Extension Development

This chapter lists important integration events and examples that help in developing an extension for SQUEEZE for BC. Important: Customizations may only be performed by appropriately trained consultants/developers. Furthermore, these are always outside of the standard support.

Adding a field in the validation (up to Squeeze Version 1.*)

The following is an example extension that shows how developers can add fields in validation.

Required integration events incl. sample procedures:

```
codeunit 50100 EventSubs
{
    //
    //Any header field that you want to transfer to the resulting document, has to be added to the source JSON Object
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.", 'OnBeforeAddLineToDocumentJObj', '', false, false)]
    local procedure SQZDocMgtOnBeforeAddLineToDocumentJObj(var DocumentJObj: JsonObject; DocHeader: Record "DXP SQZ Document Header")
    begin
        AddCustomHeaderFieldsToJson(DocumentJObj, DocHeader);
    end;


    //
    // Any line field that you want to transfer to the resulting document, has to be added to the source JSON Object
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.", 'OnBeforeAddLineJObjToLineJArray', '', false, false)]
    local procedure SQZDocMgtOnBeforeAddLineJObjToLineJArray(var LineJObj: JsonObject; DocLine: Record "DXP SQZ Document Line")
    begin
        AddCustomLineFieldsToJson(LineJObj, DocLine);
    end;
```

```
    //
    // [If you want to perform plausibility checks on the newly added header field, this is
the place]
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.",
'OnAfterDoHeaderPlausibilityChecks', '', false, false)]
    local procedure SQZDocMgtOnBeforeDoHeaderPlausibilityChecks(DocHeader: Record "DXP SQZ
Document Header"; var PlausibilityCheck: Codeunit "DXP Plausiblity Check Mgt.")
    begin
        CheckCustomer(PlausibilityCheck, DocHeader."Customer No.");
    end;

    //
    // [If you want to perform plausibility checks on the newly added line field, this is the
place]
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.",
'OnAfterDoLinePlausibilityChecks', '', false, false)]
    local procedure SQZDocMgtOnAfterDoLinePlausibilityChecks(DocHeader: Record "DXP SQZ
Document Header"; DocLine: Record "DXP SQZ Document Line"; var PlausibilityCheck: Codeunit
"DXP Plausiblity Check Mgt.")
    begin
        CheckCustomer(PlausibilityCheck, DocLine."Customer No.");
    end;

    //
    // The value of the previously extended JSON Object now has to saved to the corresponding
field in the SQUEEZE Document Line
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.",
'OnBeforeModifySQUEEZEDocumentHeader', '', false, false)]
    local procedure SqzDocMgtOnBeforeModifySQUEEZEDocumentHeader(var DocHeader: Record "DXP
SQZ Document Header"; RawJson: JsonObject)
    var
        JSONHelper: Codeunit "DXP Json Helper";
        SQZApiTokenMgt: Codeunit "DXP SQZ API Token Mgt.";
        TokenMgt: Codeunit TokenMgt;
    begin
        DocHeader."Customer No." := COPYSTR(JsonHelper.ValAsTxt(RawJson,
```

```
SQZApiTokenMgt.GetHeaderValueByNameTok(TokenMgt.GetCustomerNoTok()), false), 1,
MaxStrLen(DocHeader."Customer No."));
    end;

    //
    // The value of the previously extended JSON Object now has to saved to the corresponding
field in the SQUEEZE Document Header
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.",
'OnBeforeTransferRecognizedValuesOnAfterAssignRecognizedValues', '', false, false)]
    local procedure
SQZDocMgtOnBeforeTransferRecognizedValuesOnAfterAssignRecognizedValues(RowJTok: JsonToken;
var DocLine: Record "DXP SQZ Document Line")
    var
        JSONHelper: Codeunit "DXP Json Helper";
        SQZApiTokenMgt: Codeunit "DXP SQZ API Token Mgt.";
        TokenMgt: Codeunit TokenMgt;
    begin
        DocLine."Customer No." := CopyStr(JsonHelper.ValAsTxt(RowJTok.AsObject(),
SQZApiTokenMgt.GetCellValueTok(TokenMgt.GetCustomerNoTok()), false), 1,
MaxStrLen(DocLine."Customer No."));
    end;

    //
