DYNAVISION

Procedure:
Support Document

Date: 7/10/2026

Auteur: Dynavision Product Team

1. First Line Support 4

1.1 Admin Mode VS License Mode 4

2. Procedure bug creation 5

2.1 Reporting an issue 5

2.2 Finding the bug 5

2.2.1 Check the application version 5

2.3 Reproduce 6

2.3.1 Follow up and informing the consultant 6

3. Core – Connect 7

3.1 Expiration dates page 7

3.2 Retention policies 7

3.3 Sales price simulation - Inspect Price List Filters 8

4. Excise Duties 9

4.1 Edit fixed data 9

4.1.1 Excise Tariffs 9

4.1.2 Excise Tariff Codes 9

4.1.3 Packaging 9

4.2 Excise Ledger Entries 10

4.2.1 Create Manual Entry 10

5. Cloud POS Light 11

5.1 Possible POS closing issues 11

6. Technical support 12

6.1 Advanced Order Promising API 12

6.2 FTP Azure Function – to whitelist IP addresses 12

6.3 Price Simulation: Temporary Sales Header & Lines 12

6.3.1 Other functionalities that use the simulation 13

6.4 Extensibility requests 13

6.4.1 Change function visibility 13

Request event publishers 15

6.5 Report Layout Version selection 16

6.5.1 Technical implications 17

6.6 Download setup – Add Message Types 18

6.6.1 Setup 18

6.6.2 (Sender) Create Request Body 18

6.6.3 (Receiver) Parse Request Body / Return Response 19

6.6.4 (Sender) Parse / process the response 20
7. Layouts 20
7.1 Report layout Line Definitions – add custom data 20
7.1.1 Add custom field 21
7.1.2 Add custom data item 22
7.2 CA layouts 23
8. POM 24
8.1 POM add QR code to other custom RDLC's 24
8.2 Publisher to provide structured message 25
9. Promotions 26
9.1 Connect to Sana webshop 26
9.2 Update documents when info from SA NA is received 27
9.2.1 Contributions & declarations and SANA 27
10. Advanced Finance 28
10.1 CODA import statements in locked state 28
10.2 Apply Customer Ledger Entries slow 28
10.3 Isabel Connect – support Custom Reports in export 28
10.4 Payment Reconciliation journal CAMT import 29
10.4.1 Addition to Data Exchange Definition – SEPA CAMT 29
10.4.2 Field Mapping - table 274 (SEPA CAMT to Bank Acc. Reconciliation Line) 30
10.4.3 Result 30
11. Price calculation functions and webshops 31
11.1 Basics 31
11.2 Dynavision 31
11.3 Dynavision Advanced Promotions 31
11.4 Dynavision Advanced Discount & Pricing 32
11.5 Dynavision Contributions & Declarations 32
11.6 Dynavision Reusable Packaging 32
11.7 Solution 32
11.8 Technical 32
11.8.1 Create sales document 32
11.8.2 Update sales document 33
11.8.3 Retrieve the result 33
11.8.4 Custom Dynavision API endpoint 33
11.8.5 Summary 34

1. First Line Support

1.1 Admin Mode VS License Mode

The **Dynavision Setup** page contains two modes:

1. **Admin Mode** (Support Mode)
   Sets the application in administrator mode, allowing helpdesk agents and consultants to perform restricted operations.
2. **License Mode**
   Sets the application in License administrator mode, allowing users to change the Module setup.

2. Procedure bug creation

2.1 Reporting an issue

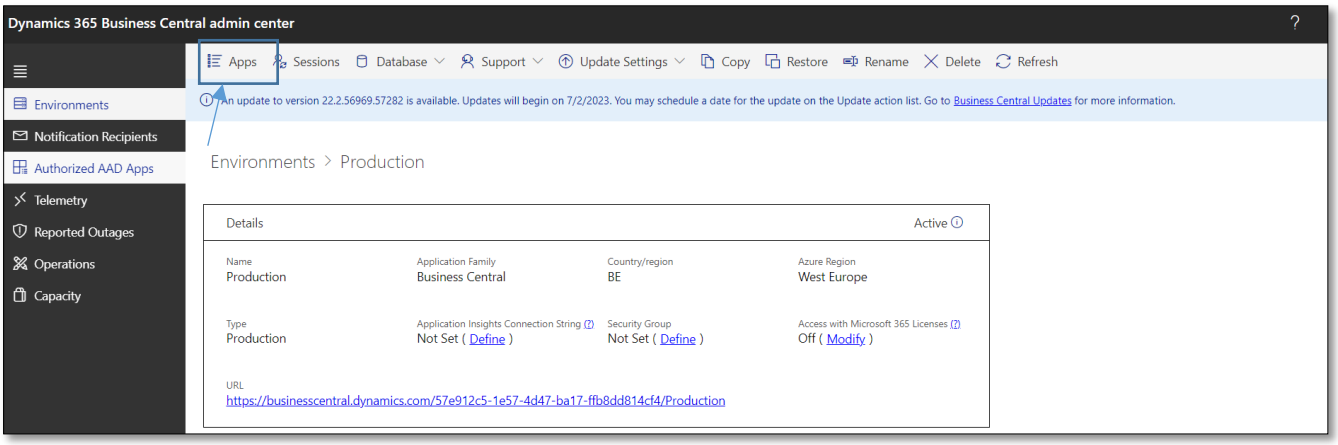
Issues can be reported to support@dynavision.be
Please provide a good description of the issues. If possible create a sandbox environment for the customer tenant on which the issue occurs and supply the link.

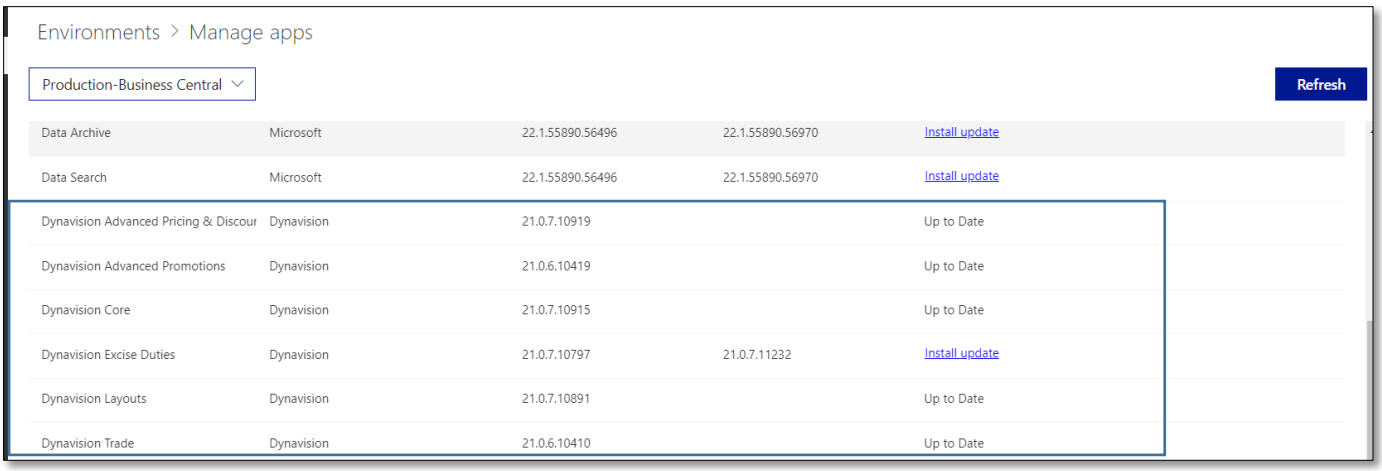
2.2 Finding the bug

When a possible bug is noticed when testing a certain feature that is linked to a Dynavision Application, following steps need to be followed before logging a bug.

2.2.1 Check the application version

The first check that needs to happen is checking the version of the Dynavision Application(s) on the tenant of the customer.
Navigate to Partner center and search for the customer. Select the corresponding customer from the list.
Navigate to the **Service Management** page and select **Dynamics 365 Business Central**.
The **Admin center** will open. Choose the environment on which the bug has been noticed (open the link, not the environment) and select **Apps** in the action bar on top.
In the list of apps, the Dynavision Application(s) can be found and the corresponding version and if an update is needed to the app is visualized.





