Type of Submission: Article

Title: Developing an IBM Content Navigator Plug-in using non-Dojo Javascript Widget library such as jQuery UI

Subtitle: Developing ICN Plug-in using jQuery UI

Keywords: IBM Content Navigator, IBM Case Mananger, plugin, non-dojo, jQuery, ICN, ICM

Author: Chad Lou
Job Title: Senior Software Engineer, IBM Case Manager and Enterprise Content Management
Email: chadlou@us.ibm.com
Bio: Senior Software Engineer of the IBM Case Manager Client.
Company: IBM
Author: Brett Morris
Job Title: Senior Technical Staff Member, IBM Enterprise Content Management Client Development
Email: bwmorris@us.ibm.com
Bio: Architect of IBM Content Navigator experience platform
Company: IBM

Abstract: This article describes a jQuery plug-in to extend IBM Content Navigator and IBM Case Manager using non-Dojo javascript library

Developing an IBM Content Navigator Plug-in using non-Dojo Javascript Widget library such as jQuery UI

Chad Lou, Brett Morris

Introduction

This is Part II of our research into using non-Dojo toolkit assets in IBM Content Navigator and IBM Case Manager.

In Part I, we described how to use the IBM Content Navigator JavaScript API from a standalone web page created using the jQuery toolkit. This standalone web page was added to a sub-directory of the IBM Content Navigator web application. The Dojo Toolkit is shipped with IBM Content Navigator. The jQuery toolkit was copied separately onto the Navigator web application. The standalone web page loads jQuery from the same web application on the same application server host. It uses a combination of Dojo and jQuery to execute IBM Content Navigator JavaScript APIs.

While the standalone web page shows that we can mix jQuery API calls, such as AJAX calls, with Dojo-based APIs there are more features that come with jQuery. One major aspect of jQuery is jQuery UI. The jQuery UI is a set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library.

What we did in Part I:

- Loaded jQuery from Dojo
- Mixed Dojo and jQuery
- Made IBM Content Navigator API calls from Dojo and jQuery

Limitations in Part I:

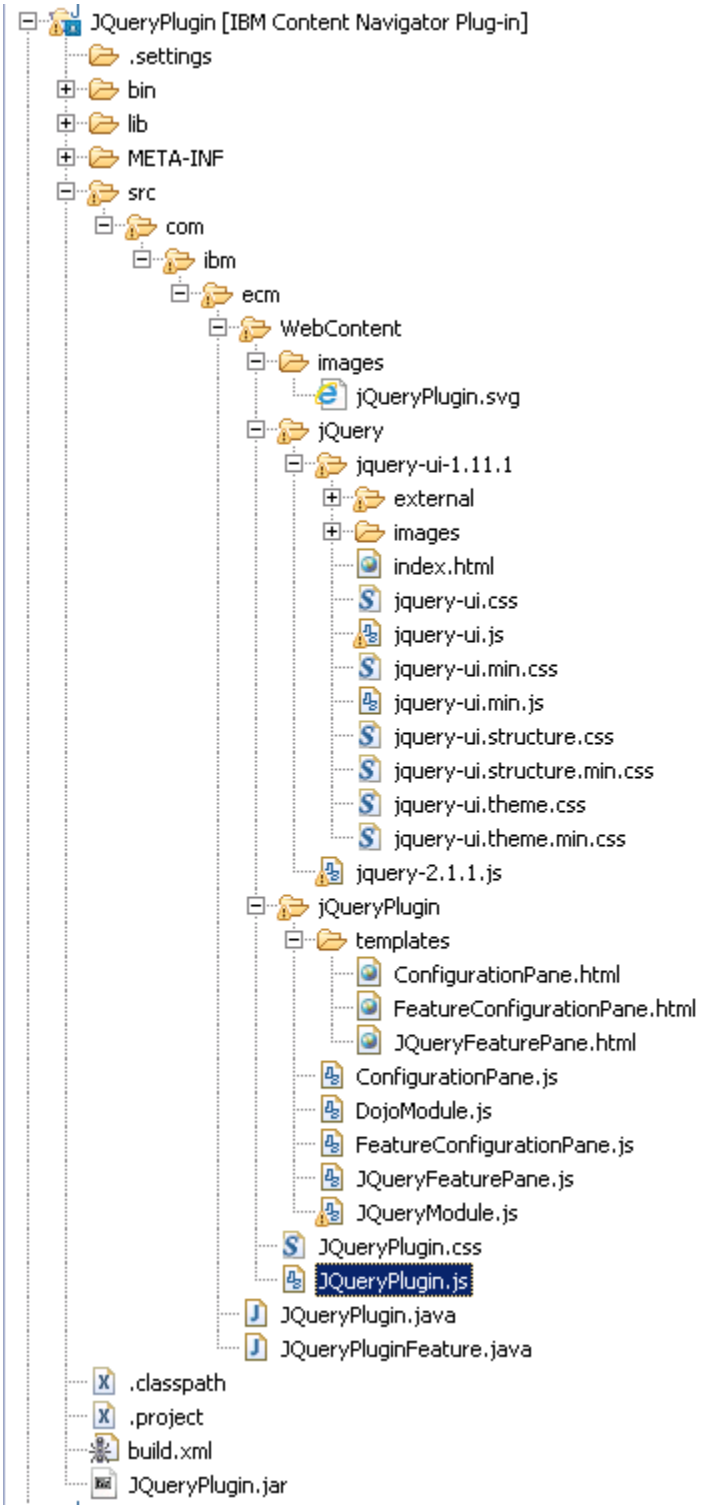
- It is a standalone web page, and not integrated with IBM Content Navigator
- It only utilized the jQuery library, not the jQuery UI widgets

