JavaScript

Overview

HUBzero comes with the jQuery Javascript Framework included by a system plugin. jQuery is not only a visual effects library–it also support Ajax request and JSON notation, table sort, drag & drop operations and much more. All current HUBzero JavaScripts are built on this framework.

Directory & Files

The jQuery framework can be found within the /core/assets/js directory. It is a compressed version used for production. An uncompressed version may be found at jquery.com.

```
/hubzero
  /media
    /system
      /js
        jquery.js
```

Most HUBzero templates will include some scripts of their own for basic setup, visual effects, etc. These are generally stored in (but not limited to) a sub-directory, named /js, of the template's main directory.

```
/hubzero
  /media
    /system
      /js
        jquery.fancybox.js
        jquery.fileuploader.js
        jquery.ui.js
```

Of the scripts commonly found in a HUBzero template, hub.js is perhaps the most important and it is strongly encouraged that developers include these files in their template.

hub.js

```
//-------------------------------------------------------------
//  Create our namespace
//-------------------------------------------------------------
var HUB = HUB || {};
HUB.Base = {};

var alertFallback = true;
if (typeof console === "undefined" || typeof console.log === "undefine
d") {
 console = {};
 console.log = function() {};
}

//-------------------------------------------------------------
//  Various functions - encapsulated in HUB namespace
//-------------------------------------------------------------
if (!jq) {
 var jq = $;

 $.getDocHeight = function(){
  var D = document;
  return Math.max(Math.max(D.body.scrollHeight, D.documentElement.scro
llHeight), Math.max(D.body.offsetHeight, D.documentElement.offsetHeigh
t), Math.max(D.body.clientHeight, D.documentElement.clientHeight));
 };
} else {
 jq.getDocHeight = function(){
  var D = document;
  return Math.max(Math.max(D.body.scrollHeight, D.documentElement.scro
llHeight), Math.max(D.body.offsetHeight, D.documentElement.offsetHeigh
t), Math.max(D.body.clientHeight, D.documentElement.clientHeight));
 };
}

var template = {};

jQuery(document).ready(function(jq){
 var $ = jq,
  w = 760,
  h = 520,
  templatepath = '/templates/template/';

 // Set focus on username field for login form
 if ($('#username').length > 0) {
```

```
  $('#username').focus();
 }

 // Turn links with specific classes into popups
 $('a').each(function(i, trigger) {
  if ($(trigger).is('.demo, .popinfo, .popup, .breeze')) {
   $(trigger).on('click', function (e) {
    e.preventDefault();

    if ($(this).attr('class')) {
     var sizeString = $(this).attr('class').split(' ').pop();
     if (sizeString && sizeString.match(/d+xd+/)) {
      var sizeTokens = sizeString.split('x');
      w = parseInt(sizeTokens[0]);
      h = parseInt(sizeTokens[1]);
     }
     else if(sizeString && sizeString == 'fullxfull')
     {
      w = screen.width;
      h = screen.height;
     }
    }

    window.open($(this).attr('href'), 'popup', 'resizable=1,scrollbars
=1,height='+ h + ',width=' + w);
   });
  }
  if ($(trigger).attr('rel') && $(trigger).attr('rel').indexOf('extern
al') !=- 1) {
   $(trigger).attr('target', '_blank');
  }
 });

 if (jQuery.fancybox) {
  // Set the overlay trigger for launch tool links
  $('.launchtool').on('click', function(e) {
   $.fancybox({
    closeBtn: false,
    href: templatepath + 'images/anim/circling-ball-loading.gif'
   });
  });

  // Set overlays for lightboxed elements
  $('a[rel=lightbox]').fancybox();
 }
```

```
 // Init tooltips
 if (jQuery.ui && jQuery.ui.tooltip) {
  $(document).tooltip({
   items: '.hasTip, .tooltips',
   position: {
    my: 'center bottom',
    at: 'center top'
   },
   // When moving between hovering over many elements quickly, the too
ltip will jump around
   // because it can't start animating the fade in of the new tip unti
l the old tip is
   // done. Solution is to disable one of the animations.
   hide: false,
   content: function () {
    var tip = $(this),
     tipText = tip.attr('title');

    if (tipText.indexOf('::') != -1) {
     var parts = tipText.split('::');
     tip.attr('title', parts[1]);
    }
    return $(this).attr('title');
   },
   tooltipClass: 'tooltip'
  });

  // Init fixed position DOM: tooltips
  $('.fixedToolTip').tooltip({
   relative: true
  });
 }

 //test for placeholder support
 var test = document.createElement('input'),
  placeholder_supported = ('placeholder' in test);

 //if we dont have placeholder support mimic it with focus and blur ev
ents
 if (!placeholder_supported) {
  $('input[type=text]:not(.no-legacy-placeholder-
support)').each(function(i, el) {
   var placeholderText = $(el).attr('placeholder');

   //make sure we have placeholder text
   if (placeholderText != '' && placeholderText != null) {
```

```
    //add plceholder text and class
    if ($(el).val() == '') {
     $(el).addClass('placeholder-support').val(placeholderText);
    }

    //attach event listeners to input
    $(el)
     .on('focus', function() {
      if ($(el).val() == placeholderText) {
       $(el).removeClass('placeholder-support').val('');
      }
     })
     .on('blur', function(){
      if ($(el).val() == '') {
       $(el).addClass('placeholder-support').val(placeholderText);
      }
     });
   }
  });

  $('form').on('submit', function(event){
   $('.placeholder-support').each(function (i, el) {
    $(this).val('');
   });
  });
 }
};
```

