Using JQuery and AngularJS to Customize the Web Transfer Client in EFT v7.2 and later

THE INFORMATION IN THIS ARTICLE APPLIES TO:

- EFT v7.2 - 8.0.4.x
- jQuery was removed in 8.0.5.7
- EFT 8.3.0.x

DISCUSSION

The Web Transfer Client in EFT version 7.2 and later can be customized if you are familiar with with HTML, Javascript, jQuery, and AngularJS.

Be sure to back up any files that you plan to edit.

The **customization.js** file provides access for custom code. Opening the file for the first time should bring your attention to this function:

```
/**
* Any javascript customization code should be placed inside this
* function to allow for easy upgrading. Allows
* full access to JQuery and certain AngularJS functions and contexts.
* If customization is enabled via the
* adminConfig, this code will be the last thing run on the loaded
```

```
$('li').each(function () {
$(this).attr('style', 'background-color:#ffffcc;');
});
};
```

EFT 8.3.0 now supports CSP Level 3, as a result certain changes will need to be done to avoid failures with customization:

Switch from using .setAttribute(...) to direction manipulation and instead use .style.visibility

JQuery

The short comment states that this function has full access to JQuery. Indeed it does, the two examples listed are examples using JQuery manipulation.

```
context.customFunction = function () {
// Example: hide the navigation tree
$('.left-sidebar').hide();
// Example: edit style of li elements without modifying the css files
$('li').each(function () {
$(this).attr('style', 'background-color:#ffffcc;');
});
};
```

Anything listed in the existing JQuery documentation is available in this function. The JQuery documentation can be found here: http://api.jquery.com/. WTC-specific CSS or ID selectors for jQuery's use can be found using any browser's inspection tool. This tool can be accessed by right clicking on the element and selecting "Inspect Element," or the browser's specific hotkeys.

AngularJS

AngularJS is not applicable to thew WTC in EFT v8.0.5.8 or later.

The comment details that any potential developer also has access to AngularJS functions. It is even possible to write entire Angular controllers or directives in this file, if that is desired. In customization.js, the code might look something like this:

```
(function(context) {

/**
* Any javascript customization code should be placed inside this

* function to allow for easy upgrading. Allows
* full access to JQuery and certain AngularJS functions and

* contexts.

* If customization is enabled via the
* adminConfig, this code will be the last thing run on the loaded

* page.
* CONTACT SUPPORT FOR MORE INFORMATION.
*/
context.customFunction = function () {
// Other code…
};

gsb.angular.AddCtrl = gsb.angular.AddCtrl || {};
angular.module('jument').controller('AddCtrl', AddController);
AddController.$inject = ['$scope'];
function AddController($scope){
$scope.x = 0;
$scope.y = 0;
```

```
$scope.sum = 0;

$scope.add = function(x, y){

$scope.sum = x+y

}

}
```

}(gsb.customization));

Note that this code lies outside the **customFunction()**. This is something unique for building components such as controllers and directives. All other loose and unassociated component code should remain inside the customFunction() for proper execution.

Directives and controllers require an HTML component. The WTC's HTML files can be found in **jument/ng-views** folder. The files can be edited to include any new directive written into the customization.js file, or they can have their own new standalone. The following example for the controller above can be standalone (a new file in the **jument/ng-views** folder) or integrated into another existing file:

```
<div ng-controller="AddCtrl">

<input type="number" ng-model="x"> +

<input type="number" ng-model="y"> = {{ sum }}

<p><button ng-click="add(x, y)">Add</button></p>

</div>
```

Using these code examples will create a simple addition component. Depending on what kind of component is written, files may need to be imported into index.html to make sure they're included in the project. This is a task left up to the developer; there is an easy to follow established convention for file imports. For more information on AngularJS controllers, directives, or any other angular functionality, please see the AngularJS documentation here: https://docs.angularjs.org/api, or the developer guide: https://docs.angularjs.org/guide. Any style or error handling concessions are left to the developer.

Contexts

The brief comment also mentions the idea of contexts. In the earlier example we use

contexts to define the configuration file:

```
gsb.customization = gsb.customization || {};
```

as well as the controller we're constructing:

```
gsb.angular.AddCtrl = gsb.angular.AddCtrl || {};
```

The 'gsb' is a global variable that we use to share contexts across the program. The 'gsb.angular' context contains all of the controllers and directives that make up WTC as a whole. Globalscape will not provide a list of these components, even though they are accessible from the customization file. The code in question is highly obfuscated to any auxillary developer and will likely change upon upgrade. If any modifications are done to those contexts the developer does so at his or her own risk. It is recommended to add your own contexts on top of the gsb.angular or even gsb contexts, and maintain any customizations separately from the main WTC project, instead of modifying those provided by GSB.

To that end, the customizations.js file also has full access to the gsb.config context, which contains any configuration information for WTC. Located in the adminConfig.js file, it is possible to add configuration information as necessary and be able to consume those variables in the customizations.js file.