The standalone web page provides a good playground and a good starting point for enabling jQuery in IBM Content Navigator. However, the only supported mechanism for extending IBM Content Navigator is via an IBM Content Navigator plug-in. In this article, we will describe how to leverage jQuery UI assets within an IBM Content Navigator plug-in.

Creating an IBM Content Navigator Plug-in

To create an IBM Content Navigator plug-in, you can use Eclipse along with the IBM Content Navigator Eclipse extension. The eclipse extension will create an IBM Content Navigator plug-in project and provide a build.xml for packaging the plug-in extensions into a JAR file.

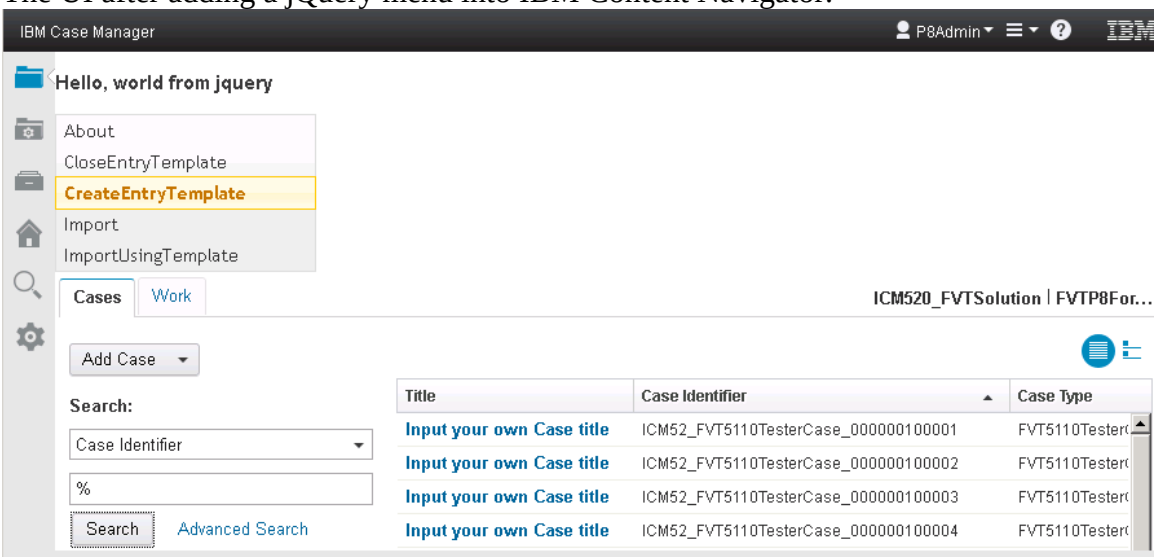
For this article, I used the Eclipse extension provided with the IBM Content Navigator Redbook, to create a new IBM Content Navigator Plug-in project in Eclipse.



We will call our plug-in JQueryPlugin. The corresponding jar file will be `JQueryPlugin.jar`.

The core and the entry point of the plug-in is `JQueryPlugin.js`.

The UI after adding a jQuery menu into IBM Content Navigator.



