Oracle CPQ JET Configuration and Transaction UI: Refactoring Existing JavaScript Customizations

In this Document

REVISION HISTORY:

Initial Document published	10/15/2018
Updated for 18C Library upgrade	01/10/2019
Updated for 19C Library upgrade	11/07/2019
Updated for 20B Library upgrade	05/05/2020

APPLIES TO:

Oracle CPQ Cloud Update 20B and later
Information in this document applies to any platform.

PURPOSE:

The purpose of this document is to detail how to refactor any custom JavaScript to be used within the new JET UI.

SCOPE:

CPQ Cloud Administrators with knowledge of JavaScript.

DETAILS:

While CPQ does not endorse or warranty the use of JavaScript customizations, we recognize that some customers have extended the CPQ platform to address gaps in product functionality. Recent and upcoming releases are adding platform support for these requirements in order to eliminate the need for JavaScript customizations. To support critical use cases, Oracle CPQ Cloud Release 19C introduces a JavaScript API ("CPQJS"), which includes methods for accessing attributes, actions, and other elements on the JET Configuration and JET Transaction UIs (also known as JET Responsive UIs).

Many of the most commonly customized features are now available in CPQ Cloud and no longer require custom JavaScript. Sticky headers, column freezing, responsiveness, and filtering of line items are now supported by the CPQ JET UI. Hiding of Actions and Transaction Arrays (in Release 18D JET Transaction UI) are also now available. Customers should remove all JS associated with these features when implementing the new JET Transaction and Configuration UI. In upcoming releases additional features that previously required JS customization will be available with platform-support.

The new JET UI first introduced in Release 18C supports customization of the UI using JS and CSS, but with some differences in structure and approach. Customers with JavaScript customizations to their transaction UI may experience conflicts when the new JET UI is enabled due to these differences. Customizations can be refactored to address these differences. However, customers should consider carefully the relative benefits of JavaScript customizations in light of the associated risks. Customizations may conflict with new CPQ platform features, data may be corrupted or lost, maintenance and support may be difficult, cross-browser support must be verified, performance may be impaired, and testing is required for each upgrade.

This document explains the key structural differences vs. CPQ's Legacy UI, and reviews the new methods provided in Release 19C that enable JavaScript to access elements of the JET UIs.

For details on styling of JET UIs using CSS, see CPQ Cloud JET CSS: Branding and Styling (Doc ID 2462711.1).

Differences between Legacy and JET UI JavaScript Customization

In CPQ's Legacy UI, JavaScript customizations of Commerce UIs are commonly implemented in several ways:

- Modifications to the CPQ JavaScript Framework files (bm-framework.js, commerce.js, config.js)
- Inclusion of JavaScript in an HTML attribute
- JavaScript added to the Header or Footer

Modifications to CPQ JavaScript Framework files will require refactoring because these files are no longer loaded in the JET UI.

Previous customizations to the CPQ JavaScript Framework will need to be re-implemented in either an HTML attribute or in the site Header/Footer. With JET single, combined and minified files (transactionMain.js and configMain.js) are loaded which can be inspected using web development tools such as Firebug and Chrome Developer tools, but ***cannot be directly modified and should not be referenced*** in custom JavaScript.

Inclusion of JavaScript in an HTML attribute or addition to the site Footer will continue to work with the JET UI, but may require syntactic revision. JavaScript in the site Header will work only under certain circumstances, as detailed below. For this reason, inclusion in an HTML attribute or in the site Footer will provide a more reliable approach.

When implementing JavaScript in an HTML attribute, best practice is to store the script in File Manager and reference this in the HTML attribute directly or in a default BML function for the HTML attribute. This simplifies discovery and search for JavaScript implemented on a site.

JET UI Architecture

The HTML and DOM (Document Object Model) for the JET UI differ from the Legacy Commerce and Configuration UIs. JET also differs from the Legacy UI in the JS functions or methods available. JavaScript customizations may need to be refactored to reflect these differences. Developers working on JET UI customizations should familiarize themselves with the differences between the Legacy and JET UI DOM structures, including access to hidden attributes not previously accessible in the JET DOM.

Rather than IDs, the JET UI DOM uses variable names for references, and all components are prefaced by ‘oj'.

The JET UI architecture in CPQ Cloud uses a client-side data model, REST to communicate with the server, and JavaScript to render UI components. Manipulating data or calling actions via the UI components can introduce data inconsistency and timing problems that could break functionality. Instead, the CPQJS API introduced in CPQ Release 19C provides direct access to the data and methods.

For improved performance, UI components are only rendered when the user opens the containing panel or tab. However, values can be set or retrieved even for non-rendered attributes using the CPQJS API.

Additionally, UI components are rendered progressively after the DOM Ready event has been fired. Because of this, scripting may need to be called after elements are rendered, or deferred using setTimeout.

Using the CPQJS API

Administrators may access these methods using custom JavaScript contained in a separate file or in a read-only HTML attribute. Reference the methods with the namespace "CPQJS" as in the examples below.

```
CPQJS.setAttributeVal('myAttributeVarName','New Value');
```

```
CPQJS.onTableLoaded('lineItemGrid', setRowColors);
```

Important Considerations:

- Be aware of timing. To run scripts after the layout is fully loaded, call scripts from an HTML attribute that appears on the layout. This must be on a panel/tab that is visible by default.
- Use setTimeout to defer a script until current threads are completed.
- Avoid overuse of JavaScript, as it can affect performance.
- Enlist an experienced JavaScript developer to implement customizations on the JET UIs.