    // The newly added fields now have to be added to the field mapping
    // Otherwise the recognized values will be saved as meta data
    // This step is necessary to make sure that the fields can be highlighted in the SQUEEZE
Viewer
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ API Mgt.",
'OnAfterInitMapping', '', false, false)]
    local procedure SQZApiMgtOnAfterInitMapping(var HeaderMapping: Dictionary of [Text,
Integer]; var LineMapping: Dictionary of [Text, Integer]; DocClass: Enum "DXP Document Class")
    var
        SQZDocHeader: Record "DXP SQZ Document Header";
        SQZDocLine: Record "DXP SQZ Document Line";
        TokenMgt: Codeunit TokenMgt;
    begin
        // The standard (without installed extensions) supports one Document Class
        case DocClass of
```

```
            DocClass::"DXP Invoice / Credit Memo":
                begin
                    HeaderMapping.Add(TokenMgt.GetCustomerNoTok(),
SQZDocHeader.FieldNo("Customer No."));
                    LineMapping.Add(TokenMgt.GetCustomerNoTok(), SQZDocLine.FieldNo("Customer
No."));
                end;
        end;
    end;

    //
    // The value in the JSON Object now has to be transferred to the Purchase Header
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP Document Transfer Mgt.",
'OnAfterCreatePurchaseHeader', '', false, false)]
    local procedure DocTransferMgtOnAfterCreatePurchaseHeader(var JObject: JsonObject;
PurchaseHeader: Record "Purchase Header")
    begin
        TransferCustomHeaderFieldFromJsonToPurchaseHeader(JObject, PurchaseHeader);
    end;

    //
    // The value in the JSON Object now has to be transferred to the Purchase Line
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP Document Transfer Mgt.",
'OnBeforeInsertPurchaseLine', '', false, false)]
    local procedure DocTransferMgtOnAfterInsertPurchaseLineOnBeforeAddDimensions(var
JObjectLine: JsonObject; var PurchaseLine: Record "Purchase Line")
    begin
        TransferCustomHeaderFieldFromJsonToPurchaseLine(JObjectLine, PurchaseLine);
    end;

    local procedure TransferCustomHeaderFieldFromJsonToPurchaseHeader(var JObject:
JsonObject; PurchaseHeader: Record "Purchase Header")
    var
        JsonHelper: Codeunit "DXP Json Helper";
        TokenMgt: Codeunit TokenMgt;
    begin
        PurchaseHeader.Validate("DXP Customer No.", JsonHelper.ValAsTxt(JObject,
TokenMgt.GetCustomerNoTok(), false));
```

```
        PurchaseHeader.Modify();
    end;

    local procedure TransferCustomHeaderFieldFromJsonToPurchaseLine(var JObjectLine:
JsonObject; var PurchaseLine: Record "Purchase Line")
    var
        JsonHelper: Codeunit "DXP Json Helper";
        TokenMgt: Codeunit TokenMgt;
    begin
        PurchaseLine.Validate("DXP Customer No.", JsonHelper.ValAsTxt(JObjectLine,
TokenMgt.GetCustomerNoTok(), false));
    end;

    local procedure AddCustomHeaderFieldsToJson(var DocumentJObj: JsonObject;
SQZDocumentHeader: Record "DXP SQZ Document Header")
    var
        TokenMgt: Codeunit TokenMgt;
    begin
        DocumentJObj.Add(TokenMgt.GetCustomerNoTok(), SQZDocumentHeader."Customer No.");
    end;

    local procedure AddCustomLineFieldsToJson(var LineJObj: JsonObject; SQZDocumentLine:
Record "DXP SQZ Document Line")
    var
        TokenMgt: Codeunit TokenMgt;
    begin
        LineJObj.Add(TokenMgt.GetCustomerNoTok(), SQZDocumentLine."Customer No.");
    end;

    [TryFunction]
    local procedure CustomerExists(CustomerNo: Code[20])
    var
        Customer: Record Customer;
    begin
        Customer.Get(CustomerNo);
    end;

    local procedure CheckCustomer(var PlausibilityCheck: Codeunit "DXP Plausiblity Check
Mgt."; CustomerNo: Code[20]): Boolean
    begin
```

```
        if CustomerNo = '' then
            exit(false);

        if not CustomerExists(CustomerNo) then begin
            PlausibilityCheck.AddPlausibilityCheckEntry(GetLastErrorText(), Page::"Customer
List");
            exit(false);
        end;

        exit(true);
    end;
}
```

Table Extensions:

SQUEEZE 4 BC

```
tableextension 50100 "SQZ Doc. Header Ext." extends "DXP SQZ Document Header"
{
    fields
    {
        field(50100; "Customer No."; Code[20])
        {
            TableRelation = Customer;
            ValidateTableRelation = false;
            Caption = 'Customer No.';
        }
    }
}
```

```
tableextension 50101 "SQZ Doc. Line Ext." extends "DXP SQZ Document Line"
{
    fields
    {
        field(50100; "Customer No."; Code[20])
        {
            TableRelation = Customer;
            ValidateTableRelation = false;
            Caption = 'Customer No.';
        }
```

```
    }
}
```

Target document (here: purchasing document)

```
tableextension 50102 "Purchase Header Ext." extends "Purchase Header"
{
    fields
    {
        field(50100; "DXP Customer No."; Code[20])
        {
            DataClassification = CustomerContent;
            TableRelation = Customer;
            Caption = 'DEXPRO Customer No.';
        }
    }
}
```

Page Extensions:

SQUEEZE 4 BC

```
pageextension 50100 "SQZ Document Ext." extends "DXP SQZ Document"
{
    layout
    {
        addafter(BuyFromVendorInternal)
        {
            field("Customer No. Internal"; Rec."Customer No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the value of the Customer No. field.';
                trigger OnAssistEdit()
                begin
                    MarkField(Rec.FieldNo("Customer No."), Rec."Customer No.",
Rec);
                end;
```

```
                trigger OnValidate()
                begin
                    CheckPlausibility();
                end;
            }
        }
        addafter(BuyFromVendor)
        {
            field("Customer No."; Rec."Customer No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the value of the Customer No. field.';
                trigger OnAssistEdit()
                var
                    ApiMgt: Codeunit "DXP SQZ API Mgt.";
                    ViewerResident: Codeunit "DXP SQUEEZE Viewer Resident";
                begin
                    MarkField(Rec.FieldNo("Customer No."), Rec."Customer No.",
Rec);
                end;

                trigger OnValidate()
                begin
                    CheckPlausibility();
                end;
            }
        }
    }

    local procedure MarkField(AppFldNo: Integer; FieldVal: Variant; DocHeader: Record "DXP
SQZ Document Header")
    var
        ApiMgt: Codeunit "DXP SQZ API Mgt.";
        ViewerResident: Codeunit "DXP SQUEEZE Viewer Resident";
    begin
        ViewerResident := GetViewerResident();
        ApiMgt.MarkField(AppFldNo, FieldVal, ViewerResident, DocHeader);
    end;
}
```

```
pageextension 50101 "SQZ Document Sf. Ext." extends "DXP SQZ Document Subform"
{
    layout
    {
        addafter("No.")
        {
            field("Customer No."; Rec."Customer No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the value of the Customer No. field.';
            }
        }
    }
}
```

Target document (here: purchasing document)

```
pageextension 50102 "Purchase Invoice Ext." extends "Purchase Invoice"
{
    layout
    {
        addafter("Buy-from Vendor No.")
        {
            field("DXP Customer No."; Rec."DXP Customer No.")
            {
                ToolTip = 'Specifies the value of the DEXPRO Customer No.
field.';
                ApplicationArea = all;
            }
        }
    }
}
```

```
pageextension 50103 "Purchase Inv. Sf Ext." extends "Purch. Invoice Subform"
{
    layout
    {
        addafter("No.")
        {
            field("DXP Customer No."; Rec."DXP Customer No.")
```

```
            {
                ToolTip = 'Specifies the value of the DEXPRO Customer No.
field.';
                ApplicationArea = all;
            }
        }
    }
}
```

Optional auxiliary objects:

```
codeunit 50101 TokenMgt
{
    procedure GetCustomerNoTok(): Text
    begin
        exit(CustomerNoTok);
    end;

    var
        CustomerNoTok: Label 'customerNo', Locked = true;
}
```

Deviating use of the SQUEEZE attachments

The following is a brief explanation of how the attachments downloaded through SQUEEZE for BC can be used for a scenario that differs from the intended use.