Here is the code listing for the initial, basic implementation. Later on, we will make it object-oriented, and further on; we will contain the UI modification in an IBM Content Navigator feature.

```
require(["dojo/_base/declare",
        "dojo/_base/connect",
        "dojo/_base/lang",
        "dojo/_base/array",
        "dojo/dom-class",
        "ecm/model/Desktop"
        ],
function(declare, connect, lang, array, domClass, Desktop) {
    connect.connect(Desktop, "onLogin", function() {
        var jQueryVersion = "jquery-2.1.1";
        var jQueryUiVersion = "jquery-ui-1.11.1";
        require({paths: {"jquery":"../plugin/JQueryPlugin/getResource/jQuery/" +
jQueryVersion,
            "jqueryui": "../plugin/JQueryPlugin/getResource/jQuery/" + jQueryUiVersion +
"/jquery-ui"}},
            ["jquery", "jqueryui", "dojo/dom", "dojo/request/xhr"], function(jquery,
jqueryui, dom, xhr) {

            var dojoApi = function() {
                var postData = "{}";
                var url = "/navigator/getDesktop.do";
                xhr(url, {
                    data: postData,
                    method: "POST",
                    handleAs: "text"
                }).then(function(data){
                    data = dojo.fromJson(data.substring(4));
                    console.log("dojo xhr success");
                    //console.log(data);
                    var security_token = data.security_token;
                    if (security_token) {
                        //dom.byId("token").innerHTML = "security token: " +
security_token;
                        jqueryApi(security_token);
                    }
```

```
                    else console.error("no security token available");
                    //else dom.byId("message").innerHTML = "please login to navigator
first";
                }, function(error){
                    console.log(error);
                });
            };

            var jqueryApi = function(security_token) {
                var url = "/navigator/getActions.do";
                var postData =
"action_id=allActions&sorted=true&includeCustomActions=false&security_token=" +
security_token;
                $.ajax({
                    type: "POST",
                    url: url,
                    //dataType: "json",
                    data: { action_id: "allActions", sorted: "true",
                    includeCustomActions: false,
                    security_token: security_token}
                })
                .done(function( msg ) {
                    msg = msg.substring(4); // have to skip the leading {}&& otherwise
jquery error
                    var value = jQuery.parseJSON(msg);
                    if (value.errors) {
                        $("#message").html(value.errors[0].explanation + " : " +
value.errors[0].userResponse);
                        return;
                    }
                    $("#message").html("jquery ajax returns actions: " +
                        //msg);
                        value.actions.length);
                    $("#actions").append("<ul id='" + menuId + "'></ul>");
                    $.each(value.actions, function (i, val) {
                        if (i >= 5) return;
                        $("#" + menuId).append("<li>" + val.id + "</li>");
                    });
                    $("#" + menuId).menu();
                    $("#" + menuId).width("200px");
                })
                .fail(function() {
                    console.log("jquery post failed:\n");
                });
            };


            console.log("jQuery ready");
            $('head').append('<link rel="stylesheet" type="text/css"
href="./plugin/JQueryPlugin/getResource/jQuery/' + jQueryUiVersion + '/jquery-ui.css">');
            var menuId = "menu1";
            $(".launchBarContentArea").prepend(
                '<div id="hello"></div>' +
                '<div id="actions"></div>'
                //'<span><ul id="' + menuId + '" style="display: inline; width:
120px;"><li>menu1</li><li>menu2</li><li>menu3</li></ul>'
            );
            $("#hello").html("<h3>Hello, world from jquery</h3>");
            dojoApi();
        });
    });
});
```

It executes the main logic and performs the setup in the IBM Content Navigator Desktop’s onLogin function.

One thing to point out is that we don't actually need to make a Dojo xhr call to the server to get the security_token. After login the security_token is saved on the static Request

model object, so we can just get it from the Request directly. We leave the implementation as above only to illustrate how to call between jQuery and Dojo.

Packaging of jQuery and jQuery UI

The jQuery components can be setup in a different location on the server, or be included as part of the plug-in. If it's at a different location on the server, it should be copied to a sub-directory under the /navigator application. Including jQuery as part of the plug-in makes it modular and self-contained and you can easily change the versions of jQuery being used.

For the ease of future expansion, we define the version of jQuery and the version of jQuery UI.

For jQuery, here we used "jquery-2.1.1", which is the latest jQuery version to date. For jQuery UI, we used "jquery-ui-1.11.1", which is also the latest jQuery UI version at the present time. The latest jQuery can be downloaded from http://www.jquery.com. The latest jQuery UI can be downloaded from http://ui.jquery.com.

When newer version of jQuery or jQuery UI becomes available, you can update the jQuery versions in the plug-in by copying the latest jQuery file to replace the WebContent/jQuery/jquery-2.1.1.js. After downloading the latest jQuery UI, it can be unzipped to replace the jQuery UI folder in our Plug-in.

