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

POS Payment extension

We added new extension points in POS to support payment extensibility, you will implement the core payment logic in the payment device or payment connector using the Hardware station APIs but for scenario where you want to pass some additional information as extension properties to your connector/device or you want to show some custom messages in between the payment flow or you need some input from cashier to complete the flow and send some intermediate response back to the payment device/connector Ex: Customer ID validation status, voice authorization then you can override POS payment request to do this additional logics. This topic is applicable for Dynamics 365 for Finance and Operations or Dynamics 365 for Retail platform update 8 with retail App update 3 hotfix.

Below are the request/response you can override from the POS side to customize the payment flow for the above scenarios:

- PaymentTerminalAuthorizePaymentRequestHandler
- PaymentTerminalCapturePaymentRequestHandler
- PaymentTerminalExecuteTaskRequestHandler
- PaymentTerminalRefundPaymentRequestHandler
- PaymentTerminalVoidPaymentRequestHandler

The POS runtime checks the extension manifest and see if there is any extension for these requests and if there some extension found then it loads the extended request and execute the overridden request. In the extension project, you can override these requests and add your own implementation to call the custom payment providers and update the response based on status returned by your providers. When you override these request, you are overriding only the core logic (calling the methods in Hardware station and process the response per your own logic to show some custom messages or calling the Hardware station again by passing some additional information or sending some intermediate response back to Hardware station like voice authorization code, customer ID validation status etc.). After all your custom logic is done send the updated response you received from Hardware station (Payment device/connector) to POS, you no need to worry about how we add/void/decline the payment line and conclude the transaction based on the response, all the standard workflow is taken care by the POS.

Scenario: Lets override the authorize payment request in POS and pass some additional properties and validate whether we can receive those custom properties in Hardware station. For the lab we will only override authorize request.

PaymentTerminalAuthorizePaymentRequestHandler:

Authorize request is core payment request from POS which initiates and authorize the card payment request, you can override this request if you want to change anything in the authorize workflow. To override the request, you need to extend the PaymentTerminalAuthorizePaymentRequestHandlerin POS:

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 PaymentExtension.
4. Under PaymentExtension, create new folder called Handlers.
5. In the Handlers folder, add a new ts (typescript) file and name it has PaymentHandlerHelper.ts

6. Add the below import statement to import the relevant entities.

```
import { ClientEntities, ProxyEntities } from "PosApi/Entities";
import { ObjectExtensions, StringExtensions } from "PosApi/TypeExtensions";
```

7. Create a new class called `PaymentHandlerHelper`:

```
export class PaymentHandlerHelper {}
```

8. Add public static method to return the extension properties:

```
public static FillExtensionProperties(
    cart: ProxyEntities.Cart,
              extensionProperties: ClientEntities.IExtensionTransaction):
      ClientEntities.IExtensionTransaction {

        let extraProperties: ClientEntities.IExtensionTransaction = null;
    // Build extra extension properties.
    if (!ObjectExtensions.isNullOrUndefined(cart)) {
        extraProperties = {
            ExtensionProperties: [
                <ProxyEntities.CommerceProperty>{
                    Key: "CartId",
                    Value: <ProxyEntities.CommercePropertyValue>{
                                   StringValue:
      !ObjectExtensions.isNullOrUndefined(cart) ? cart.Id : ""
                    }
                }, {
                    Key: "ChannelId",
                    Value: <ProxyEntities.CommercePropertyValue>{
                                   StringValue:
      (ObjectExtensions.isNullOrUndefined(cart.ChannelId)) ? "" :
      cart.ChannelId.toString()
                           }
                       }, {
                           Key: "TerminalId",
                           Value: <ProxyEntities.CommercePropertyValue>{
                               StringValue: cart.TerminalId
                           }
                   }, {
                           Key: "StaffId",
                           Value: <ProxyEntities.CommercePropertyValue>{
   StringValue: cart.StaffId }
                       }, {
                           Key: "CustomerId",
                           Value: <ProxyEntities.CommercePropertyValue>{
                               StringValue:
   (StringExtensions.isNullOrWhitespace(cart.CustomerId)) ? "" : cart.CustomerId
                           }
                       }, {
                           Key: "ShippingZipCode",
                           Value: <ProxyEntities.CommercePropertyValue>{
                               StringValue:
   !ObjectExtensions.isNullOrUndefined(cart.ShippingAddress) ?
```

```
          (StringExtensions.isNullOrWhitespace(cart.ShippingAddress.ZipCode) ? "" :
          cart.ShippingAddress.ZipCode) : ""
                                  }
                              }
                          ]
                  };
              };

              if (ObjectExtensions.isNullOrUndefined(extensionProperties)) {
                  extensionProperties = extraProperties;
              } else {
                  for (let i: number = 0; i <
      extraProperties.ExtensionProperties.length; i++) {

      extensionProperties.ExtensionProperties.push(extraProperties.ExtensionProperties[i
      ]);
                  }
              }

              return extensionProperties;
          }
```

