Part 8 - Calling JavaScript webresources from the Command Bar

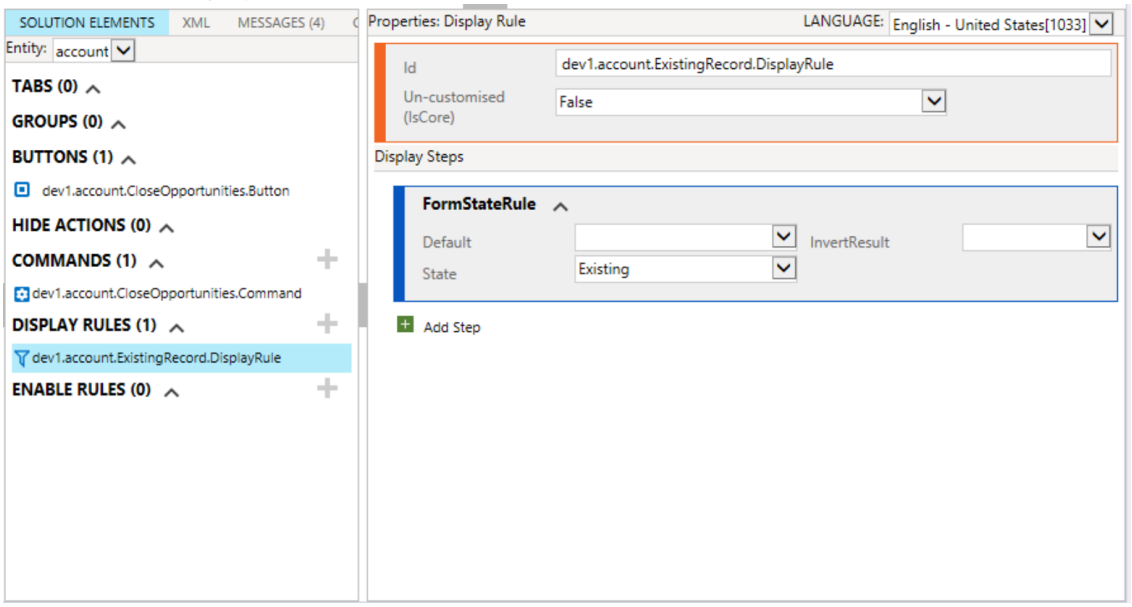
This is part of the course 'Scott's guide to building Power Apps JavaScript Web Resources using TypeScript'.

In this eighth part we will cover how to create a button on the Ribbon to call the JavaScript webresource that we have created. In a later part of this series we will convert this to call a Custom API.

Add Command Bar Button using the Ribbon Workbench

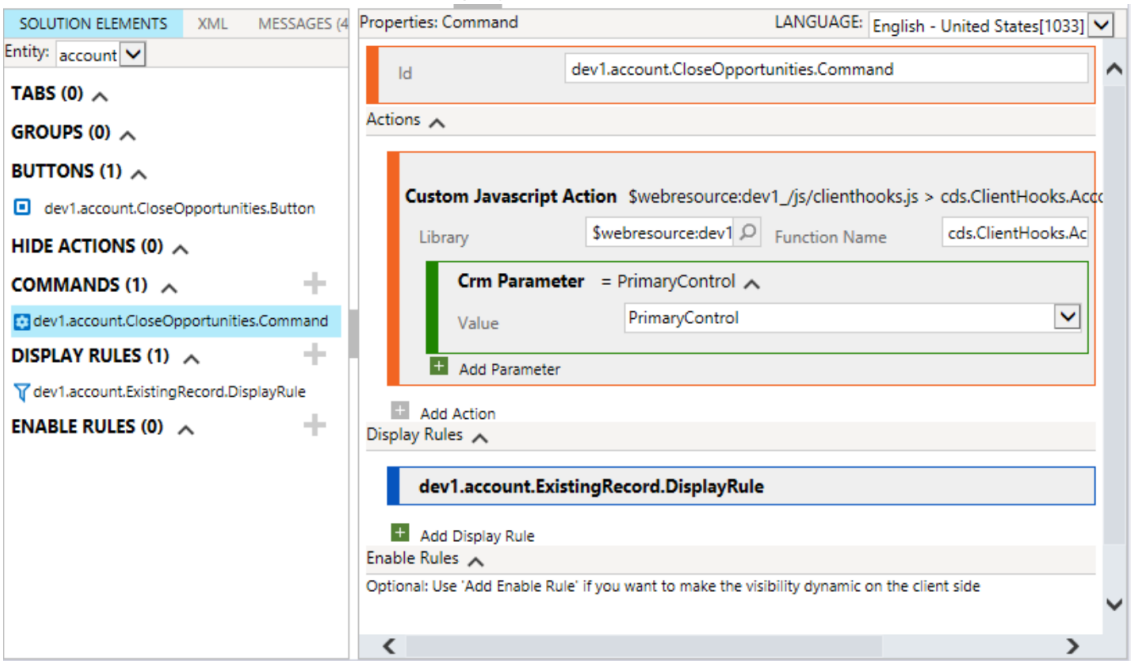
Before we can test our code inside the Model Driven App, we must configure the Command Bar to have a button that calls our AccountRibbon code.