After downloading and replacing the latest jQuery and/or jQuery UI, update the JQueryPlugin.js to reflect the version of jQuery that you used.

```
var jQueryVersion = "jquery-2.1.1";
var jQueryUiVersion = "jquery-ui-1.11.1";
```

Loading jQuery and jQuery UI

For loading jQuery, we define the path of "jQuery", which is `"../plugin/JQueryPlugin/getResource/jQuery/" + jQueryVersion`. The ".." is added because the path is relative to Dojo. This results in loading jQuery from the following absolute URL –
http://server:9080/navigator/plugin/JQueryPlugin/getResource/jQuery/jquery-2.1.1.js

Similarly, we load jQuery UI from
`"../plugin/JQueryPlugin/getResource/jQuery/" + jQueryUiVersion + "/jquery-ui"`, which corresponds to the absolute URL of
`http://server:9080/navigator/plugin/JQueryPlugin/getResource/jQuery/jquery-ui-1.11.1/jquery-ui.js`

Then we define two javascript functions:

1. `dojoApi`, which is implemented using Dojo javascript library
2. `jqueryApi`, which is implemented using jQuery API.

Both functions do an Ajax `xhr` call to the Navigator server API. The `dojoApi` function fetches the security token from the server. This is a valid token after user logs in to the IBM Content Navigator server.

The `jQueryApi()` function fetches all the available actions from the IBM Content Navigator server. After fetching the actions, it parses the actions that are returned in JSON format. It then constructs a "<ul>" list consisting of the first 5 IBM Content Navigator actions. Then it creates a jQuery UI menu widget, by calling `$("#" + menuId).menu()`. Finally it calls the jQuery `width()` function to set the width of the menu to 200px.

Loading the CSS resource

There are two CSS resources to be loaded. One is that of the plug-in CSS file, which is setup by the Eclipse extension. We will modify the plug-in CSS file later on for our feature pane. The other CSS resource is the jQuery UI CSS. We load the jQuery UI CSS dynamically by creating an entry in the `<head>` tag of the IBM Content Navigator web page. This is a stylesheet. It's loaded with a link of `"./plugin/jQueryPlugin/getResource/jQuery/' + jQueryUiVersion + "/jquery-ui.css"`, which corresponds to `http://server:9080/navigator/plugin/jQueryPlugin/getResource/jQuery/jquery-ui-1.11.1/jquery-ui.css`

Showing the jQuery UI widget

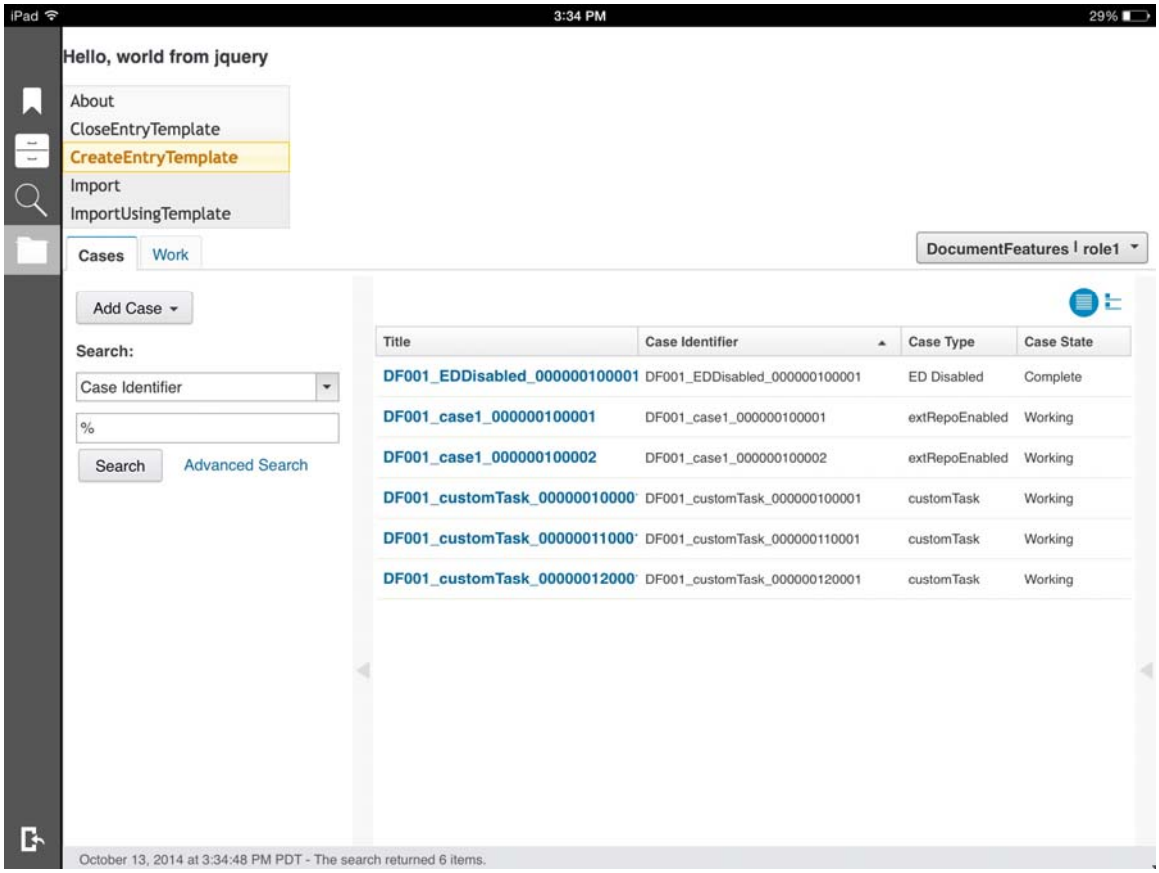
For illustration, we add the jQuery UI widget to the launch bar content area. This corresponds to the element at `$(".launchBarContentArea")`. We use the `prepend()` function to insert anchor elements at the beginning of the launch bar content area.

