Dynamically editing web content inline, using JavaScript and AJAX

Posted At : 10 February 2010 15:30 | Posted By : Shaun McCran
Related Categories: JQuery, Coldfusion, Javascript, AJAX

Most of us are familiar with the standard method of displaying data in a tabulated fashion, selecting a record, and populating the form that follows. What about editing the content directly into a template that mirrors the actual live version of a page?

This article examines how to edit web content directly inline, and commit it back to a server using an AJAX post request.

The main catalyst for this is that clients that use a content management system do not often have a clear image of how their content will look online. The traditional form layout for entering text does not lend itself well to representing the actual content in the format it is display in.

The aim here is to build a flexible system that allows for inline content editing, and saves it gracefully to a server based database.

I will start by saying thank you to Peter-Paul Koch. His article here (http://www.quirksmode.org/dom/cms.html) on making content editable was invaluable, and a lot of this is based on his theory.

We start by setting a value "editing" to false. This is the default for the page, as the user isn't editing anything when the page loads.

```
var editing  = false;

if (document.getElementById && document.createElement) {
        var butt = document.createElement('BUTTON');
      var buttext = document.createTextNode('Save');
              butt.appendChild(buttext);
               butt.onclick = saveEdit;
                      }

function catchIt(e) {
```

```
                        if (editing) return;
        if (!document.getElementById || !document.createElement) return;
                if (!e) var obj = window.event.srcElement;
                        else var obj = e.target;
                       while (obj.nodeType != 1) {
                              obj = obj.parentNode;
                              }
          if (obj.tagName == 'TEXTAREA' || obj.tagName == 'A') return;
            while (obj.nodeName != 'P' && obj.nodeName != 'HTML') {
                              obj = obj.parentNode;
                              }
                    if (obj.nodeName == 'HTML') return;
                           var x = obj.innerHTML;
                var y = document.createElement('TEXTAREA');
                           var z = obj.parentNode;
                            z.insertBefore(y,obj);
                          z.insertBefore(butt,obj);
                             z.removeChild(obj);
                                y.value = x;
                                 y.focus();
                              editing = true;
                                 getId(e)
                                }

                             function getId(e) {
                                    var targ;
                               if (!e) var e = window.event;
                              if (e.target) targ = e.target;
                      else if (e.srcElement) targ = e.srcElement;
                      if (targ.nodeType == 3) // defeat Safari bug
                                 targ = targ.parentNode;
                           thisTarget = e.target.id;

                                    }

                             function saveEdit() {
            var area = document.getElementsByTagName('TEXTAREA')[0];
                          var y = document.createElement('P');
       // set the id back to the original value as the real one is destroyed
                           y.setAttribute('id', thisTarget);

                             var z = area.parentNode;
                            y.innerHTML = area.value;
                             z.insertBefore(y,area);
                               z.removeChild(area);
           z.removeChild(document.getElementsByTagName('button')[0]);
                                editing = false;
     // action the server request, first var is the value, second var is the id
                          saveToServer(y.innerHTML,thisTarget);
                                    }

                 function saveToServer(valToCommit,fieldname) {
                                //alert(valToCommit);
"view/appeals/act_commitChange.cfm", { newValue: valToCommit, field: fieldname, a

                                function(data){
                                     alert(data);
                                      });

                           document.onclick = catchIt;
```

I wont go into massive depth on a line by line basis but Peter's article does break this down a lot. The premise is that there is a function catchit(), which will intercept any click events. It will then check that the event was triggered from a 'P' tag, which is our defining element for editable content. IE any P elements hold editable content. It will then remove the P html container, replacing it with a textarea, and re insert the P tags previous html content using the innerHTML JavaScript function.

In this way we can create editable inline textareas within the framework of our page.

The next step is to create a save function. The function 'saveToServer{}' take several arguments. It needs the value to commit, IE what the amended text string is, and the fieldname. Each 'P' tag has an id that I am matching to a data field. In this way if there are multiple p tags in a display they can each be attributed to a specific storage field in a database.

Because we are destroying the 'P' tag when we create the textarea we need to re assign the id to it when we save. We can do this by using the JavaScript function 'setAttribute'. The setAttribute function is used to set the value of an attribute on an object. It is typically used along with objects returned by document.getElementById to assign a new value to the object's attribute.

```
// set the id back to the original value as the real one is destroyed
y.setAttribute('id', thisTarget);
```

If we don't do this then the recreated 'P' tag no longer has an id attribute, so will error on any subsequent updates.

Next we use a JQuery Post function to post the values through an AJAX request.

```
post("commitChange.cfc", { newValue: valToCommit, field: fieldname, appeal: intId
```

This will post the values to the cfml CFC "commitChange.cfc", which handles them in a function.

This will allow you to perform seamless inline edits to the display layer, and commit them back to a server, so they are stored in real time.

There is an example of this here . (Minus the storing). You can track the AJAX post using a tool like charles http proxy, or firefox's firebug.

Now, to write a nice JQuery response handler to fade the returned massage in and out.