9. In the Handlers folder, add a new ts (typescript) file and name it has PaymentTerminalAuthorizePaymentRequestHandlerExt.ts
10. Add the below import statement to import the relevant entities.

```
import { PaymentTerminalAuthorizePaymentRequestHandler } from
"PosApi/Extend/RequestHandlers/PeripheralsRequestHandlers";
import { PaymentTerminalAuthorizePaymentRequest,
PaymentTerminalAuthorizePaymentResponse } from "PosApi/Consume/Peripherals";
import { ClientEntities, ProxyEntities } from "PosApi/Entities";
import { GetCurrentCartClientRequest, GetCurrentCartClientResponse } from
"PosApi/Consume/Cart";
import { PaymentHandlerHelper } from "./PaymentHandlerHelper";
import { ObjectExtensions } from "PosApi/TypeExtensions";
```

11. Create a new class called `PaymentTerminalAuthorizePaymentRequestHandlerExt which extends from PaymentTerminalAuthorizePaymentRequestHandler`:

```
export default class PaymentTerminalAuthorizePaymentRequestHandlerExt extends
PaymentTerminalAuthorizePaymentRequestHandler {}
```

12. Add the executeAsync method to execute the request with custom logic:

```
 public executeAsync(request:
PaymentTerminalAuthorizePaymentRequest<PaymentTerminalAuthorizePaymentResponse>):

Promise<ClientEntities.ICancelableDataResult<PaymentTerminalAuthorizePaymentRespon
se>> {
        let cart: ProxyEntities.Cart = null;
        let cartRequest: GetCurrentCartClientRequest<GetCurrentCartClientResponse>
= new GetCurrentCartClientRequest();

        // Get cart first and then build extension properties based on cart info.
        return this.context.runtime.executeAsync(cartRequest)
```

```
            .then((result: 
ClientEntities.ICancelableDataResult<GetCurrentCartClientResponse>): void => {
                if (!(result.canceled ||
ObjectExtensions.isNullOrUndefined(result.data))) {
                    cart = result.data.result;
                }
            }).then((): 
Promise<ClientEntities.ICancelableDataResult<PaymentTerminalAuthorizePaymentRespon
se>> => {
                let newRequest: 
PaymentTerminalAuthorizePaymentRequest<PaymentTerminalAuthorizePaymentResponse> =
                    new 
PaymentTerminalAuthorizePaymentRequest<PaymentTerminalAuthorizePaymentResponse>(
                        request.paymentConnectorId,
                        request.amount,
                        request.tenderInfo,
                        request.voiceAuthorization,
                        request.isManualEntry,
                        PaymentHandlerHelper.FillExtensionProperties(cart, 
request.extensionTransactionProperties));

                return this.defaultExecuteAsync(newRequest);
            });
           }
```

13. Create a new json file and under the PaymentExtension folder and name it as manifest.json.
14. In the manifest.json file, copy and paste the below code:

```
{
  "$schema": "../manifestSchema.json",
  "name": "Pos_Payment_Samples",
  "publisher": "Microsoft",
  "version": "7.2.0",
  "minimumPosVersion": "7.2.0.0",
  "components": {
    "extend": {
      "requestHandlers": [
        {
          "modulePath": 
"Handlers/PaymentTerminalAuthorizePaymentRequestHandlerExt"
        }
      ]
    }
  }
       }
```

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

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

16. 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"
    //"PaymentExtension"
         ],
```

17. Compile and rebuild the project.

Build Deploy the sample payment project:

18. Open visual studio 2015 in administrator mode.
19. Open Extension.PaymentSample.proj from …\RetailSDK\SampleExtensions\HardwareStation\Extension.PaymentSample
20. Right click and rebuild the project.

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

21. Navigate to …\ RetailSDK\SampleExtensions\HardwareStation\Extension.PaymentSample\bin\Debug
22. Copy the Contoso.Commerce.HardwareStation.Extension.PaymentSample.dll and paste it in C:\Program Files (x86)\Microsoft Dynamics 365\70\Retail Modern POS\ClientBroker
23. Open the HardwareStation.Extension.config
24. Add the extension payment library details under the composition section.
```
  <add source="assembly"
value="Contoso.Commerce.HardwareStation.Extension.PaymentSample" />
```
25. Save the file.
26. Close Modern POS if running.
27. Open Task manager and kill all the dllhost.exe

How to validate your extension:

28. Press F5 and deploy the POS to test your customization.
29. Once the POS is launched, login to POS and add any item to transaction.
30. Navigate to the Extension.PaymentSample.proj and Click Debug > Attach to Process and select Dllhost.exe from the available processes list and Click Attach.
31. In the POS hit the pay by Credit card payment method.
32. In the payment view select enter manually and enter the sample card number as "4111111111111111", and security code as 123 and select some future expiry date and month.
33. Click the tender button in the App bar.
34. The debugger should hit the payment SDK and you can validate the extension properties in the SDK and complete the payment flow.