We then add a hello world message to the beginning of the area, and call the `DojoApi` function, which then call the `jQueryApi` function asynchronously, which then shows the jQuery UI menu widget asynchronously.

Using jQuery for IBM Content Navigator/IBM Case Manager Mobile

IBM Case Manager is a product based on IBM Content Navigator. The mobile application is a hybrid application.

While jQuery has its own mobile packages for developing applications, jQuery can also be used in a hybrid application for developing IBM Content Navigator or IBM Case Manager. The screenshot below shows how the jQuery code works and looks on the iPad, in the IBM Case Manager feature of the Navigator application.



Object-oriented Refactoring

Everything works so far, but Dojo and jQuery are all mixed in a single `JQueryPlugin.js`. This certainly helps to illustrate how Dojo and jQuery are combined, but for enterprise development, we want to have a clean separation of Dojo modules and jQuery modules.

Now we refactor the code to include 3 javascript files:

1. `JQueryPlugin.js`
2. `jQueryPlugin/DojoModule.js`
3. `jQueryPlugin/JQueryModule.js`

After refactoring, the main plug-in file will be simple, and it consists of only a few lines, to include DojoModule, and to instantiate JQueryModule.

```
require(["dojo/_base/declare",
        "dojo/aspect",
        "ecm/model/Desktop",
        "jQueryPlugin/DojoModule",
        "jQueryPlugin/JQueryModule"
            ],
function(declare, aspect, Desktop, DojoModule, JQueryModule) {
      aspect.after(Desktop, "onDesktopLoaded", function() {
```

```
            jQueryPlugin.jQueryModule = new JQueryModule();
            jQueryPlugin.jQueryModule.loadJQuery();
            console.log("Load jQuery with jquery UI version: " +
jQueryPlugin.jQueryModule.jQueryUiVersion);
      });
});
```

The Dojo module

The Dojo module contains logic written in Dojo. Here we include the async function.

```
define(["dojo/_base/declare", "dojo/request/xhr"],
    function(declare, xhr) {

    var DojoModule = declare("jQueryPlugin.DojoModule", [], {

        retrieveSecurityTokenAsync: function(callback) {
            var postData = "{}";
            var url = "/navigator/getDesktop.do";
            xhr(url, {
                data: postData,
                method: "POST",
                handleAs: "text"
            }).then(function(data){
                data = dojo.fromJson(data.substring(4));
                console.log("dojo xhr success");
                //console.log(data);
                var security_token = data.security_token;
                if (security_token) {
                    //dom.byId("token").innerHTML = "security token: "
+ security_token;
                    if (callback) {
                        security_token = callback(security_token);
                    }
                    return security_token;
                }
                else {
                    console.error("no security token available");
                    return null;
                }
                //else dom.byId("message").innerHTML = "please login to
navigator first";
            }, function(error){
                console.log(error);
                return null;
            });
        },

        _EOF: null
    });
    jQueryPlugin.dojoModule = new DojoModule();
    return jQueryPlugin.dojoModule;
});
```