2.3 Reproduce

When the Dynavision apps and the version of the tenant of the customer seem to be up to date, the next step is to mail the bug information to the support mailbox of Dynavision.

The Dynavision team will try and reproduce the bug on the internal Dynavision test environment.
If the bug cannot be reproduced on the test environment of Dynavision, then the bug has possibly already been fixed as part of another story or another customer who reported this bug. In this case, the fix will be deployed and active when the apps are released.
This information can be communicated back to the consultant that reported the bug. He/She can inform the customer.

When the bug can be reproduced on the test environment of Dynavision, a story will be made by the Product team in order to fix the bug.
A developer will be assigned to fix the bug, and the Dynavision Team will contact the person who sent the bug in order to give updates about when the fix will be ready.

2.3.1 Follow up and informing the consultant

Once the bug has been fixed and tested in the Dynavision test environment, the consultant/partner will be informed that the fix will be available after the next release.

3. Core – Connect

3.1 Expiration dates page

In order to manage the expiration dates of secrets or certificates that are used in Business Central to communicate with external sources, *example given SharePoint,* a page is added in which te expiration dates can be grouped. This can maintain information of different companies in the tenant.

These expiration dates can be added to the page **Expiration Date,** using the links provided on the **Authentication Setup Card**, and the **Document Attachment Setup** page when using SharePoint as database.

1. When a **secret or certificate will expire** (expiration date lies before the set up Notification Period), a **notification** is shown on the Role center on business central.
2. Navigate to the page **Expiration Dates** to view the expiration dates of the setup pages.
3. For the lines that are about to expire, the Expiration date will be indicated in red.
4. Navigate to the source page, using the link in the column **Description** and update the expiration date on the page.
    a. Possible navigate to the correct company to update the information in the company for which the expiration date was added. *Shown in column Company Name*.
5. Update the expiration date on the page **Expiration dates**. This will not be update automatically.

Remark: *don't forget to update this page when an update is done to the source setup pages from which the expiration dates are maintained on the page* ***Expiration dates.***

3.2 Retention policies

It is possible to configure the retention policies of the ESCA tables (*Web Request, Message, FTP Request,…*).

1. Navigate to the **Retention Policies** page.
2. Choose the option **Refresh Allowed Tables** to make sure that all available tables are added to the retention policies list.
3. Select the retention policy that needs to be configured and set up the necessary parameters.
4. Enable the retention policy.

3.3 Sales price simulation - Inspect Price List Filters

After executing a sales price simulation, an option will become available.
Click on **Inspect Price List Filters** to see the parameters that were used to get the shown simulated sales price.

Selecting a record on the page, will update the fact box information that is visible in the fact box pane. This fact box shows the Price Filter fields that were used in the filter sequence.

There is a flow field **Resulting Lines** that can be selected, to see the results that the filter sequence yielded, together with their filters.
*You can also open the page by selecting the filter sequence and clicking on the action **View Resulting Filtered Price List Lines…***

On the page that opens, you see the price list lines that are the result of the filters in the simulation.
You can play around with the filters to see why you are missing a certain result line in the simulation.

4. Excise Duties

4.1 Edit fixed data

4.1.1 Excise Tariffs

The Excise Tariffs table contains the AC4 Codes and County Codes, and for each combination the related Excise Tariff and Excise Category, the Packaging Code, and the range for the minimum and maximum of Production Quantity, Degrees Alcohol, Degrees Plato and Sugar.
Remark: if the value is respectively 999.999 (max. prod. Qty.), 100 (Max. degr. Alcohol), 100(Max. degr. Plato) or 9.999(Max. sugar), it specifies the maximum possible value for that type.

For each AC4 Code and Country Code, the related Excise Tariff Lines are displayed with a starting and ending date, an excise code, the excise amount for 1 unit defined by the calculation method.
The Calculation Method can be one of the following options:

- By HL,
- By HL Alcohol,
- By HL Deg Alcohol,
- By HL Plato, and
- By KG.

The data on this page is **fixed** and cannot be edited by a customer. It is however possible to edit this data in Admin mode if irregularities need to be fixed or when this information needs to be changed for this specific customer. Choose the action **Admin Mode**, trough the **Actions** function in the Actions bar.

4.1.2 Excise Tariff Codes

This page gives information about the Excise Tariff, its required properties.
This data is **fixed** and cannot be modified by a customer. It is however possible to edit this data in Admin mode if irregularities need to be fixed or when this information needs to be changed for this specific customer. Choose the action **Admin Mode**, trough the **Actions** function in the Actions bar.

4.1.3 Packaging

This page shows the different packaging types.
This data is **fixed** and cannot be modified by a customer. It is however possible to edit this data in Admin mode if irregularities need to be fixed or when this information needs to be changed for this specific customer. Choose the action **Admin Mode**, trough the **Actions** function in the Actions bar.

4.2 Excise Ledger Entries

4.2.1 Create Manual Entry

Any movement of items that are subjected to excise and are stored under suspension in the system will be recorded in the excise Ledger.
In case the items that are under suspense are moved in the system and if they stay under suspension, the entries in the ledger for this item will have the **To Declare** set to false. This means that the item is not yet included in the final excise declaration and therefore the excise duties on these items have not yet been paid.

1. Choose the search icon, enter **Excise Ledger Entries**, and then choose the related link.
2. On the page that opens, an overview is shown containing the Excise Entries of all items subjected to Excise. These entries are created for the different Entry Types that are made in the system containing these items (Sale, Purchase, Transfer, …). All incoming and outgoing movements are recorded.
   The excise amounts that are calculated are also noted for the item in the **Lines Tab** when an item is selected.
   The document that has created the entries is linked in the **Document No.** column on the page.

The data on this page is **fixed** and cannot be edited by a customer. It is however possible to edit this data in Admin mode if irregularities need to be fixed or when this information needs to be changed for this specific customer. Choose the action **Admin Mode**, trough the **Actions** function in the Actions bar.

When in **Admin Mode**, it is possible to manually create Excise Ledger Entries. This can be done by choosing the action **Create Manual Entry**.
When this action is chosen, some fields need to be filled in in order to create the entry.

1. **Item No.**
   The item for which an entry needs to be created has to be selected in this field.
   Remark: the list of items from which can be chosen is not filtered.
2. **Variant Code**
   If applicable the variant of the item needs to be chosen for which the entry has to be created.
3. **Unit of Measure Code**
   Specify the Unit of Measure Code that has to be used for the entry.
4. **Quantity**
   Specifies the number of units of the item in the excise entry.
5. **Location Code**
   Specifies the code for the location that the entry is linked to.
6. **Posting Date**
   The posting date for the entry.

7. **Reason Code**
   Specifies the reason code, a supplementary source code that enables the user to trace the entry.

After filling in the necessary fields, choose **OK** to create the entry. A manually created Excise Ledger Entry will always be created with **To Declare** set to False. The line will be inserted with a blank **Entry Type, Document Type** and **Document Number.**

Remark: the function of this action is to correctly display the total excise balance in Business Central when a possible error would have occurred in the system. This does not affect the Excise Declaration.

5. Cloud POS Light

When performing the POS closing, a receipt journal is created for the document payments. The renumber action is disabled on the Receipt journal if the lines are related to a POS Closing.
Using the admin-mode with password, this document number can be edited if needed.

5.1 Possible POS closing issues

When going into Admin-mode, the following items are overwritable:

- Manually adjusting **Status** of closing,
- Manually adjusting **Closing Report Printed** field,
- Manually removing tickets,
- Manually adjust payments on tickets,
- Possible to override balance check,
- Possible to override posting invoice in case of incomplete payment postings.

