Microsoft Dynamics D365 Retail Experts - https://www.meetup.com/Dynamics-User-Group-India/
Sheikhmydheen
https://www.linkedin.com/in/sheikhmydheen/
Sheikhmydheen@gmail.com , d365retail@gmail.com

Add custom control to POS transaction grid:

This topic explains how to add new POS custom control in the POS transaction page using the screen layout designer. This topic is applicable for Dynamics 365 for Finance and Operations or Dynamics 365 for Retail platform update 8 with retail App update 4 hotfix.

You can add more information to the Retail POS transaction page by using custom controls. Custom control can be added to the transaction page by using the screen layout designer, from the screen layout designer you can drag and drop the custom control and define your own location, height and width by adjusting the custom control in the designer. Business logic for the custom control can be implemented in your own extensions using the POS extension framework.

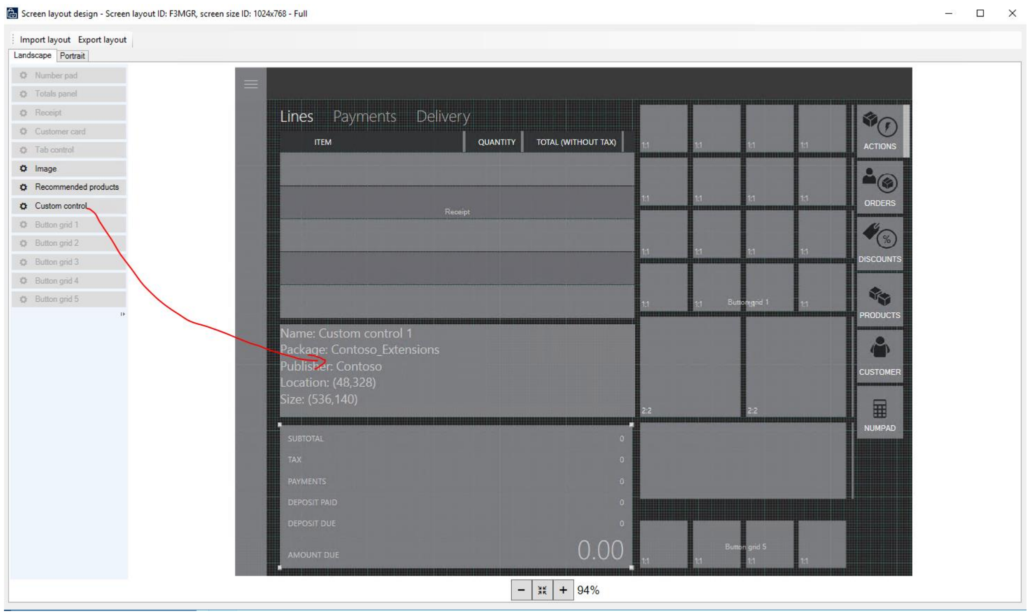
Scenario/business problem

Let's add custom control in POS transaction to show the selected line item details item id and description.

Lab actions

Add new custom control:

1. Login to Dynamics 365 for Retail.
2. Navigate to Retail > Channel setup > POS setup > POS > Screen layouts
3. Select the F3MGR screen layout ID and click the Designer button in the action bar.
4. Select the 1440x960 – Full layout from the layout sizes and click the Layout designer button.
5. If prompted click Open and follow the instruction to install the designer tool.
6. After installing it will ask for AAD credentials, provide the details to launch the designer.
7. In the designer drag the custom control to the transaction page as shown in the picture.



Name: Custom control 1
Package: Contoso_Extensions
Publisher: Contoso
Location: (48,328)
Size: (536,140)

8. Right click on the custom control in the transaction page and click customize
9. In the custom control window set the Control Name, Package name and Publisher name

   Control Name: `lineDetails`
   Package Name: `Pos_Extensibility_Samples`
   Publisher Name: `Contoso`

   Note: These names should match the names in the extension manifest.

10. Click the X button in the designer to close the designer.
11. When prompted to Save changes, click Yes. If you click No the changes will not be saved.
12. Navigate to Retail > Retail IT > Distribution schedule
13. Select the Registers (1090) job and click Run now.