The jQuery Module

In JQueryModule, we encompass all logic in jQuery, after declaring it using Dojo declare.

```
define(["dojo/_base/declare"],
        function(declare) {

    var JQueryModule = declare("jQueryPlugin.JQueryModule", [], {

        jQueryVersion: "jquery-2.1.1",

        jQueryUiVersion:"jquery-ui-1.11.1",

        menuId: "menu1",

        loadJQuery: function(callback) {
            var _this = this;
            require({paths:
{"jquery":"../plugin/JQueryPlugin/getResource/jQuery/" +
this.jQueryVersion,
                "jqueryui":
"../plugin/JQueryPlugin/getResource/jQuery/" + this.jQueryUiVersion +
"/jquery-ui"}},
                ["jquery", "jqueryui", ], function(jquery, jqueryui) {
                    _this._loadJQueryCSS();
                    console.log("jQuery ready");
                    if(callback) {
                        callback();
                    }
                });
        },

        _loadJQueryCSS: function() {
            $('head').append('<link rel="stylesheet" type="text/css"
href="./plugin/JQueryPlugin/getResource/jQuery/' + this.jQueryUiVersion
+ '/jquery-ui.css">');
        },

        execute: function() {
            console.log("assuming jQuery has loaded");
            var jQueryCallback = this.showJQueryMenu;
            this.callDojoFunction(jQueryCallback);
        },

        callDojoFunction: function(jQueryCallback) {
            var dojoModule = jQueryPlugin.dojoModule;
            // note that we avoid using dojo.hitch() in jQuery module
            //return
dojoModule.retrieveSecurityTokenAsync(dojo.hitch(this,
jQueryCallback));
            return
dojoModule.retrieveSecurityTokenAsync(jQueryCallback.bind(this));
        },

        initializeICNUI: function() {
```

```
            $("#hello").html("<h3>Hello, world from jquery</h3>");
            $("#jQueryVersion").html("<h4>jQuery version: " +
this.jQueryVersion + "</h4>");
            $("#jQueryUiVersion").html("<h4>jQuery UI version: " +
this.jQueryUiVersion + "</h4>");
        },

        showJQueryMenu: function(security_token) {
            this.initializeICNUI();
            var url = "/navigator/getActions.do";
            var postData =
"action_id=allActions&sorted=true&includeCustomActions=false&security_t
oken=" + security_token;
            $.ajax({
                type: "POST",
                url: url,
                //dataType: "json",
                data: { action_id: "allActions", sorted: "true",
                includeCustomActions: false,
                security_token: security_token}
            })
            .done(function( msg ) {
                msg = msg.substring(4); // have to skip the leading
{}&& otherwise jquery error
                var value = jQuery.parseJSON(msg);
                if (value.errors) {
                    $("#message").html(value.errors[0].explanation + "
: " + value.errors[0].userResponse);
                    return;
                }
                $("#message").html("jquery ajax returns actions: " +
                    //msg);
                    value.actions.length);
                var menuId = this.menuId;
                $("#actions").append("<ul id='" + menuId + "'></ul>");
                $.each(value.actions, function (i, val) {
                    if (i >= 5) return;
                    $("#" + menuId).append("<li>" + val.id + "</li>");
                });
                $("#" + menuId).menu();
                $("#" + menuId).width("200px");
            })
            .fail(function() {
                console.log("jquery post failed:\n");
            });
            return security_token;
        },

        EOF: null
    });
    return JQueryModule;
});
```

Calling jQuery API from Dojo

This example demonstrates calling jQuery API from Dojo in `JQueryPlugin.js`. After instantiating the JQueryModule, we can access its members and functions.

```
            jQueryPlugin.jQueryModule = new JQueryModule();
            jQueryPlugin.jQueryModule.loadJQuery();
            console.log("Load jQuery with jquery UI version: " +
jQueryPlugin.jQueryModule.jQueryUiVersion);
```

Calling Dojo API from jQuery

This example also demonstrates calling Dojo API from jQuery in `JQueryModule.js`.

```
    callDojoFunction: function(jQueryCallback) {
        var dojoModule = jQueryPlugin.dojoModule;
        //return dojoModule.retrieveSecurityTokenAsync(dojo.hitch(this, jQueryCallback));
        return dojoModule.retrieveSecurityTokenAsync(jQueryCallback.bind(this));
    },
```

Here we avoided using the dojo `hitch` function by using the generic `bind` function.