HUB Namespace

Typically the template will include a file (hub.js) that first establishes a HUB namespace and then proceeds through some basic setup routines. All HUBzero built components, modules, and templates that employ JavaScript place scripts within this HUB namespace. This helps prevent any naming collisions with third-party libraries. While it is recommended that any scripts you may add to your code is also placed within the HUB namespace, it is not required.

Note: When not using jQuery, the template will include a global.js file that establishes the HUB namespace.

Some additional sub-spaces for further organization are available within the HUB namespace. Separate spaces for Modules, Components, and Plugins are created. Once again, this further helps avoid possible naming/script collisions. Additionally, one more Base space is created for

basic setup and utilities that may be used in other scripts.

```
//  Create our namespace
if (!HUB) {
 var HUB = {};

 // Establish a space for setup/init and utilities
 HUB.Base = {};

 // Establish sub-spaces for the various extensions
 HUB.Components = {};
 HUB.Modules = {};
 HUB.Plugins = {};
}
```

To demonstrate adding code to the namespace, below is code from a script in a component named com_example.

```
//  Create our namespace
if (!HUB) {
 var HUB = {};

 // sub-space for components
 HUB.Components = {};
}

// The Example namespace and init method
HUB.Components.Example = {
 init: function() {
  // do something
 }
}

// Initialize the code (jQuery)
jQuery(document).ready(function($){
 Components.Example.init();
});
```

Loading From An Extension

Components

Occasionally a component will have scripts of its own. Pushing JavaScript to the template from a component is quite easy and involves only a few lines of code.

```
HubzeroDocumentAssets::addComponentScript('com_example');
```

First, we load the HubzeroDocumentAssets class. Next we call the static method addComponentScript, passing it the name of the component as the first (and only) argument. This will first check for the presence of the style sheet in the active template's overrides. If found, the path to the overridden script will be added to the array of scripts the template needs to include in the <head>. If no override is found, the code then checks for the existence of the script in the component's directory. Once again, if found, it gets pushed to the template.

Modules

Loading Javascript from a module works virtually the same as loading from a component save one minor difference in code. Instead of calling the addComponentScript method, we call the addModuleScript method and pass it the name of the module.

```
HubzeroDocumentAssets::addModuleScript('mod_example');
```

Plugins

Loading Javascript from a plugin works similarly to loading from a component or module but instead we call the addPluginScript method and pass it the name of the plugin group **and** the name of the plugin.

```
HubzeroDocumentAssets::addPluginScript('examples', 'test');
```

Plugin Javascript must be named the same as the plugin and located within a directory of the same name as the plugin inside the plugin group directory.

```
/plugins
  /examples
    /test
      test.css
```

```
    test.php
    test.xml
```

View Helpers (all extensions)

Modules, Component, and plugin views now have helpers for pushing Cascading StyleSheets and JavaScript assets to the document. Each method automatically looks for overrides within the current, active template, taking out the busy work of checking yourself each time assets are added. The method names are short, accept a range of options, and allow for method chaining, all tailored for brevity and ease of use.

The css() method provides a quick and convenient way to attach stylesheets. For components, it accepts two arguments:

1. The name of the stylesheet to be pushed to the document (file extension is optional). If no name is provided, the name of the component or plugin will be used. For instance, if called within a view of the component "com_tags", the system will look for a stylesheet named "tags.css".
2. The name of the extension to look for the stylesheet. For components, this will be the component name (e.g., com_tags). For plugins, this is the name of the plugin folder and requires the third argument of plugin group (type) be passed to the method.
3. *Plugin views only.* The name of the plugin.

Example:

```
<?php
// Push a stylesheet to the document
$this->css()
      ->css('another')  // Extension (.css) is optional
      ->css('tags.css', 'com_tags');  // Load CSS from another compone
nt
?>
... view HTML ...
```

Along with file names, the method also accepts style declarations:

```
<?php
// Push a stylesheet to the document
$this->css('.foo {
 color: #000;
```

```
}');
?>
... view HTML ...
```

Similarly, a js() method is available for pushing javascript assets to the document. The arguments accepted are the same as the css() method described above.

```
<?php
// Push some javascript to the document
$this->js()
      ->js('another');
?>
... view HTML ...
```

And, just as the css() method accepts style declarations, the js() method accepts script declarations:

```
<?php
// Push some javascript to the document
$this->js('
 jQuery(document).ready(function($){
  $("a").on("click", function(e){
   console.log($(this).attr("href"));
  });
 });
');
?>
... view HTML ...
```