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

How to extend POS views:

This topic explains how you can extend existing POS views like Customer Add edit screen etc. 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.

Few POS views (Transaction screen and Welcome) can be extended using the screen layout designer and all the other views can be extended using the Retail SDK. In this document we will focus on how to extend existing POS views using Retail SDK.

In general, below are the extension points/patterns supported by POS views:

1. **Custom App Bar Buttons** - Adding custom buttons to the App Bar on select pages
2. **Custom Column Sets** – The ability to replace the grid columns with a custom column set on select pages
3. **Custom control** – Add new control to the selected pages.

List of POS views supported currently for extensions:

Note: We are adding support for more extension points in others views in the upcoming releases and hotfix.

POS Views	Custom control	Custom columns	Custom App bar buttons
Cart view (Screen layout based)	Yes	Yes	No
CustomerAddEditView	Yes		Yes
CustomerDetailsView	Yes		Yes
SearchView		Yes	Yes
InventoryLookupView		Yes	Yes
ShowJournalView		Yes	Yes
SimpleProductDetailsView		Yes	Yes
AddressAddEditView	Yes		
PaymentView			Yes
PriceCheckView	Yes		
SearchOrdersView		Yes	

Scenario/business problem

Let’s add custom column and App bar in POS customer search screen.

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

Lab actions

Add new custom column and App bar button:

1. Open visual studio 2015 in administrator mode.
2. Open ModernPOS solution from ...\RetailSDK\POS
3. Under the POS.Extensions project create a new folder called SearchExtension.
4. Under SearchExtension, create new folder called ViewExtensions.
5. Under ViewExtensions, create new folder called Search.
6. In the Search folder, add a new ts (typescript) file and name it has CustomCustomerSearchColumns.ts
7. Add the below import statement to import the relevant entities and context.

```
import { ICustomerSearchColumn } from "PosApi/Extend/Views/SearchView";
import { ICustomColumnsContext } from "PosApi/Extend/Views/CustomListColumns";
import { ProxyEntities } from "PosApi/Entities";
```

8. Next step we will add the existing column plus the custom column to ts file.

```
export default (context: ICustomColumnsContext): ICustomerSearchColumn[] => {
    return [
        {
            title: context.resources.getString("string_2"),
            computeValue: (row: ProxyEntities.GlobalCustomer): string => { return
row.AccountNumber; },
            ratio: 15,
            collapseOrder: 5,
            minWidth: 120
        }, {
            title: context.resources.getString("string_3"),
            computeValue: (row: ProxyEntities.GlobalCustomer): string => { return
row.FullName; },
            ratio: 20,
            collapseOrder: 4,
            minWidth: 200
        }, {
            title: context.resources.getString("string_4"),
            computeValue: (row: ProxyEntities.GlobalCustomer): string => { return
row.FullAddress; },
            ratio: 25,
            collapseOrder: 1,
            minWidth: 200
        }, {
            title: context.resources.getString("string_5"),
            computeValue: (row: ProxyEntities.GlobalCustomer): string => { return
row.Email; },
            ratio: 20,
            collapseOrder: 2,
            minWidth: 200
        }, {
            title: context.resources.getString("string_7"),
            computeValue: (row: ProxyEntities.GlobalCustomer): string => { return
row.Phone; },
            ratio: 20,
            collapseOrder: 3,
            minWidth: 120
        }
    ];
       };
```

9. Let's add the resource file for the column name localization: Create a new folder called Resources under SearchExtension.
10. Inside Resources folder add new folder called Strings and inside that create one more folder called en-US
11. Create a new file inside called resources.resjson file and copy paste the below code:

```
//=================================================================================
=======================
//======================================= Sample comment.
==============================================
//=================================================================================
=======================
{
    //======================== Sample View extensions strings.
========================
   "string_0"                                        : "Quick compare products",
    "_string_0.comment"                               : "Product search page app
bar command label.",

    "string_1"                                        : "View customer summary",
    "_string_1.comment"                               : "Customer search page app
bar command label.",

    //======================== Column names.  ========================
    "string_2"                                        : "ACCOUNT
NUMBER_CUSTOMIZED",
    "_string_2.comment"                               : "Customer search column
name.",

    "string_3"                                        : "NAME",
    "_string_3.comment"                               : "Customer search column
name.",

    "string_4"                                        : "ADDRESS",
    "_string_4.comment"                               : "Customer search column
name.",

    "string_5"                                        : "CONTACT EMAIL",
    "_string_5.comment"                               : "Customer search column
name.",

    "string_7"                                        : "PHONE NUMBER",
    "_string_7.comment"                               : "Customer search column
name."

}
```