Add business logic to custom control:

14. Open visual studio 2015 in administrator mode.
15. Open ModernPOS solution from …\RetailSDK\POS
16. Under the POS.Extensions project create a new folder called CustomControlExtensions.
17. Under CustomControlExtensions, create new folder called Cart.
18. In the Cart folder, add a new ts (typescript) and name it has CartViewController.ts
19. Add the below import statement to import the relevant entities and context.

```
import { ProxyEntities } from "PosApi/Entities";
import { IExtensionCartViewControllerContext } from
"PosApi/Extend/Views/CartView";
import * as CartView from "PosApi/Extend/Views/CartView";
```

20. Create a new class called CartViewController and extend it from CartExtensionViewControllerBase. We are extending from CartExtensionViewControllerBase class to get the cart tender lines or cart line selected or cleared handlers, so that we can show the selected line in our custom control.

```
export default class CartViewController extends
CartView.CartExtensionViewControllerBase {

}
```

21. Inside the class add two private variables to get the selected cart line and tender lines.

```
private _selectedCartLines: ProxyEntities.CartLine[];
private _selectedTenderLines: ProxyEntities.TenderLine[];
```

22. Create a class constructor method to set the cart and tender selection information.

```
constructor(context: IExtensionCartViewControllerContext) {
        super(context);
```

```
            this.cartLineSelectedHandler = (data: CartView.CartLineSelectedData): void
        => {
                this._selectedCartLines = data.cartLines;
            };

            this.cartLineSelectionClearedHandler = (): void => {
                this._selectedCartLines = undefined;
            };

            this.tenderLineSelectedHandler = (data: CartView.TenderLineSelectedData):
        void => {
                this._selectedTenderLines = data.tenderLines;
            };

            this.tenderLineSelectionClearedHandler = (): void => {
                this._selectedCartLines = undefined;
            };
                      }
```

The overall class should look like below:

```
        /**
         * SAMPLE CODE NOTICE
         *
         * THIS SAMPLE CODE IS MADE AVAILABLE AS IS.  MICROSOFT MAKES NO WARRANTIES,
        WHETHER EXPRESS OR IMPLIED,
         * OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY OR COMPLETENESS OF RESPONSES,
        OF RESULTS, OR CONDITIONS OF MERCHANTABILITY.
         * THE ENTIRE RISK OF THE USE OR THE RESULTS FROM THE USE OF THIS SAMPLE CODE
        REMAINS WITH THE USER.
         * NO TECHNICAL SUPPORT IS PROVIDED.  YOU MAY NOT DISTRIBUTE THIS CODE UNLESS YOU
        HAVE A LICENSE AGREEMENT WITH MICROSOFT THAT ALLOWS YOU TO DO SO.
         */

        import { ProxyEntities } from "PosApi/Entities";
        import { IExtensionCartViewControllerContext } from
        "PosApi/Extend/Views/CartView";
        import * as CartView from "PosApi/Extend/Views/CartView";

        export default class CartViewController extends
        CartView.CartExtensionViewControllerBase {

            private _selectedCartLines: ProxyEntities.CartLine[];
            private _selectedTenderLines: ProxyEntities.TenderLine[];

            /**
             * Creates a new instance of the CartViewController class.
             * @param {IExtensionCartViewControllerContext} context The events Handler
        context.
             * @remarks The events handler context contains APIs through which a handler
        can communicate with POS.
             */
            constructor(context: IExtensionCartViewControllerContext) {
                super(context);
                this.cartLineSelectedHandler = (data: CartView.CartLineSelectedData): void
        => {
                    this._selectedCartLines = data.cartLines;
                };

                this.cartLineSelectionClearedHandler = (): void => {
                    this._selectedCartLines = undefined;
                };

                this.tenderLineSelectedHandler = (data: CartView.TenderLineSelectedData):
        void => {
                    this._selectedTenderLines = data.tenderLines;
                };

                this.tenderLineSelectionClearedHandler = (): void => {
                    this._selectedCartLines = undefined;
                };
            }
        }
```

23. Inside the Cart folder, create a new html file and name it as LineDetailsCustomControl.html
24. In the html file we will add two text fields tow show the selected line item and description.

    Delete the default code and copy paste the below sample code:

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <!-- Note: The element id is different than the id generated by the POS
extensibility framework. This 'template' id is not used by the POS extensibility
framework. -->
    <script id="Microsot_Pos_Extensibility_Samples_LineDetails" type="text/html">
        <!-- ko ifnot: isCartLineSelected -->
        <h4>No cart lines selected</h4>
        <!-- /ko -->
        <!-- ko if: isCartLineSelected -->
        <h4 data-bind="text: cartLineItemId">Item ID</h4>
        <h4 data-bind="text: cartLineDescription">Item ID</h4>
        <!-- /ko -->
    </script>
</body>
       </html>
```

25. Inside the Cart folder, create a new ts file and name it as LineDetailsCustomControl.ts
26. In the ts file we will add the logic to bind the line details information.

27. Import the POS entities and Type extensions to use the reference type in the constructor and other events.