6. Technical support

6.1 Advanced Order Promising API

- New public codeunit centralizes showing the ATP screen, calculating availability, accepting results, and related helpers;
- Sales Header exposes ESCD_CalculateAdvancedOrderPromising to run a full ATP calculation and accept in one step (for automation from Item Availability or similar entry points);
- Existing ShowAdvancedOrderPromising on Sales Header now delegates to the new codeunit

6.2 FTP Azure Function – to whitelist IP addresses

The FTP Azure Function of Dynavision is located in the US, and not in West Europe. Because of this, some IP addresses need to be whitelisted on the customers firewall for the function to work correctly.
Those IP addresses are:
52.226.143.156,52.226.138.178,52.226.143.193,52.226.143.225,52.226.143.246,52.255.216.13,52.255.217.112,52.255.217.114,52.255.217.127,52.255.217.145,52.255.217.147,52.255.217.155,52.255.217.158,52.255.217.195,52.255.217.199,52.255.217.201,52.255.217.242,52.255.217.248,48.216.145.213,52.255.216.22,52.226.139.43,52.255.216.62,52.255.216.127,52.255.216.188,52.255.216.231,52.255.216.242,52.255.216.249,52.255.216.252,52.255.217.28,52.255.217.68,52.255.217.75,52.226.141.13,52.255.217.81,52.255.217.82,52.255.217.87,52.226.139.26,52.255.217.105,52.255.217.253,52.226.141.73,52.255.218.4,52.255.218.8,52.255.218.11,52.255.218.12,20.119.8.57

6.3 Price Simulation: Temporary Sales Header & Lines

The Sales Simulation uses temporary sales header(s) and sales line(s).
When code on the temporary sales line is executed, it will also trigger any PTE code on the sales line that is on validation/modify triggers and events.
In the exception where you need to get the sales header for the sales line, you should **never** use:

```
SalesHeader.Get("SalesLine.Document Type", SalesLine."Document No.");
```

As this will cause an error like this one:
The simulation will then use the events in that procedure to set the temporary SalesHeader as the SalesHeader, preventing an error like this one:
As the code above will fail to get the salesheader, which is a temporary instance of the table during the Price Simulation.

To solve this issue, All occurencies of “SalesHeader.Get(DocType, DocNo)” on the salesline logic should be replaced with:

- "SalesLine.GetSalesHeader", as this one contains the event on which we subscribe to pass our temporary salesheader for the simulation.
- Or if the .Get() of an actual header is required and the code should not be executed during the simulation (for instance when a PTE that has certain requirements) then they should only use the .Get() instead of the .GetSalesHeader() after they have verified that the salesline itself is not temporary.

Code:

```
procedure GetSalesHeader(var OutSalesHeader: Record "Sales Header"; var OutCurrency: Record Currency)
var
    IsHandled: Boolean;
begin
    OnBeforeGetSalesHeader(Rec, SalesHeader, IsHandled, Currency);
    if IsHandled then
        exit;

    TestField("Document No.");
    if ("Document Type" <> SalesHeader."Document Type") or ("Document No." <> SalesHeader."No.") then begin
        SalesHeader.Get("Document Type", "Document No.");
        if SalesHeader."Currency Code" = '' then
            Currency.InitRoundingPrecision()
        else begin
            SalesHeader.TestField("Currency Factor");
            Currency.Get(SalesHeader."Currency Code");
            Currency.TestField("Amount Rounding Precision");
        end;
    end;

    OnAfterGetSalesHeader(Rec, SalesHeader, Currency);
    OutSalesHeader := SalesHeader;
    OutCurrency := Currency;
end;
```

The simulation will then use the events in the procedure above to set the temporary SalesHeader as the SalesHeader, preventing the error that the SalesHeader does not exist.

6.3.1 Other functionalities that use the simulation

- Shop Labels,
- Sales Price List.

6.4 Extensibility requests

The following types of requests can be submitted to unblock an app:

6.4.1 Change function visibility

- Make a function **public / external** or a similar change so it can be called from an external extension and reuse the business logic.
- Make a page / report variable **protected** so it can be used from an external extension.

Template

- **Object**
  Type number name.

- **Expose**
  FunctionName or VariableName.

Request event publishers

- Add new **integration eve**nts – Get the event that is needed to hook-in to a process.
- **Extensibility enhancements** – Request changes in the application code that will improve extensibility.

Template

- **Object**
  Type number name.
- **Function**
  FunctionName.
- **Signature Integration Event**
  Suggest event publisher name, required parameters, by reference (var) or not, IsHandled pattern.
  - *OnAfterGetMatchScore(GenJournalLine: Record "Gen. Journal Line"; CustLedgerEntry: Record "Cust. Ledger Entry"; var Score: Integer);*
- **Location**
  Screenshot with your Integration Event between begin end tags.
- **Handled Pattern**
  Should the publisher support the IsHandled pattern to allow overriding the standard code execution.
- **Additional Context**
  Why this publisher is required. If possible, please provide additional context on what you would like to achieve.

Example

- Object: codeunit 1250 "Match General Journal Lines"
- Function: FindMatchingCustEntry
- Integration Event:

```
OnAfterGetMatchScore(GenJournalLine: Record "Gen. Journal Line"; CustLedgerEntry: Record "Cust. Ledger Entry"; var Score: Integer);
```

- Location: Just after the score is set

```
local procedure FindMatchingCustEntry(var TempBankStatementMatchingBuffer: Record "Bank Statement Matching Buffer" temporary; var CustLedgerEntry: Record "Cust. Le
var
    Customer: Record Customer;
    Score: Integer;
begin
    Customer.Get(CustLedgerEntry."Customer No.");
    if GenJournalLine.FindSet then
        repeat
            Score := GetMatchScore(GenJournalLine, CustLedgerEntry."Document No.", Customer."No.", CustLedgerEntry."External Document No.",
                Customer.Name, CustLedgerEntry."Remaining Amt. (LCY)", 1, CustLedgerEntry."Posting Date");
            // event: begin
            OnAfterGetMatchScore(GenJournalLine, CustLedgerEntry, Score);
            // event: end
            if Score > 5 then
                TempBankStatementMatchingBuffer.AddMatchCandidate(GenJournalLine."Line No.", CustLedgerEntry."Entry No.", Score,
                  TempBankStatementMatchingBuffer."Account Type"::Customer, CustLedgerEntry."Customer No.");
        until GenJournalLine.Next = 0;
end;
```

It's very important that the Score is var!

6.5 Report Layout Version selection

It is possible to set the version of report that needs to be printed trough development.

1. Add a custom print / email action (or update your existing code) that calls the report and **set the Version**.
2. **Clear the version** after the report has run.

```
pageextension 50100 "Sales Order Card" extends "Sales Order"
{

    actions
    {
        addafter("&Order Confirmation")
        {
            group("Basic Order")
            {
                Caption = 'Basic Order';
                Image = Email;
                action("Send Email Basic Order")
                {
                    ApplicationArea = Basic, Suite;
                    Caption = 'Email Basic Order';
                    Ellipsis = true;
                    Image = Email;
                    ToolTip = 'Send a Basic Order.';

                    trigger OnAction()
                    begin
                        ReportDatastore.SetReportLayoutVersion(1); // Add custom version here
                        DocPrint.EmailSalesHeader(Rec);
                        ReportDatastore.ClearReportLayoutVersion(); // Clear version
                    end;
                }
                action("Print Basic Order")
                {
                    ApplicationArea = Basic, Suite;
                    Caption = 'Print Basic Order';
                    Ellipsis = true;
                    Image = Print;
                    ToolTip = 'Print a Basic Order.';

                    trigger OnAction()
                    begin
                        ReportDatastore.SetReportLayoutVersion(1); // Add custom version here
                        DocPrint.PrintSalesOrder(Rec, 0);
                        ReportDatastore.ClearReportLayoutVersion(); // Clear version
                    end;
                }
            }
        }
    }

    var
        DocPrint: Codeunit "Document-Print";
        ReportDatastore: Codeunit "ESCA Report Datastore";
}
```

When the Dynavision report is ran, it will search for the related **Report Layout Setup** record, using the report's OnInitReport() trigger to call "ESCA Report Layout".Initialize() function. This function will set the appropriate Report Layout Setup record, based on the **Report ID** and the **Version**.

