accordion headings. By doing so, the element stored as a data object is immediately available after being cached.

We've also created two helper methods which slide up or down contents by using event data. The final result is a robust implementation that can be adapted and extended in many different use-case scenarios.

You can see the demo on this page.