```
import {
    ObjectExtensions,
    StringExtensions,
    ArrayExtensions
} from "PosApi/TypeExtensions";


       import { ProxyEntities } from "PosApi/Entities";
```

28. Create a new class and extend it from CartViewCustomControlBase.

```
export default class LineDetailsCustomControl extends CartViewCustomControlBase {}
```

29. Declare the below private variables to set the cart item id and description.

```
private static readonly TEMPLATE_ID: string =
"Microsot_Pos_Extensibility_Samples_LineDetails";

public readonly cartLineItemId: Computed<string>;
public readonly cartLineDescription: Computed<string>;
public readonly isCartLineSelected: Computed<boolean>;
private readonly _cartLine: Observable<ProxyEntities.CartLine>;
private _state: ICartViewCustomControlState;
```

30. Create the constructor method to initialize the and get the selected handler.

```
constructor(id: string, context: ICartViewCustomControlContext) {
        super(id, context);

        this._cartLine = ko.observable(null);
        this.cartLineItemId = ko.computed(() => {
            let cartLine: ProxyEntities.CartLine = this._cartLine();
            if (!ObjectExtensions.isNullOrUndefined(cartLine)) {
                return cartLine.ItemId;
            }

            return StringExtensions.EMPTY;
        });

        this.cartLineDescription = ko.computed(() => {
            let cartLine: ProxyEntities.CartLine = this._cartLine();
            if (!ObjectExtensions.isNullOrUndefined(cartLine)) {
                return cartLine.Description;
            }

            return StringExtensions.EMPTY;
        });
```

```
            this.isCartLineSelected = ko.computed(() =>
    !ObjectExtensions.isNullOrUndefined(this._cartLine()));

            this.cartLineSelectedHandler = (data: CartLineSelectedData) => {
                if (ArrayExtensions.hasElements(data.cartLines)) {
                    this._cartLine(data.cartLines[0]);
                }
            };

            this.cartLineSelectionClearedHandler = () => {
                this._cartLine(null);
            };
               }
```

31. Add the onReady method to bind the control to the specified html element.

```
     public onReady(element: HTMLElement): void {
         ko.applyBindingsToNode(element, {
             template: {
                 name: LineDetailsCustomControl.TEMPLATE_ID,
                 data: this
             }
         });
            }
```

32. Add the init method to set the state.

```
    public init(state: ICartViewCustomControlState): void {
     this._state = state;
        }
```

The overall class should look like below:

```
          /**
     * SAMPLE CODE NOTICE
     *
     * THIS SAMPLE CODE IS MADE AVAILABLE AS IS.  MICROSOFT MAKES NO WARRANTIES,
    WHETHER EXPRESS OR IMPLIED,
     * OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY OR COMPLETENESS OF RESPONSES,
    OF RESULTS, OR CONDITIONS OF MERCHANTABILITY.
     * THE ENTIRE RISK OF THE USE OR THE RESULTS FROM THE USE OF THIS SAMPLE CODE
    REMAINS WITH THE USER.
     * NO TECHNICAL SUPPORT IS PROVIDED.  YOU MAY NOT DISTRIBUTE THIS CODE UNLESS YOU
    HAVE A LICENSE AGREEMENT WITH MICROSOFT THAT ALLOWS YOU TO DO SO.
     */

    import {
        CartViewCustomControlBase,
        ICartViewCustomControlState,
        ICartViewCustomControlContext,
        CartLineSelectedData
    } from "PosApi/Extend/Views/CartView";
```

Microsoft Dynamics D365 Retail Experts - https://www.meetup.com/Dynamics-User-Group-India/
Sheikhmydheen
https://www.linkedin.com/in/sheikhmydheen/
Sheikhmydheen@gmail.com , d365retail@gmail.com