Creating an IBM Content Navigator Feature plug-in

A proper way to extend IBM Content Navigator would be to create an IBM Content Navigator feature for the third-party javascript library. We can confine all the functionality of the third-party library within this feature pane. While the main pane is dojo-driven, this new feature pane can be jQuery-driven.

There are several steps involved when creating an IBM Content Navigator feature.

- Create a new Java class `JQueryPluginFeature.java`
- Return a PluginFeature array in the main JQueryPlugin.java
  ```
  public PluginFeature[] getFeatures() {
        return new PluginFeature[] { new JQueryPluginFeature()};
  }
  ```
- Create a feature icon for the feature under `WebContent/images/`. Feature icons should be created as a scalable vector graphics file according to the IBM Content

Navigator documentation: http://www-01.ibm.com/support/knowledgecenter/SSEUEX_2.0.3/com.ibm.developingeuc.doc/eucso014.htm

- Create a Feature Pane in javascript, `jqueryPlugin/JQueryFeaturePane.js`, along with the associated template file as `jQueryPlugin/templates/JQueryFeaturePane.html`
- Create a Feature configuration pane, `FeatureConfigurationPane.js`, along with the associated template file as `jQueryPlugin/templates/FeatureConfigurationPane.html`
- Define the style sheet for the FeaturePane, in `WebContent/jQueryPlugin/JQueryPlugin.css`

Feature Pane – JQueryFeaturePane.js

The feature pane needs to extend `_LaunchBarPane`, to adhere to IBM Content Navigator plugin feature requirement.

Note that we execute the jQuery UI feature and present the jQuery menu in the `postCreate` function of the Feature Pane. For jQuery developers, `postCreate` is a builtin function of dijit widget class. In the life cycle of a dijit, `postCreate` is executed just before the widget is shown on the UI.

Another possibility is to execute the jQuery UI feature in the `startup` function, after calling the super class's function by `this.inherited(arguments)`.

```
define([
    "dojo/_base/declare",
    "ecm/widget/layout/_LaunchBarPane",
    "dojo/text!./templates/JQueryFeaturePane.html"
],

function(declare,
        _LaunchBarPane,
        template) {

    return declare("jQueryPlugin.JQueryFeaturePane", [
        _LaunchBarPane
    ], {

        templateString: template,
        widgetsInTemplate: true,

        postCreate: function() {
            this.logEntry("postCreate");
            this.inherited(arguments);

            //jQueryPlugin.jQueryModule was initialized in
JQueryPlugin.js
            jQueryPlugin.jQueryModule.execute();
```

```
            console.log("in postCreate of JQueryFeaturePane with jquery
UI version: " + jQueryPlugin.jQueryModule.jQueryUiVersion);

            this.logExit("postCreate");
        },

    });
});
```

The associated template file for the Feature Pane

jQueryPlugin/templates/JQueryFeaturePane.html

```
<div class="ecmjQueryPane">
      <div id="hello"></div>
      <div id="jQueryVersion"></div>
      <div id="jQueryUiVersion"></div>
      <div id="message"></div>
      <div id="actions"></div>
</div>
```

During the execute function of the jQuery module, the `<div>`'s can be accessed in jQuery via `$("#id")` or `$(".class")`.

The corresponding css style definition for the Feature Pane

The css style defines padding for the feature pane so the information would be presented with proper spacing. Note that we cannot simply define the rule as `.ecmjQueryPane`, otherwise the rule would be overwritten by a more specific system style rule.

It also defines the feature icon.