1. Open the Ribbon Workbench (see https://ribbonworkbench.uservoice.com/knowledgebase/articles/71374-1-getting-started-with-the-ribbon-workbench)
2. Load a solution that contains just the account table/entity - see https://ribbonworkbench.uservoice.com/knowledgebase/articles/169819-speed-up-publishing-of-solutions
3. Create a new Display Rule:



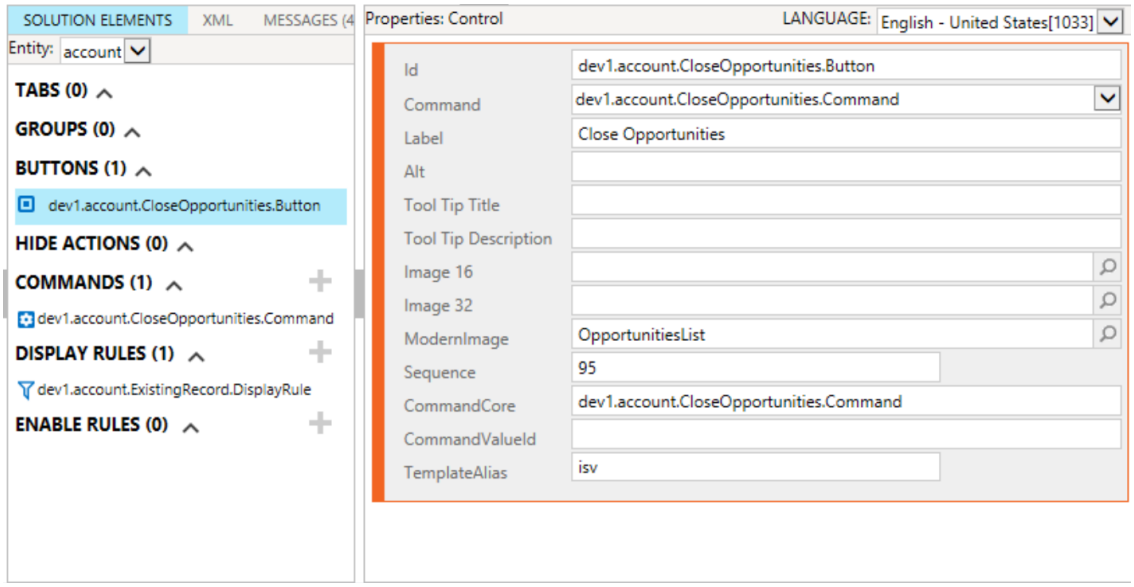
4. Add a new Step to the Enable Rule of type `FormStateRule`. Set the State Property to `Existing`. This will be used to ensure our button is only visible on existing records.

5. Add a new Command and add the Display Rule:



6. To the Command add a Custom JavaScript Action
7. Set the Command Properties as follows:
   - **Library:** `\$webresource:dev1_/js/clienthooks.js`
   - **Function Name:** `cds.ClientHooks.AccountRibbon.closeOpporunities`

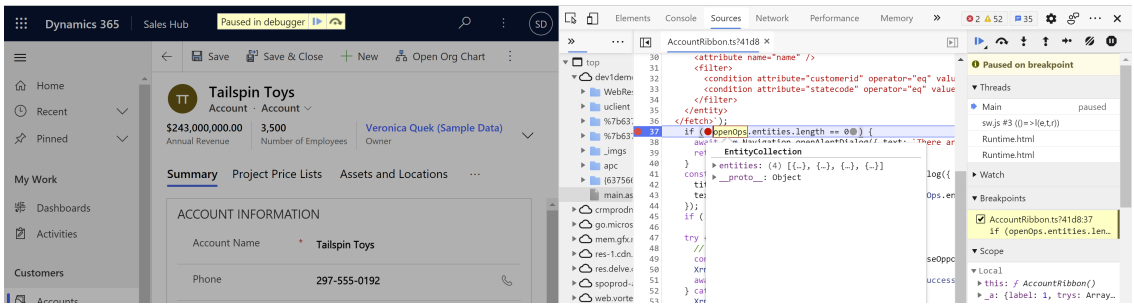
     The name of the Function is very important that it matches the name you have exported. The first part will be the namespace you set in the `webpack.common.js` `library` property, and the second part will be the name of the class you exported followed by the static method.
8. Add a `Crm Parameter` of Type `PrimaryControl` - this is so that the form context is passed to `closeOpporunities(context: Xrm.FormContext)`
9. Drag a new button on to the Account Form Command Bar and set the Command to be the one you just created. Give the button a name such as 'Close Opportunities' and possibly a `ModernImage`.



10. Publish your changes in the Ribbon Workbench.

Testing in the Browser

1. Build your VSCode project using `npm start` and ensure Fiddler is running with auto-responders turned on as we did in Part 4.
2. Press F12 to open the Developer Tools (or use `Ctrl+Shift+I`)
3. Open an Account record that has some open Opportunities. You should see your new Ribbon Button.
   Note: Using integration tests has a distinct advantage over in-browser testing since you can automatically create records that are needed to execute the tests rather than doing it manually each time.
4. In the Developer Tools, search for the file `webpack://AccountRibbon.ts` and place a breakpoint in the `closeOpportunitiesInternal` method.
5. Press the Command Bar button and you should see your code hitting the breakpoint and be able to step through and debug.
6. If you make any changes in VSCode you simply need to refresh the record and you will get the re-built version because of the Fiddler auto-responder.



Troubleshooting

setMetadataCache not called

If you get the following error 'Error: Metadata not found for opportunity. Please create the early bound types.' This usually means you have forgotten to call `setMetadataCache(metadataCache);`

Missing await

If you receive 'UnhandledPromiseRejectionWarning: Error: Caught error after test environment was torn down' or 'Jest did not exit one second after the test run has completed.' – it usually means that you have called an `async` function in your test that returns a `Promise`, but you have not used `await`.

Next Up

Now that we've created some fairly complex JavaScript logic - we will look at moving some of this logic to the server via a Custom API - and then invoking that Custom API using TypeScript and `dataverse-ify` early-bound types.