```
import {
    ObjectExtensions,
    StringExtensions,
    ArrayExtensions
} from "PosApi/TypeExtensions";


import { ProxyEntities } from "PosApi/Entities";


export default class LineDetailsCustomControl extends CartViewCustomControlBase {

    private static readonly TEMPLATE_ID: string =
"Microsot_Pos_Extensibility_Samples_LineDetails";

    public readonly cartLineItemId: Computed<string>;
    public readonly cartLineDescription: Computed<string>;
    public readonly isCartLineSelected: Computed<boolean>;

    private readonly _cartLine: Observable<ProxyEntities.CartLine>;
    private _state: ICartViewCustomControlState;



    constructor(id: string, context: ICartViewCustomControlContext) {
        super(id, context);

        this._cartLine = ko.observable(null);
        this.cartLineItemId = ko.computed(() => {
            let cartLine: ProxyEntities.CartLine = this._cartLine();
            if (!ObjectExtensions.isNullOrUndefined(cartLine)) {
                return cartLine.ItemId;
            }

            return StringExtensions.EMPTY;
        });

        this.cartLineDescription = ko.computed(() => {
            let cartLine: ProxyEntities.CartLine = this._cartLine();
            if (!ObjectExtensions.isNullOrUndefined(cartLine)) {
                return cartLine.Description;
            }

            return StringExtensions.EMPTY;
        });

        this.isCartLineSelected = ko.computed(() =>
!ObjectExtensions.isNullOrUndefined(this._cartLine()));

        this.cartLineSelectedHandler = (data: CartLineSelectedData) => {
            if (ArrayExtensions.hasElements(data.cartLines)) {
                this._cartLine(data.cartLines[0]);
            }
        };
```

```
            this.cartLineSelectionClearedHandler = () => {
                this._cartLine(null);
            };
        }

        /**
         * Binds the control to the specified element.
         * @param {HTMLElement} element The element to which the control should be
    bound.
         */
        public onReady(element: HTMLElement): void {
            ko.applyBindingsToNode(element, {
                template: {
                    name: LineDetailsCustomControl.TEMPLATE_ID,
                    data: this
                }
            });
        }

        /**
         * Initializes the control.
         * @param {ICartViewCustomControlState} state The initial state of the page
    used to initialize the control.
         */
        public init(state: ICartViewCustomControlState): void {
            this._state = state;
        }
    }
```

33. Create a new json file and under the CustomControlExtensions folder and name it as manifest.json.
34. In the manifest.json file, copy and paste the below code:

```
{
  "$schema": "../manifestSchema.json",
  "name": "Pos_Extensibility_Samples",
  "publisher": "Contoso",
  "version": "7.2.0",
  "minimumPosVersion": "7.2.0.0",
  "components": {
    "extend": {
      "views": {
        "CartView": {
          "viewController": { "modulePath": "Cart/CartViewController" },
          "controlsConfig": {
            "customControls": [
              {
                "controlName": "lineDetails",
                "htmlPath": "Cart/LineDetailsCustomControl.html",
                "modulePath": "Cart/LineDetailsCustomControl"
              }
            ]
          }
        }
```

```
            }
        }

    }
          }
```

35. Open the extensions.json file under POS.Extensions project and update it with CustomControlExtensions samples, so that POS during runtime will include this extension.

```
{
  "extensionPackages": [
    {
      "baseUrl": "SampleExtensions2"
    },
    {
      "baseUrl": "POSAPIExtension"
    },
    {
      "baseUrl": "CustomColumnExtensions"
    },
    {
      "baseUrl": "EODSample"
    },
    {
      "baseUrl": "ProdDetailsCustomColumnExtensions"
    },
    {
      "baseUrl": "SerachExtension"
    },
    {
      "baseUrl": "SuspendReceiptSample"
    },
    {
      "baseUrl": "CoinDispenserSample"
    },
    {
      "baseUrl": "StoreHoursSample"
    },
    {
      "baseUrl": "CustomControlExtensions"
    }
  ]
        }
```

36. Open the tsconfig.json to comment out the extension package folders from the exclude list. POS will use this file to include or exclude the extension. By default, the list contains all the excluded extensions list, if you want to include any extension part of the POS then you need add the extension folder name and comment the extension from the extension list like below.

```
"exclude": [
"AuditEventExtensionSample",
"B2BSample",
"CustomerSearchWithAttributesSample",
"FiscalRegisterSample",
 "PaymentSample",
 "PromotionsSample",
"SalesTransactionSignatureSample",
"SampleExtensions",
//"SampleExtensions2",
//"StoreHoursSample",
"SuspendTransactionReceiptSample"
//"POSAPIExtension",
//"CustomColumnExtensions",
//"EODSample",
//"ProdDetailsCustomColumnExtensions",
//"SerachExtension",
//"SuspendReceiptSample",
//"CoinDispenserSample",
//"CustomControlExtensions"
  ],
```

37. Compile and rebuild the project.

Validate the customization:

38. Login to MPOS using 000160 as operator id and 123 as password.
39. Click the current transaction button on the welcome screen
40. Add any item (0005) to transaction and select the line item added.
41. The custom should display the selected line item id and description.