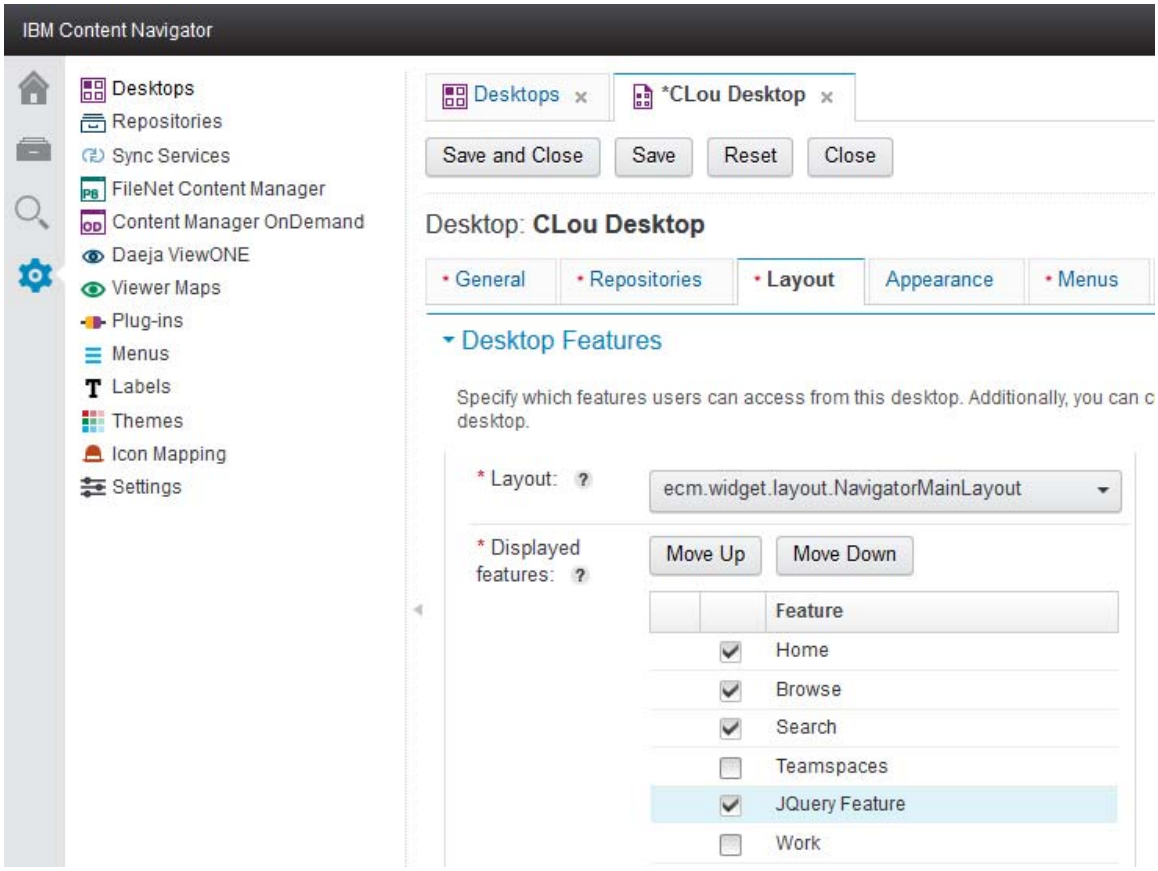
```
.oneui.ecm .ecmjQueryPane.dijitContentPane  {
      padding: 10px;
}

.jQueryFeatureLaunchIcon {
      width: 32px;
      height: 32px;
      background-image: url('images/jQueryPlugin.svg');
      background-repeat: no-repeat;
      background-position: -16px -16px;
}
```

Note: We used the SVG file provided with the IBM Content Navigator SamplePlugin.

Enabling the new feature

After installing the jQueryPlugin, in order to show the icon on the left hand panel of the navigator, remember to add the new jQuery feature to the corresponding IBM Content Navigator or IBM Content Manager desktop from the Layout tab of the admin desktop.



Conclusion

The jQuery plug-in loads both jQuery and jQuery UI, as well as jQuery UI CSS. Through a combination of Dojo and jQuery calls, it retrieves the available actions in IBM Content Navigator, and displays the top 5 actions, using the menu widget from jQuery UI.

IBM Content Navigator or IBM Case Manager plug-ins can be developed using either Dojo, or jQuery, or another Javascript library, or a mixture of Dojo and jQuery.

What we did in Part II:

- Packaged the functionality in an IBM Content Navigator feature plug-in
- Separated Dojo and jQuery modules
- Utilized a jQuery UI widget in IBM Content Navigator
- Packaged jQuery inside the IBM Content Navigator plug-in, so that no separate setup is needed
- Made it easier to upgrade to latest jQuery and jQuery UI
- Load jQuery library during the initialization stage of the jQuery plug-in
- Execute the jQuery UI feature when the feature icon is activated

The plug-in demonstrates that jQuery can be safely mixed with Dojo. jQuery can co-exist with Dojo, and jQuery UI widgets can be used in or to extend IBM Content Navigator or IBM Case Manager widget. The plug-in is attached. jQuery library and jQuery UI library

do not need to be downloaded separately , they have already been included in the JQueryPlugin.jar. No separate setup is needed on IBM Content Navigator server. The plug-in works on the Navigator mobile app in the ICM feature as well.

Reference

1. Extending and Customizing IBM Content Navigator, IBM Redbook
http://www.redbooks.ibm.com/redbooks/pdfs/sg248055.pdf