12. In the SearchExtension folder add a new folder called DialogSample
13. In the DialogSample folder, add a new ts (typescript) file and name it has MessageDialog.ts
14. Add the below import statement to import the relevant entities and context.

```
import { ShowMessageDialogClientRequest, ShowMessageDialogClientResponse,
IMessageDialogOptions } from "PosApi/Consume/Dialogs";
import { IExtensionContext } from "PosApi/Framework/ExtensionContext";
import { ClientEntities } from "PosApi/Entities";
```

15. Create a new class called `MessageDialog`
    `export default class MessageDialog {}`
16. Inside the class add a new show method like below:

```
 public static show(context: IExtensionContext, message: string): Promise<void> {
        let promise: Promise<void> = new Promise<void>((resolve: () => void,
reject: (reason?: any) => void) => {
            let messageDialogOptions: IMessageDialogOptions = {
                title: "Extension Message Dialog",
                message: message,
                showCloseX: true, // this property will return "Close" as result
when "X" is clicked to close dialog.
                button1: {
                    id: "Button1Close",
                    label: "OK",
                    result: "OKResult"
                },
                button2: {
                    id: "Button2Cancel",
                    label: "Cancel",
                    result: "CancelResult"
                }
            };

            let dialogRequest:
ShowMessageDialogClientRequest<ShowMessageDialogClientResponse> =
                new
ShowMessageDialogClientRequest<ShowMessageDialogClientResponse>(messageDialogOptio
ns);

            context.runtime.executeAsync(dialogRequest).then((result:
ClientEntities.ICancelableDataResult<ShowMessageDialogClientResponse>) => {
                if (!result.canceled) {
                    context.logger.logInformational("MessageDialog result: " +
result.data.result.dialogResult);
                    resolve();
                }
            }).catch((reason: any) => {
                context.logger.logError(JSON.stringify(reason));
                reject(reason);
            });
        });
```

```
            return promise;
                }
```

Let's add custom app bar button in search view to show a dialog with selected customer info:

17. In the ViewExtensions folder, add a new ts (typescript) file and name it has ViewCustomerSummaryCommand.ts
18. Add the below import statement to import the relevant entities and context.

```
import { ProxyEntities } from "PosApi/Entities";
import { ArrayExtensions, ObjectExtensions } from "PosApi/TypeExtensions";
import { IExtensionCommandContext } from "PosApi/Extend/Views/AppBarCommands";
import * as SearchView from "PosApi/Extend/Views/SearchView";
import MessageDialog from "../DialogSample/MessageDialog";
```

19. Create a new class called ViewCustomerSummaryCommand and extend it from CustomerSearchExtensionCommandBase

```
export default class ViewCustomerSummaryCommand extends
SearchView.CustomerSearchExtensionCommandBase {}
```

20. Inside the class declare private variable to capture the selected customer search results

```
private _customerSearchResults: ProxyEntities.GlobalCustomer[];
```

21. Add the class constructor method to initialize and clear the search handler:

```
constructor(context:
IExtensionCommandContext<SearchView.ICustomerSearchToExtensionCommandMessageTypeMa
p>) {
        super(context);

        this.id = "viewCustomerSummaryCommand";
        this.label = context.resources.getString("string_1");
        this.extraClass = "iconLightningBolt";
        this._customerSearchResults = [];

        this.searchResultsSelectedHandler = (data:
SearchView.CustomerSearchSearchResultSelectedData): void => {
            this._customerSearchResults = data.customers;
            this.canExecute = true;
        };

        this.searchResultSelectionClearedHandler = (): void => {
            this._customerSearchResults = [];
            this.canExecute = false;
        };
            }
```

22. Add the init method to initialize the visible property:

```
protected init(state: SearchView.ICustomerSearchExtensionCommandState): void {
      this.isVisible = true;
   }
```

23. Add the execute method to handle the app button click handler, we will read the selected customer data from the handler and show it in a simple dialog.

```
protected execute(): void {
let customer: ProxyEntities.GlobalCustomer =
ArrayExtensions.firstOrUndefined(this._customerSearchResults);
 if (!ObjectExtensions.isNullOrUndefined(customer)) {
           let message: string = "Customer Account: " + (customer.AccountNumber
|| "") + "  |  ";
     message += "Name: " + customer.FullName + "  |  ";
     message += "Phone Number: " + customer.Phone + "  |  ";
     message += "Email Address: " + customer.Email;

     MessageDialog.show(this.context, message);
 }     }
```