This is mainly intended for PTE development. Dynavision apps will not promote this pattern, as it has various limitations, including:

- All standard code that simply runs a Report.Run(), Report.SaveAs(), ... without knowing the version, will not be able to read the correct 'Report Layout Setup' record
- Reports scheduled in job queue, standard 'Print' actions on pages, standard 'Send E-mail' actions on pages, the 'Save as PDF / Attachment' action, integration with report selections, email scenario support, ...
- If required, these features can be adopted via a PTE to set the version before the report is ran.

6.5.1 Technical implications

- Before this change, the 'Report Layout Setup' primary key consisted of a single 'Report Id' field. This field is now solely used as a unique entry no, with no further value. In order to retrieve the 'Report Layout Setup' for a given report / version combination, you should use the '**New** Report Id' field instead.
- **ReportLayoutSetup.Get(ReportId)** is no longer advised.

6.6 Download setup – Add Message Types

It is always possible to add Message Types to the download framework to make this available for the customer tenants. The Message Types are added by developers and cannot be added using the Admin module.

6.6.1 Setup

Create an "ESCA Message Format" enum extension with a value that implements below interfaces:

- **ESCA Message Format**: allows to process the incoming message in a custom format
- **ESCA Message API**: indicates the communication is using the 'Message API' (and encapsulates the body according to the required Message endpoint
- **ESCA Message Request API**: allows to build the outgoing request content (sender) and parse the request / return a response (receiver)

```
enumextension 50000 "ESC Message Format" extends "ESCA Message Format"
{
    value(50001; "ESC Items")
    {
        Caption = 'BC Items';
        Implementation = "ESCA Message Format" = "ESC Items", "ESCA Message API" = "ESC Message Items", "ESCA Message Request API" = "ESC Items";
    }
}
```

6.6.2 (Sender) Create Request Body

Initiate a call to the custom function to retrieve the needed items (custom action, part of a process, ...).
(optional) Pass required parameters (item no, list of items, filters, ...).
The code in the image below will send a custom built request body to the BC Message API in other company.

```
Items.ImportItem();
```

```
#region ImportItem
internal procedure ImportItem()
var
    MessageType: Record "ESCA Message Type";
begin
    MessageType := GetMessageType(); // Get the message type for the current format

   // ...
   // (optional) Get required parameters (item no, list of items, filters, ...)
   // ...

    MessageType.ImportData(GetItemRequestContent(ItemNo), false);
end;
#endregion ImportItem
```

```
#region GetItemRequestContent
local procedure GetItemRequestContent(ItemNo: Code[20]) Content: Codeunit "Temp Blob"
var
    RequestJObject: JsonObject;
    OutStr: OutStream;
begin
    // ...
    RequestJObject.Add(itemNoLbl, ItemNo);  // Build the custom request body. Pass all parameters needed for receiving API to process request and provide response
    // ...

    Content.CreateOutStream(OutStr);
    RequestJObject.WriteTo(OutStr);
end;
#endregion GetItemRequestContent
```

6.6.3 (Receiver) Parse Request Body / Return Response

```
#region ProcessIncomingRequest
procedure ProcessIncomingRequest(Request: Codeunit "Temp Blob"; var Response: Codeunit "Temp Blob");
var
    Item: Record Item;
    TempItem: Record Item temporary;
    JsonTools: Codeunit "ESC Json Tools";
    InStr: InStream;
    RequestJObject, ResponseJObject : JsonObject;
    ItemNoToken: JsonToken;
begin
    // Parse the request body
    Request.CreateInStream(InStr);
    RequestJObject.ReadFrom(InStr);
    RequestJObject.Get(itemNoLbl, ItemNoToken);
    ...

    // Build the response content
    Item.Get(ItemNoToken.AsValue().AsCode());
    ResponseJObject := JsonTools.Rec2Json(Item);
    ...

    // Return the response as Blob
    ReadJsonObjectIntoTempBlob(ResponseJObject, Response);
end;
#endregion ProcessIncomingRequest
```

6.6.4 (Sender) Parse / process the response

```
#region ProcessIncomingMessage
procedure ProcessIncomingMessage(var Message: Record "ESCA Message")
var
    Item: Record Item;
    JsonTools: Codeunit "ESC Json Tools";
    PageManagement: Codeunit "Page Management";
    TempBlob: Codeunit "Temp Blob";
    InStr: InStream;
    RecObject: JsonObject;
    ItemCouldNotBeRetrievedErr: Label 'Item with no. %1 could not be retrieved';
begin
    // Get the response
    Message.GetAPIResponseFromAttachment(TempBlob, InStr);

    RecObject.ReadFrom(InStr);

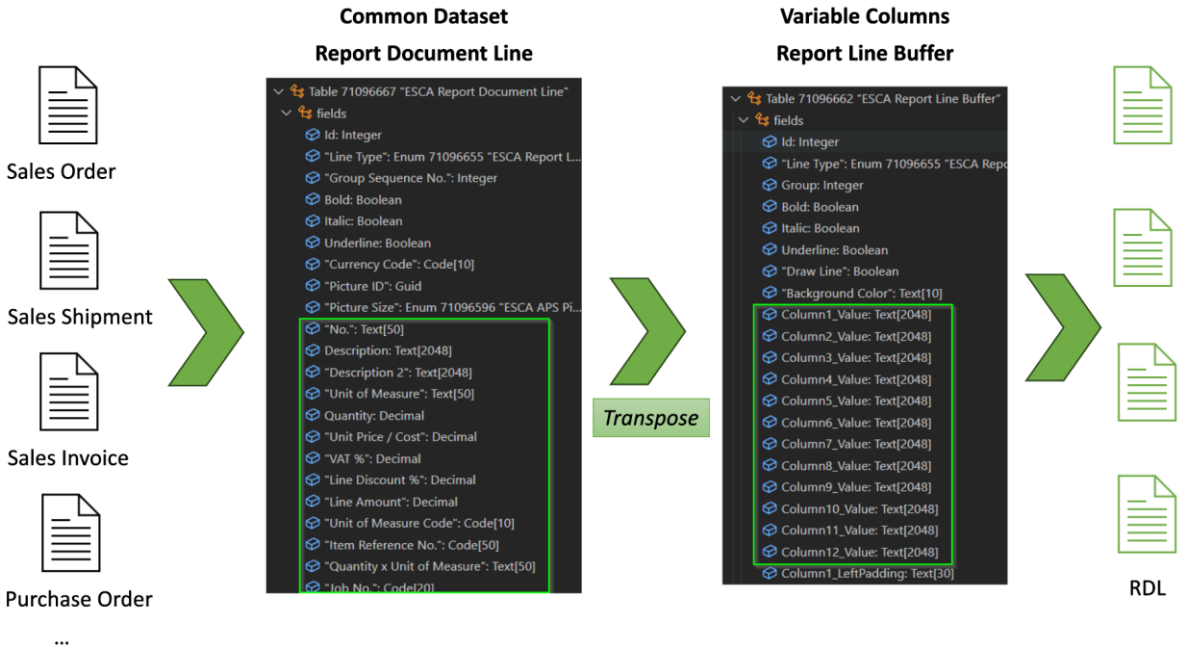
    // Parse the response content / process
    Item := JsonTools.Json2Rec(RecObject, Database::Item);
    Item.Validate("No.", NewItemNo);
    Item.Insert(true);
    ...

end;
#endregion ProcessIncomingMessage
```

7. Layouts

7.1 Report layout Line Definitions – add custom data

It is possible to add custom fields and data items, trough a PTE. How the information can be added is described in the sections below.



7.1.1 Add custom field

Following steps need to be followed in order to add a custom field, that can then be used in the Report Line Definitions for printing on a layout.