Methods: JET Configuration

Method	Arguments	Notes
attributeExists	varName	Returns true if attribute exists
actionExists	varName	Returns true if action exists
isConfig		Returns true if Config page
isJet		Returns true if JET page
setAttributeVal	varName, value, [index]	Sets the specified attribute value
getAttributeVal	varName, index	Gets the specified attribute value
onAttributeChange	varName, handler	Calls a function on attribute change (use function name without parentheses) Handler will receive an object with: • value: the current value • varName: the varName of the changed attribute • index: the array index, if applicable
performAction	varName	Calls an action
onActionComplete	varName, handler	Calls a function when an action completes (use function name without parentheses) Handler will receive an object with: • varName: action varname passed to this call • actionName: name of REST action, possibly the same unless corrected from legacy • status: [success|fail]
openPopup	content, title	Opens a popup with specified content and title

Methods: JET Transaction

Method	Arguments	Notes
attributeExists	varName	Returns true if attribute exists
actionExists	varName	Returns true if action exists
isCommerce		Returns true if Commerce page
isJet		Returns true if JET page
setAttributeVal	varName, value, [index]	Sets the specified attribute value
getAttributeVal	varName, [index]	Gets the specified attribute value
onAttributeChange	varName, handler	Calls a function on attribute change (use function name without parentheses) Handler will receive an object with: • value: the current value, • varName: the varName of the changed attribute • index: the array index, if applicable

Method	Arguments	Notes
`onTableLoaded`	`varName, handler`	Calls a function when the specified table is loaded (use 'lineItemGrid' for the transaction LIG). Handler receives an object with {varName} This will be triggered each time the table is loaded, which can happen repeatedly (during filtering, for example). Ensure functions are lightweight and idempotent (can safely run repeatedly).

`getTableInfo`	`varName`	Returns the following info:
		• totalSize
		• pageSize
		• startIndex
		• endIndex
		• selection
		  ○ keys
		  ○ indexes
		Example:
		`CPQJS.getTableInfo('lineItemGrid')`
		Returns an object containing the listed values, including an array of row keys and indexes of selected rows
		Note:
		Beginning in 20B, the selection syntax for the Line Item Grid has changed (arrays stay the same).
		• selectAll - is added for LIG
		• keys - represents all selections across pages, or all non-selections if selectAll is true
		• indexes - represents all selected indexes on the current page (regardless of selectAll)
		Customers using getTableInfo should re-test the functionality, and may need to refactor their custom JavaScript.

`performAction`	`varName`	Calls an action
`onActionComplete`	`varName, handler`	Calls a function when an action completes (use function name without parentheses) Handler will receive an object with: • varName: action varname passed to this call • actionName: name of REST action, possibly the same unless corrected from legacy • status: [success|fail]
`openPopup`	`content, title`	Opens a popup with specified content and title
`tableExists`	`id`	Returns true if table exists

Using setTimeout

In an example where custom embedded charts or analytics are implemented in JET UI with JavaScript as part of an HTML attribute, the analytic may depend on the values of other attributes. A ***setTimeout*** may be needed to ensure the values are loaded before processing values to generate the HTML attribute.

The following JavaScript can be added to an existing script to delay running the function by a specified amount of time (1 millisecond in the example below).

```
<script>
    setTimeout(function(){
        REPLACE WITH PRE-EXISTING SCRIPT
    },1);
</script>
```

Custom JavaScript in the CPQ Header and Footer

JavaScript or custom HTML may be inserted into the HEAD, Header HTML or Footer HTML via the CPQ Admin 'Header & Footer' page. The latter two are also referred to as 'Custom Header' and 'Custom Footer'. However, depending on both the UI (JET versus Legacy) and the Navigation (ALTA versus Legacy) these customizations may not work, as shown below. Insertion in the Custom Footer is the recommended approach as these customizations will be applied in all circumstances.

UI	Navigation	HTML Head	Header HTML	Footer HTML
Legacy UI	Legacy Navigation	Included	Included	Included
Legacy UI	Alta Navigation	Included	Not included	Included
JET UI	Legacy Navigation	Not included	Included	Included
JET UI	Alta Navigation	Not included	Not included	Included

Enable Right-to-Left (RTL) for the JET Transaction or Configuration Layouts

To enable right-to-left (RTL) for the JET transaction or configuration layouts, customers can add this script to the page via the header or footer. This will work when the user's language is set to Hebrew or Arabic.

```
<script type="text/javascript">

//if user lang is Hebrew or Arabic, set HTML dir to RTL if not already set

if (document.getElementsByTagName("HTML")[0].getAttribute("dir")==null)

{ var _BM_LAYOUT_DIR = ( _BM_USER_LANGUAGE == "iw_IL" || _BM_USER_LANGUAGE == "ar_SA" )
? "rtl":"ltr"; document.getElementsByTagName("HTML")[0].setAttribute("dir",
_BM_LAYOUT_DIR); }

</script>
```

Coexistence of JET with Legacy Configuration or Transaction UI

Especially during development and testing, customers may wish to display JET Configuration or Transaction UI to some users while displaying the Legacy UI to other users. Conditional logic may need to be added to existing JavaScript to render each UI properly.

Where Can I Learn More?

- CPQ Cloud Administration Online Help
- CPQ Cloud JET CSS: Branding and Styling (Doc ID 2462711.1)
- CPQ Cloud JET Transaction UI: Planning for Adoption (Doc ID 2455080.1)

References

- CPQ Cloud JET CSS: Branding and Styling (Doc ID 2462711.1)
- Oracle CPQ Cloud Release Documentation (Doc ID 1674718.1)
- CPQ Cloud JET Transaction UI: Planning for Adoption (Doc ID 2455080.1)