The overall code 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 { ArrayExtensions, ObjectExtensions } from "PosApi/TypeExtensions";
import { IExtensionCommandContext } from "PosApi/Extend/Views/AppBarCommands";
import * as SearchView from "PosApi/Extend/Views/SearchView";
import MessageDialog from "../../DialogSample/MessageDialog";

export default class ViewCustomerSummaryCommand extends
SearchView.CustomerSearchExtensionCommandBase {

    private _customerSearchResults: ProxyEntities.GlobalCustomer[];

    /**
     * Creates a new instance of the ViewCustomerSummaryCommand class.
     * @param
{IExtensionCommandContext<CustomerDetailsView.ICustomerSearchToExtensionCommandMes
sageTypeMap>} context The command context.
     * @remarks The command context contains APIs through which a command can
communicate with POS.
     */
    constructor(context:
IExtensionCommandContext<SearchView.ICustomerSearchToExtensionCommandMessageTypeMa
p>) {
        super(context);
```

```
            this.id = "viewCustomerSummaryCommand";
            this.label = context.resources.getString("string_1");
            this.extraClass = "iconLightningBolt";
            this._customerSearchResults = [];

            this.searchResultsSelectedHandler = (data:
SearchView.CustomerSearchSearchResultSelectedData): void => {
                this._customerSearchResults = data.customers;
                this.canExecute = true;
            };

            this.searchResultSelectionClearedHandler = (): void => {
                this._customerSearchResults = [];
                this.canExecute = false;
            };
        }

        /**
         * Initializes the command.
         * @param {CustomerDetailsView.ICustomerDetailsExtensionCommandState} state
The state used to initialize the command.
         */
        protected init(state: SearchView.ICustomerSearchExtensionCommandState): void {
            this.isVisible = true;
        }

        /**
         * Executes the command.
         */
        protected execute(): void {
            let customer: ProxyEntities.GlobalCustomer =
ArrayExtensions.firstOrUndefined(this._customerSearchResults);
            if (!ObjectExtensions.isNullOrUndefined(customer)) {
                let message: string = "Customer Account: " + (customer.AccountNumber
|| "") + "  |  ";
                message += "Name: " + customer.FullName + "  |  ";
                message += "Phone Number: " + customer.Phone + "  |  ";
                message += "Email Address: " + customer.Email;

                MessageDialog.show(this.context, message);
            }
        }
}
```

24. Create a new json file and under the SearchExtension folder and name it as manifest.json.
25. In the manifest.json file, copy and paste the below code:

```
{
  "$schema": "../manifestSchema.json",
  "name": "Pos_Extensibility_Samples",
  "publisher": "Microsoft",
  "version": "7.2.0",
  "minimumPosVersion": "7.2.0.0",
  "components": {
    "resources": {
      "supportedUICultures": [ "en-US" ],
      "fallbackUICulture": "en-US",
      "culturesDirectoryPath": "Resources/Strings ",
      "stringResourcesFileName": "resources.resjson",
      "cultureInfoOverridesFilePath": "Resources/cultureInfoOverrides.json"
    },
    "extend": {
      "views": {
        "SearchView": {
          "customerAppBarCommands": [ { "modulePath":
"ViewExtensions/Search/ViewCustomerSummaryCommand" } ],
          "customerListConfiguration": { "modulePath":
"ViewExtensions/Search/CustomCustomerSearchColumns" }
        }
      }
    }
  }
 }
```

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

```
{
  "extensionPackages": [
    {
      "baseUrl": "SampleExtensions2"
    },
    {
      "baseUrl": "POSAPIExtension"
    },
    {
      "baseUrl": "CustomColumnExtensions"
    },
    {
      "baseUrl": "EODSample"
    },
    {
      "baseUrl": "ProdDetailsCustomColumnExtensions"
    },
    {
      "baseUrl": "SearchExtension"
    }
  ]
      }
```

27. 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",
       //"SearchExtension"
            ],
```

28. Compile and rebuild the project.

Validate the customization:

29. Login to MPOS using 000160 as operator id and 123 as password.
30. Search for customer 2001 using the search bar on the top.
31. You should see the custom columns added.
32. Click the new app bar button added after selecting a customer. It should open up a dialog with the selected customer details.