1. Add the field.

```
tableextension 50100 "Report Document Line" extends "ESCA Report Document Line"
{
    fields
    {
        field(50100; "Location Code"; Code[10])
        {
            Caption = 'Location Code';
        }
    }
}
```

2. Subscribe to the OnAfterCreateDocumentLine event.

```
#region EventSubscriber Codeunit "ESCA Report Document Lines" OnAfterCreateDocumentLine
[EventSubscriber(ObjectType::Codeunit, Codeunit::"ESCA Report Document Lines", OnAfterCreateDocumentLine, '',
false, false)]
local procedure OnAfterCreateDocumentLine(Source: Variant; var ReportDocumentLine: Record "ESCA Report Document
Line" temporary)
begin
    AddFieldToReportDocumentLine(Source, ReportDocumentLine);
end;
#endregion EventSubscriber Codeunit "ESCA Report Document Lines" OnAfterCreateDocumentLine
```

3. Set the value for the field.

```
case RecordRef.Number() of
    Database::"Sales Line":
        ReportDocumentLine."Location Code" := RecordRef.Field(SalesLine.FieldNo("Location Code")).Value;
end;
```

4. Add the custom field to the Report Line Definitions of the reports on which the field need to be printed. Add the header line in case a text constant needs to be printed, and the Normal line with the newly defined field.

7.1.2 Add custom data item

7.1.2.1 Text line

In order to add a text line data item to the Report Line Definitions, use the **ReportLayout.AddDocument** function to add a text value that needs to be printed on the layout.

Example given:

```
reportextension 50100 "Sales Quote" extends "ESCR Sales Quote"
{
    dataset
    {
        addafter(Intrastat)
        {
            dataitem("Sample"; Integer)
            {
                DataItemTableView = where(number = filter(1 .. 5));

                #region OnAfterGetRecord
                trigger OnAfterGetRecord()
                begin
                    ReportLayout.AddDocumentLine(Buffer, 'Sample ' + format(Number), LineSorting);
                end;
                #endregion OnAfterGetRecord
```

The result of this example is:

Nr.	Omschrijving	Aantal	Eenheid
1936-S	BERLIJN Bezoekersstoel, geel	10	Stuk
	Sample 1		
	Sample 2		
	Sample 3		
	Sample 4		
	Sample 5		
1000	ATHENE Tafel	10	Stuk
	Sample 1		
	Sample 2		
	Sample 3		
	Sample 4		

7.1.2.2 Variant

In order to add a data item variant to the Report Line Definitions, follow the steps mentioned below:

1. use the **ReportLayout.AddDocument** function to add a record or multiple fields that need to be printed on the layout.

```
#region OnAfterGetRecord
trigger OnAfterGetRecord()
begin
    ReportLayout.AddDocumentLine(Buffer, Sample, LineSorting);
end;
#endregion OnAfterGetRecord
```

2. Subscribe to the event **OnAfterCreateDocumentLine**.

```
#region EventSubscriber Codeunit "ESCA Report Document Lines" OnAfterCreateDocumentLine
[EventSubscriber(ObjectType::Codeunit, Codeunit::"ESCA Report Document Lines", OnAfterCreateDocumentLine, '',
false, false)]
local procedure OnAfterCreateDocumentLine(Source: Variant; var ReportDocumentLine: Record "ESCA Report Document
Line" temporary)
begin
    AddFieldToReportDocumentLine(Source, ReportDocumentLine);
end;
#endregion EventSubscriber Codeunit "ESCA Report Document Lines" OnAfterCreateDocumentLine
```

3. Set the field values.

```
#region AddFieldToReportDocumentLine
local procedure AddFieldToReportDocumentLine(Source: Variant; var ReportDocumentLine: Record "ESCA
var
    SalesLine: Record "Sales Line";
    DataTypeManagement: Codeunit "Data Type Management";
    RecordRef: RecordRef;
    Integer: Record Integer;
begin
    if not DataTypeManagement.GetRecordRef(Source, RecordRef) then
        exit;

    case RecordRef.Number() of
        Database::Integer:
            begin
                RecordRef.SetTable(Integer);
                ReportDocumentLine."Line Type" := ReportDocumentLine."Line Type"::Text;
                ReportDocumentLine.Description := 'Another Sample Line ' + format(Integer.Number);
            end;
    end;
end;
#endregion AddFieldToReportDocumentLine
```

4. Following will be printed on the layout.

Nr.	Omschrijving
1936-S	BERLIJN Bezoekersstoel, geel
	Sample Line 1
	Another Sample Line 1
	Sample Line 2
	Another Sample Line 2

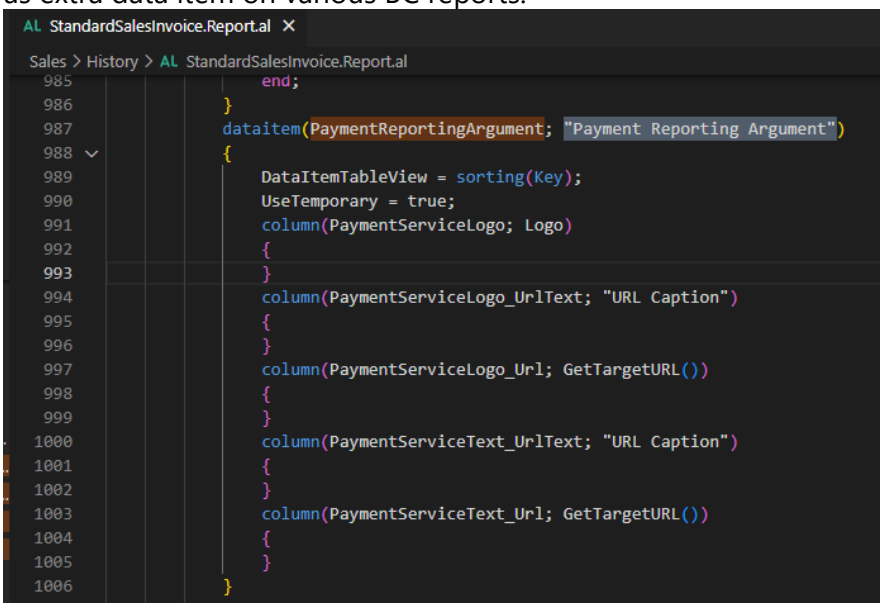
7.2 CA layouts

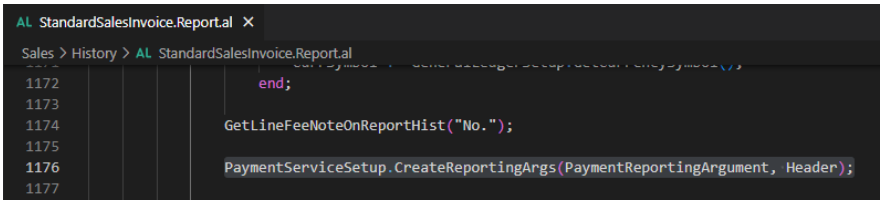
For Canadian customers, the VAT summaries on the report layouts need to be printed differently.
This functionality is added in the Dynavision Layouts CA version, which can be installed on these environments, and will take the correct VAT formatting on the report layouts when printed.

8. POM

8.1 POM add QR code to other custom RDLC's

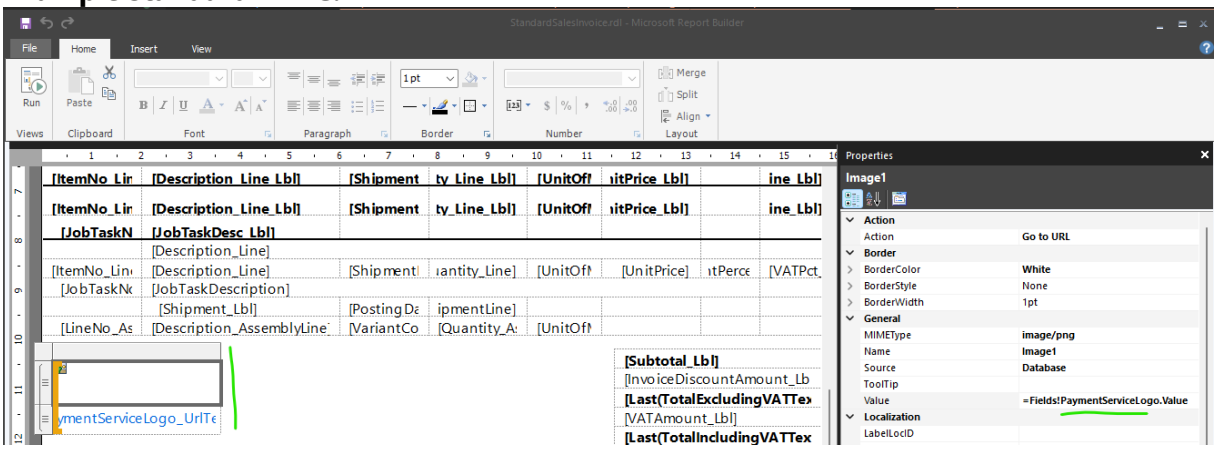
The POM Payment Service application works the same way as other BC payment service connectors. This is by default supported when printing the BC invoice reports.
This functionality uses the (temporary) Payment Reporting Argument table, which is available as extra data item on various BC reports.





