**Drafting the HTML**

I like to create a rough draft just using HTML and CSS, leaving the scripting for last. I'll write the HTML first to structure the modal, even though the final HTML will be generated by the script.

<div id='overlay'></div>

<div id='modal'>

<div id='content'>No JavaScript Yet!</div>

<a href='#' id='close'>close</a>

</div>

This modal will have two root elements: #overlay and #modal. The overlay is going to be a semi-transparent layer that fills the visitor's viewport. This turns the modal into a focal point, and reduces focus for the rest of the page.

**The CSS**

#overlay {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

background: #000;

opacity: 0.5;

filter: alpha(opacity=50);

}

The overlay has a fixed position that is relative to the visitor's viewport. The 100% width and height refer to the viewport size, so we get complete coverage no matter what part of the page the visitor is actually viewing. Unfortunately, fixed position is not supported in iOS 4 or below. IE8 and IE7 do not support opacity, so an alpha filter is used instead.

#modal {

position:absolute;

background:url(tint20.png) 0 0 repeat;

background:rgba(0,0,0,0.2);

border-radius:14px;

padding:8px;

}

#content {

border-radius:8px;

background:#fff;

padding:20px;

}

The important thing here is that #modal will be absolutely positioned, the rest of the rules just provide a nice default appearance.

The RGBa background will give a black fill (0,0,0) at 20% opacity (0.2), which will overwrite the previous background property. IE8 and IE7 do not support RGBa, so they will continue to use the previous background property which uses a semi-transparent PNG background image.

I gave #modal a larger border-radius than #content. This made a border that is natural looking and consistent in thickness. IE8 and IE7 do not support border-radius and will instead display the normal square corners.

#close {

position:absolute;

background:url(close.png) 0 0 no-repeat;

width:24px;

height:27px;

display:block;

text-indent:-9999px;

top:-7px;

right:-7px;

}

The close button is absolutely positioned relative #modal, since #modal is it's closest ancestor element with positioning. Since I used an anchor element for the close button, it's default display of inline must be changed to block in order to apply a fixed width and height to it. Text-indent moves the anchor's text offscreen so that it will not overlay the background image.

**See The** [**Work In Progress**](http://www.jacklmoore.com/demo/modal/modal1.html)

**The Script**

I would like the modal to handle the following things:

* Create it's own HTML and append it to the document
* Display specified content
* Open on command
* Close on command
* Be positioned in the center of the viewport
* Stay centered if the viewport is resized

Let's stump out the function and variable names:

var modal = (function(){

var

method = {},

$overlay,

$modal,

$content,

$close;

// Center the modal in the viewport

method.center = function () {};

// Open the modal

method.open = function (settings) {};

// Close the modal

method.close = function () {};

return method;

}());

Here we have a function that is immediately executed and returns an object containing the methods needed to control the modal. This object is assigned to the modal variable. The rest of the variables are references to their corresponding elements in the document. Now lets fill in the missing code.

**Append The HTML**

$overlay = $('<div id="overlay"></div>');

$modal = $('<div id="modal"></div>');

$content = $('<div id="content"></div>');

$close = $('<a id="close" href="#">close</a>');

$modal.hide();

$overlay.hide();

$modal.append($content, $close);

$(document).ready(function(){

$('body').append($overlay, $modal);

});

This generates the same HTML structure used earlier in the guide, now created and added to the document by our script.

**Center Method**

method.center = function () {

var top, left;

top = Math.max($(window).height() - $modal.outerHeight(), 0) / 2;

left = Math.max($(window).width() - $modal.outerWidth(), 0) / 2;

$modal.css({

top:top + $(window).scrollTop(),

left:left + $(window).scrollLeft()

});

};

Here we balance the distance between each side of the viewport and each side of the modal. jQuery's .outerWidth() and .outerHeight() are used because borders and padding should also be factored in. I used Math.max() so that I could be sure that no value returned would be less than 0. This means that if the modal is larger than the visitor's viewport, it will be positioned in the top left of the viewport rather running beyond that where it could be potentially being inaccessible. $(window).scrollTop() returns the position of the vertical scrollbar, which is the distance from the top of the document to the top of the viewport. This needs to be added to account for any scrolling that has happened.

**Open Method**

method.open = function (settings) {

$content.append(settings.content);

$modal.css({

width: settings.width || 'auto',

height: settings.height || 'auto'

})

method.center();

$(window).bind('resize.modal', method.center);

$modal.show();

$overlay.show();

};

The open method accepts an object of our settings. It can contain three properties: content, width, height. Content can be text, HTML, or an element/jQuery object. We append the data from settings.content to #content, and give #modal a width and height if either was specified in the settings object. Method.center() is called to center the modal in the viewport and is bound to the window's resize event. This will cause the modal to re-center itself whenever the viewport changes size. I made use of jQuery's [event namespacing](http://docs.jquery.com/Namespaced_Events) so that later, when we unbind the event, we can make sure not to disturb any other resize events that other scripts may have bound to the window. The last thing we do is reveal #modal and #overlay.

**Close Method**

method.close = function () {

$modal.hide();

$overlay.hide();

$content.empty();

$(window).unbind('resize.modal');

};

Here we hide the #modal and the #overlay and remove the content. Then we unbind the resize event because there is no need for it when the modal is closed. Outside of the method

**Close Button**

$close.click(function(e){

e.preventDefault();

method.close();

});

The default action of the click event is canceled (the browser from redirecting to the link's href) by calling e.preventDefault(), then the close method is called.

**Examples Calling The Modal**

modal.open({content: "Howdy"});

modal.open({content: "<p>Howdy</p>"});

modal.open({content: $("<p>Howdy</p>"), width: "500px", height: "200px"});

// Ajax

$.get('ajax.html', function(data){

modal.open({content: data});

});