By taking over that same Payment Reporting Argument logic on the custom reports, it is possible to print the POM QR code on these reports.

Example standard RDLC:



8.2 Publisher to provide structured message

A custom structured message can be used to generate the POM hyperlink that will be included on the sales invoice document that is sent to customers.
(*This can be used if there is no advanced finance BE installed that created the structured message for the invoice.)*
To make this possible, there is an event publisher added.

The sample code that can be used is as follows.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"ESCV POM Generate
Hyperlink", OnBeforeGetStructuredMessage, '', false, false)]
local procedure OnBeforeGetStructuredMessage(SalesInvoiceHeader:
Record "Sales Invoice Header"; var StructuredMessage: Text;
var IsHandled: Boolean)
begin
  If IsHandled then
    exit;

  StructuredMessage := SalesInvoiceHeader."XYZ Structured Message"; //
Use custom Structured Message field
  IsHandled := true;
end;
```

9. Promotions

9.1 Connect to Sana webshop

In order to allow Sana to use the Dynavision promotions application a function is added that can be called from PTE code, to communicate with the promotions app.
Following PTE code can be used to make the connection between Sana and the promotions app.

```
codeunit 50100 "ESC Sana - Promotions Mgt"
{
    SingleInstance = true;

    #region EventSubscriber Codeunit 11123320 OnAfterAddDocumentLines
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"SC - Order/Basket Functions",
'OnAfterAddDocumentLines', '', true, true)]
    local procedure OnAfterAddDocumentLines_C11123320(var SalesHeader: Record "Sales
Header")
    begin
        SalesHeader.ESCQ_CheckPromotions();
    end;
    #endregion EventSubscriber Codeunit 11123320 OnAfterAddDocumentLines

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"SC - Order/Basket Functions",
'OnGetSalesLine', '', false, false)]
    local procedure OnGetSalesLine(var XMLNodeBuff: Record "SC - XML Buffer (dotNET)"; var
SalesLine: Record "Sales Line"; var Params: Record "SC - Parameters Collection");
    begin
        SalesLine.CalcFields("ESCQ Created by Promotion");
        XMLNodeBuff.AddFieldElement('IsSupplementary', format((SalesLine."ESCQ Created by
Promotion"), 0, 2));
        XMLNodeBuff.AddFieldElement('IsReadOnlyLine', format((SalesLine."ESCQ Created by
Promotion"), 0, 2));
    end;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"SC - Order/Basket Functions",
'OnBeforeAddDocumentLine', '', false, false)]
    local procedure OnBeforeAddDocumentLine(var SalesLine: Record "Sales Line"; var Params:
Record "SC - Parameters Collection");
    begin
        SalesLine.CalcFields("ESCQ Created by Promotion");
        if SalesLine."ESCQ Created by Promotion" then
            Params.LineType := '';
    end;
}
```

9.2 Update documents when info from SANA is received

Certain logic in the Dynavision apps will only be executed when information is manually filled in on the Business Central Pages. When integrations are in place (e.g. webshops), it is recommended to specifically call on this logic after creation of the order lines.

9.2.1 Contributions & declarations and SANA

The insertion of the contributions and declarations, when a document is created by a Sana webshop can be achieved using the code snippet that is available.
The code snippet contains following.

- **OnAfterAddDocumentLines:** after creation of the sales order lines in Business Central (based on the items that were added to the SANA basket), the contributions need to be calculated. This can be configured line per line (option 1), or for the whole document (option 2).
- **OnGetSalesLine:** the calculated contribution needs to be made visible in the SANA basket. It is advised to mark these lines specifically, to make sure that they cannot be edited by the users.

10. Advanced Finance

10.1 CODA import statements in locked state

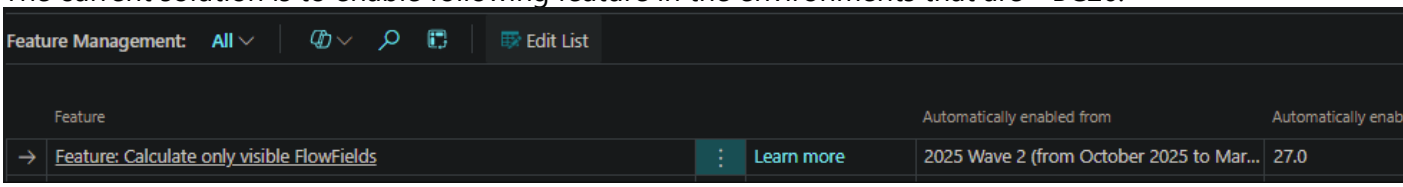
Sometimes, when importing CODA statements, the standard functionality of Business Central goes into a locked state.
There is an issue in Business Central where, during the import, there is some data that gets corrupted in the CODA Statement Source Line table (2000042).

There is extra logic added to remove the **Coda Statement Source Lines** for the current bank account, when a CODA statement is imported in the system (manually or via Isabel).

There is also an extra function added on the **Bank Account, CODA Statement Source Lines**, which allows to consult the (staging) source lines, and allows the deletion in case of troubleshooting.

10.2 Apply Customer Ledger Entries slow

It is possible that the apply customer ledger entries is slow. The reason is flowfields that are calculated in the background, slowing this process.
The current solution is to enable following feature in the environments that are >BC26.



10.3 Isabel Connect – support Custom Reports in export

The Send To Isabel action is disabled if the report is not supported.

If the customer wants to use a custom report, the following needs to be done:

- Handle the **OnIsSendAvailable** event in the Export Payments Codeunit **to make the send action available.**
- Handle the **OnGenerateXMLFile** event in the Export Payments Codeunit **to supply the contents to send to Isabel** ==> If no content is supplied an error "**There is nothing to send**" will be triggered.

10.4 Payment Reconciliation journal CAMT import

In Business Central, there are options added by Dynavision to import SEPA CAMT files and process these files in the Payment Reconciliation Journal.

In this section the setup will be discussed that needs to be done to import all necessary fields out of the SEPA CAMT file and link these to the matching fields in Business Central.
The data exchange definition that is discussed below can also be accessed via the **Get Default Data** functionality of Dynavision.

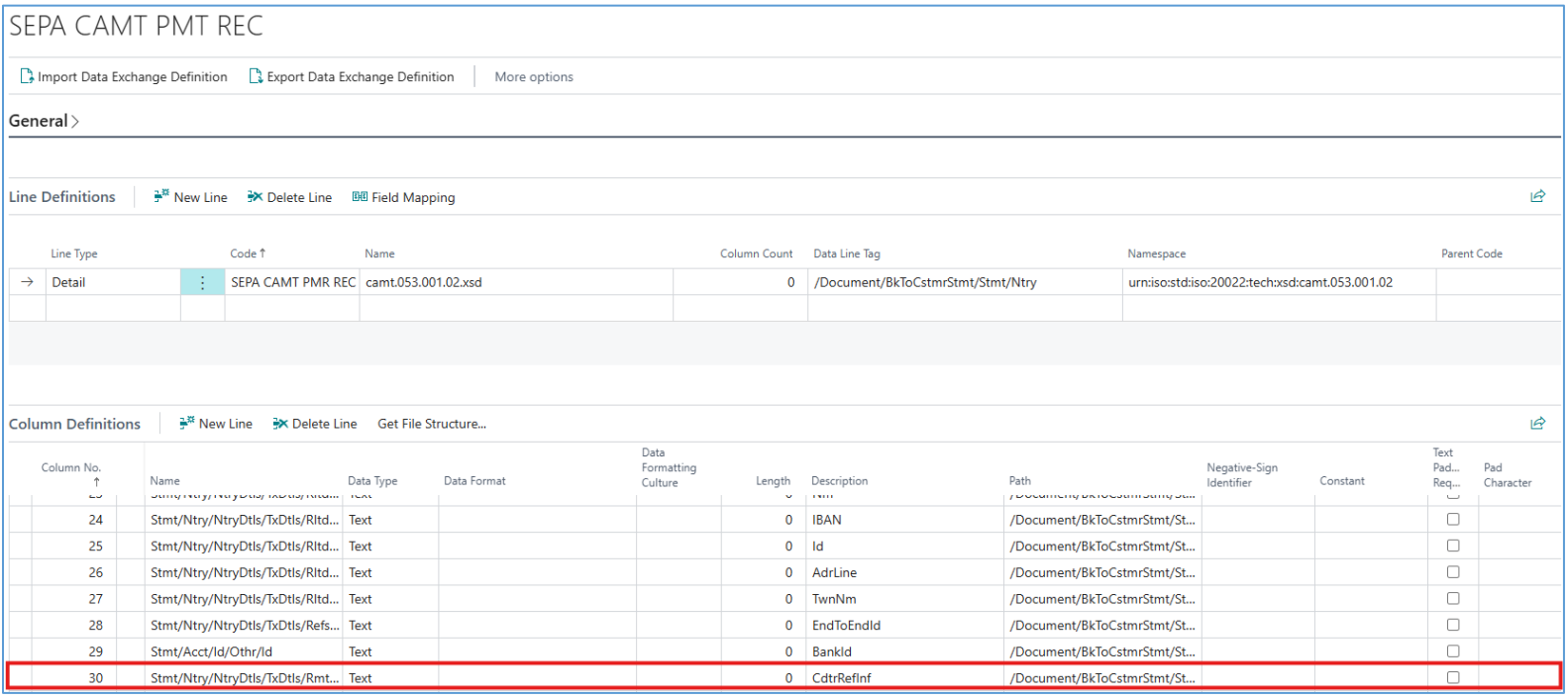
10.4.1 Addition to Data Exchange Definition – SEPA CAMT

Following field needs to be added to the Data Exchange Definition card that is created for the SEPA CAMT file. This column is important, as the Payment Reconciliation Journal will use this information to match the imported payment lines to the open customer ledger entries.

Add following column definition:

- **Column**: 30
- **Name**: Stmt/Ntry/NtryDtls/TxDtls/RmtInf/Strd/CdtrRefInf/Ref
- **Data Type**: Text
- **Path**: /Document/BkToCstmrStmt/Stmt/Ntry/NtryDtls/TxDtls/RmtInf/Strd/CdtrRefInf/Ref
- **Description**: CdtrRefInf

Example screenshot:

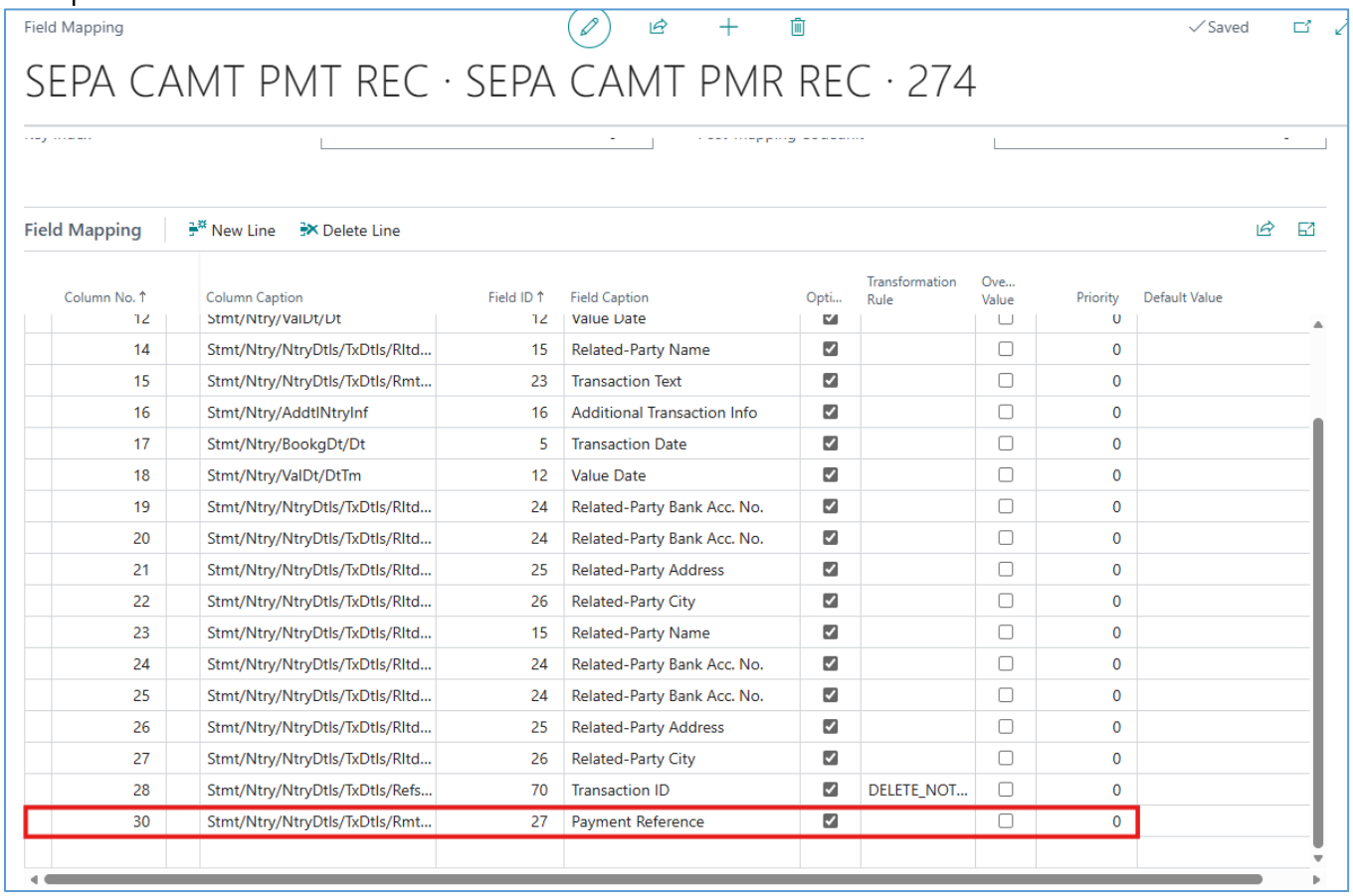


10.4.2 Field Mapping - table 274 (SEPA CAMT to Bank Acc. Reconciliation Line)

Now the newly added Column definition needs to be added to the Field Mapping table.
Add following to the lines:

- **Column**: 30
- **Field ID**: 27

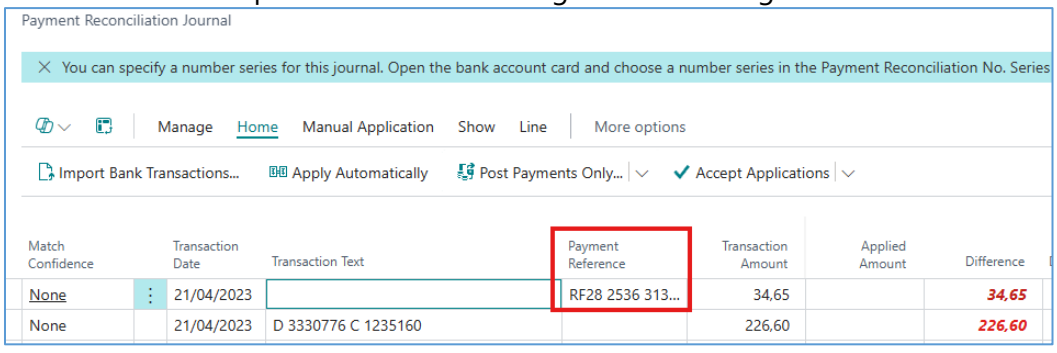
Example screenshot:



Column No.	Column Caption	Field ID	Field Caption	Opti...	Transformation Rule	Ove... Value	Priority	Default Value
12	Stmt/Ntry/ValDt/Dt	12	Value Date	☑		☐	0	
14	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	15	Related-Party Name	☑		☐	0	
15	Stmt/Ntry/NtryDtls/TxDtls/Rmt...	23	Transaction Text	☑		☐	0	
16	Stmt/Ntry/AddtlNtryInf	16	Additional Transaction Info	☑		☐	0	
17	Stmt/Ntry/BookgDt/Dt	5	Transaction Date	☑		☐	0	
18	Stmt/Ntry/ValDt/DtTm	12	Value Date	☑		☐	0	
19	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	24	Related-Party Bank Acc. No.	☑		☐	0	
20	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	24	Related-Party Bank Acc. No.	☑		☐	0	
21	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	25	Related-Party Address	☑		☐	0	
22	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	26	Related-Party City	☑		☐	0	
23	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	15	Related-Party Name	☑		☐	0	
24	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	24	Related-Party Bank Acc. No.	☑		☐	0	
25	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	24	Related-Party Bank Acc. No.	☑		☐	0	
26	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	25	Related-Party Address	☑		☐	0	
27	Stmt/Ntry/NtryDtls/TxDtls/Rltd...	26	Related-Party City	☑		☐	0	
28	Stmt/Ntry/NtryDtls/TxDtls/Refs...	70	Transaction ID	☑	DELETE_NOT...	☐	0	
30	Stmt/Ntry/NtryDtls/TxDtls/Rmt...	27	Payment Reference	☑		☐	0	

10.4.3 Result

After linking the Data Exchange Definition to the bank account card, the SEPA CAMT file can be imported in the Payment Reconciliation Journal.
The result of the import will look something like in the image below.



Match Confidence	Transaction Date	Transaction Text	Payment Reference	Transaction Amount	Applied Amount	Difference
None	21/04/2023		RF28 2536 313...	34,65		34,65
None	21/04/2023	D 3330776 C 1235160		226,60		226,60

11. Price calculation functions and webshops

11.1 Basics

To calculate sales price in Business Central, sales documents (such as quote / order / invoice) can be used for accurate price calculation. These sales documents contain the required context for the price calculation including (but not limited to) the **customer, calculation date (order date), price incl. VAT, currency, item, quantity, unit of measure,** …
When creating a sales document and adding sales line to it, Business Central will perform various price related calculations:

- The **unit price** will be calculated based on the document / line context, using the prices from the sales price lists. This includes custom price calculation methods.
- The **line discount %** will be calculated based on the document / line context, using the discounts from the sales price lists. This includes custom price calculation methods.
- The **invoice discount amount** will be calculated based on the customer invoice discounts.
- Additionally, the sales lines will be updated with the correct **VAT amounts, VAT differences, currency rounding,** …

11.2 Dynavision

Dynavision offers various add-ons that integrate with the Business Central price calculation logic. Most of these features are however integrated with manual entry of documents via the sales document pages. Other routines creating sales documents / lines, do not necessarily require all features to be applied.
Technical: These features depend on a combination of both page validation triggers and table / field validation triggers. Standard routines simply creating sales lines through code would not necessarily provide the exact same results.
In case specific features (e.g. select multiple items, suggest sales lines via AI, explode BOM, copy document, special orders, creation of data through API, EDI order import, web shop integration…) require to support any of the apps functionalities, these can be extended by calling the app's custom logic.

11.3 Dynavision Advanced Promotions

This app allows you to maintain 2 major types of promotions:

- Basic promotions: these promotions require the context of the current sales **line**, and are instantly calculated upon entry of a new sales line (via the sales order pages). If a valid promotion is found, either the current sales line is updated (e.g. apply a 'Line Discount %') , either a new (linked) sales line is created (e.g. apply an 'Item Charge' to give free items)
- Advanced promotions: these promotions require the context of the current sales **document**, and are calculated via a separate step 'Check Promotions'. The promotion

engine will scan the context of the document and group item quantities to find valid promotions. This results in new or updated sales lines.

11.4 Dynavision Advanced Discount & Pricing

This app provides various enhancements on top of the sales price lists (e.g. discount amounts, composed discounts, …) and provides extra price calculation methods. As these integrate with the standard Business Central processes, the results are immediately applied to the sales line, by updating the **unit price / line discount %** fields.

11.5 Dynavision Contributions & Declarations

This app provides various enhancements to insert additional charges (= extra linked sales lines) based on the selected item / quantity / unit of measure.

11.6 Dynavision Reusable Packaging

This app provides various enhancements to insert additional charges (= extra linked sales lines) based on the selected item / quantity / unit of measure.

11.7 Solution

Next to extending specific features to call the app's custom logic, we have provided a single point of entry to ensure the sales document is updated correctly, based on the installed apps. A separate 'Update Document' action is available on the sales document, which can be configured to include:

- Automated calculation of basic / advanced promotions on both line / document level
- Automated insertion of contributions & declarations for the document
- Automated insertion of reusable packaging for the document
- …

11.8 Technical

For external systems to benefit from all price calculation functionality of Business Central, it is advised to create a (non-temporary) sales document with sales lines. Creating a sales **quote** (possible marked with a custom status 'Draft') with sales lines would a possible solution.

11.8.1 Create sales document

The standard **salesQuote** and **salesQuoteLine** API from Business Central provide the required data for creating the sales document. In case these API lack important fields, it is advised to copy these API's and create custom API endpoints (pages).

11.8.2 Update sales document

As the above API would trigger both table / field validation, price calculation will be ran for the individual lines. Since advanced promotions require the context of the entire document to be available, the sales document is to be created first, followed by calling the 'Update Document' logic.
Dynavision provides a custom **salesQuote** (and **salesOrder**) API endpoint, with a bound action to call the Update function.

```
POST {{directUrl}}/salesOrders({{orderid}})/Microsoft.NAV.Update HTTP/1.1
```

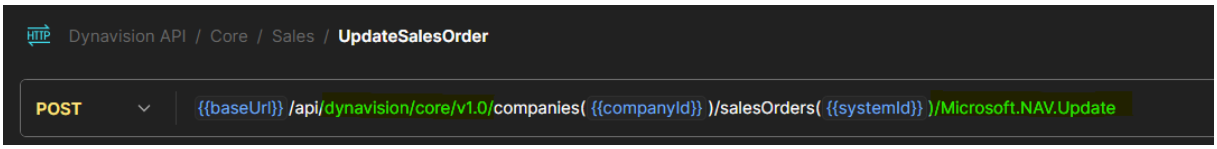
11.8.3 Retrieve the result

The standard **salesQuote** API from Business Central allow to retrieve the updated sales header to get the updated **discountAmount**.
The standard **salesQuoteLine** API from Business Central allow to retrieve the updated sales lines to get the updated **unitPrice, discountAmount, discountPercent**. The API will also return new sales lines inserted by the promotion engine.
After retrieval of the price information, the external system can be configured to

- delete the created sales document using the **salesQuote** API
- keep the document and have a job queue removed the sales document (with the custom status 'Draft')
- keep the document and use it for applying updates / new price calculation / final order creation
- …

11.8.4 Custom Dynavision API endpoint

Dynavision provides a custom Dynavision API endpoint to call the 'Update Document' feature after creating of the sales order.



11.8.4.1 Steps :

1. Creation of the sales order (or quote) can be performed using the standard BC (or a custom) API.
2. As promotions should be calculated once **all** document lines are known, a call to the 'Update' function needs to be raised, to trigger the promotion calculation for the full document.
3. Retrievel of the sales line results can be performed using the standard BC (or a custom) API again.

11.8.5 Summary