After creating the purchasing document from the validated SQUEEZE document, you have the option to retrieve the attachments (if they have been downloaded in advance).

```
    // Set IsHandled to true and implement your code
    // Do not forget to deactivate "Transfer attachments to target document" in the Document Class Setup
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP Document Mgt.", 'OnBeforeTransferCoreAttachmentsToStandardDocument', '', false, false)]
    local procedure DXPDocMgtOnBeforeTransferCoreAttachmentsToStandardDocument(Document: Record "DXP Document"; var IsHandled: Boolean)
    begin
        IsHandled := true;
        HandleCoreAttachmentsAccordingToYourNeeds(Document, IsHandled);
    end;

    procedure HandleCoreAttachmentsAccordingToYourNeeds(Document: Record "DXP Document"; var IsHandled: Boolean)
    var
        CoreAttachments: Record "DXP Document Attachment";
        InStr: InStream;
    begin
        CoreAttachments.SetRange("Document No.", Document."No.");
        CoreAttachments.IsEmpty() then
            exit;
        CoreAttachments.FindSet();
        repeat
            CoreAttachments.Calcfields(Attachment);
            CoreAttachments.Attachment.CreateInStream(InStr);
            // Here you can handle the file stream according to your needs
            [...]
```

```
        until CoreAttachments.Next() = 0;
    end;
```

Adding a field in the validation (from version 2.10)

The following example extension shows how developers can add fields in the validation (here: document class invoice/credit note).

Optional integration events incl. sample procedures:

```
codeunit 50100 EventSubs
{
    //
    // [If you want to perform plausibility checks on the newly added header field, this is the place]
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ P. Inv/Crdt Memo Impl.", 'OnAfterDoHeaderPlausibilityChecks', '', false, false)]
    local procedure SQZPInvCrdtMemoImplOnBeforeDoHeaderPlausibilityChecks(DocHeader: Record "DXP SQZ Document Header"; var PlausibilityCheck: Codeunit "DXP Plausiblity Check Mgt.")
    begin
        CheckCustomer(PlausibilityCheck, DocHeader."Customer No.");
    end;

    //
    // [If you want to perform plausibility checks on the newly added line field, this is the place]
    //
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ P. Inv/Crdt Memo Impl.", 'OnAfterDoLinePlausibilityChecks', '', false, false)]
    local procedure SQZPInvCrdtMemoImplOnAfterDoLinePlausibilityChecks(DocHeader: Record "DXP SQZ Document Header"; DocLine: Record "DXP SQZ Document Line"; var PlausibilityCheck: Codeunit "DXP Plausiblity Check Mgt.")
    begin
```

```
        CheckCustomer(PlausibilityCheck, DocLine."Customer No.");
    end;

    [TryFunction]
    local procedure CustomerExists(CustomerNo: Code[20])
    var
        Customer: Record Customer;
    begin
        Customer.Get(CustomerNo);
    end;

    local procedure CheckCustomer(var PlausibilityCheck: Codeunit "DXP Plausiblity Check
Mgt."; CustomerNo: Code[20]): Boolean
    begin
        if CustomerNo = '' then
            exit(false);

        if not CustomerExists(CustomerNo) then begin
            PlausibilityCheck.AddPlausibilityCheckEntry(GetLastErrorText(), Page::"Customer
List");
            exit(false);
        end;

        exit(true);
    end;
}
```

Table Extensions:

SQUEEZE 4 BC

```
tableextension 50100 "SQZ Doc. Header Ext." extends "DXP SQZ Document Header"
{
    fields
    {
        field(50100; "Customer No."; Code[20])
        {
            TableRelation = Customer;
            ValidateTableRelation = false;
            Caption = 'Customer No.';
```

```
        }
    }
}
```

```
tableextension 50101 "SQZ Doc. Line Ext." extends "DXP SQZ Document Line"
{
    fields
    {
        field(50100; "Customer No."; Code[20])
        {
            TableRelation = Customer;
            ValidateTableRelation = false;
            Caption = 'Customer No.';
        }
    }
}
```

Target document (here: Purchase Document)

```
tableextension 50102 "Purchase Header Ext." extends "Purchase Header"
{
    fields
    {
        field(50100; "DXP Customer No."; Code[20])
        {
            DataClassification = CustomerContent;
            TableRelation = Customer;
            Caption = 'DEXPRO Customer No.';
        }
    }
}
```

Page Extensions:

SQUEEZE 4 BC

```
pageextension 50100 "SQZ Document Ext." extends "DXP SQZ Document v2"
{
```

```
    layout
    {
        addafter(BuyFromVendorInternal)
        {
            field("Customer No. Internal"; Rec."Customer No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the value of the Customer No. field.';
              //The following code is optional. It is used to highlight a recognized value in
the Squeeze Viewer
                trigger OnAssistEdit()
                begin
                    MarkField(Rec.FieldNo("Customer No."), Rec."Customer No.",
Rec);
                end;

                trigger OnValidate()
                begin
                    CheckPlausibility();
                end;
            }
        }
        addafter(BuyFromVendorExternal)
        {
            field("Customer No."; Rec."Customer No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the value of the Customer No. field.';
                //The following code is optional. It is used to highlight a recognized value
in the Squeeze Viewer
                trigger OnAssistEdit()
                var
                    ApiMgt: Codeunit "DXP SQZ API Mgt.";
                    ViewerResident: Codeunit "DXP SQUEEZE Viewer Resident";
                begin
                    MarkField(Rec.FieldNo("Customer No."), Rec."Customer No.",
Rec);
                end;

                trigger OnValidate()
```

```
                begin
                    CheckPlausibility();
                end;
            }
        }
    }

    local procedure MarkField( AppFldNo: Integer; FieldVal: Variant; DocHeader: Record "DXP
SQZ Document Header")
    var
        ApiMgt: Codeunit "DXP SQZ API Mgt.";
        ViewerResident: Codeunit "DXP SQUEEZE Viewer Resident";
    begin
        ViewerResident := GetViewerResident();
        ApiMgt.MarkField( AppFldNo, FieldVal, ViewerResident, DocHeader);
    end;
}
```

```
pageextension 50101 "SQZ Document Sf. Ext." extends "DXP SQZ Document Subform"
{
    layout
    {
        addafter("No.")
        {
            field("Customer No."; Rec."Customer No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the value of the Customer No. field.';
            }
        }
    }
}
```

Target document (here: Purchase document)

```
pageextension 50102 "Purchase Invoice Ext." extends "Purchase Invoice"
{
    layout
    {
        addafter("Buy-from Vendor No.")
```

```
        {
            field("DXP Customer No."; Rec."DXP Customer No.")
            {
                ToolTip = 'Specifies the value of the DEXPRO Customer No.
field.';
                ApplicationArea = all;
            }
        }
    }
}
```

```
pageextension 50103 "Purchase Inv. Sf Ext." extends "Purch. Invoice Subform"
{
    layout
    {
        addafter("No.")
        {
            field("DXP Customer No."; Rec."DXP Customer No.")
            {
                ToolTip = 'Specifies the value of the DEXPRO Customer No.
field.';
                ApplicationArea = all;
            }
        }
    }
}
```

Alternative use of the SQUEEZE attachments

The following briefly explains how the attachments downloaded by SQUEEZE for BC can be used for a scenario that differs from the intended use.

After creating the purchase document from the validated SQUEEZE document, you have the option of accessing the attachments (assuming they have been downloaded in advance).

```
    // Set IsHandled to true and implement your code
    // Do not forget to deactivate "Transfer attachments to target document" in the Document Class Setup
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP Document Mgt.", 'OnBeforeTransferCoreAttachmentsToStandardDocument', '', false, false)]
    local procedure DXPDocMgtOnBeforeTransferCoreAttachmentsToStandardDocument(Document: Record "DXP Document"; var IsHandled: Boolean)
    begin
        IsHandled := true;
        HandleCoreAttachmentsAccordingToYourNeeds(Document, IsHandled);
    end;

    procedure HandleCoreAttachmentsAccordingToYourNeeds(Document: Record "DXP Document"; var IsHandled: Boolean)
    var
        CoreAttachments: Record "DXP Document Attachment";
        InStr: InStream;
    begin
        CoreAttachments.SetRange("Document No.", Document."No.");
        CoreAttachments.IsEmpty() then
        exit;
        CoreAttachments.FindSet();
        repeat
            CoreAttachments.Calcfields(Attachment);
            CoreAttachments.Attachment.CreateInStream(InStr);
            // Here you can handle the file stream according to your needs
        [...]
```

```
        until CoreAttachments.Next() = 0;
    end;
```

Modify Squeeze origin attachment

Overview

The `OnAfterSetOriginFileNameOnBeforeModifyDocumentAttachment` integration event allows third-party developers to customize document attachment records before they are modified during the SQUEEZE attachment processing workflow. An origin attachment is the relevant attachment that leads to the creation of a document (e.g. the vendor´s invoice).

Event Declaration

```
[IntegrationEvent(false, false)]
local procedure OnAfterSetOriginFileNameOnBeforeModifyDocumentAttachment(var
DocumentAttachment: Record "DXP Document Attachment")
begin
end;
```

Event Parameters

Parameter	Type	Description
`DocumentAttachment`	`Record "DXP Document Attachment"` (var)	The document attachment record that is about to be modified. Passed by reference, allowing modifications.

When This Event is Triggered

This event is fired during the `SaveAttachments` procedure when:

1. An attachment is being processed from SQUEEZE
2. The attachment is identified as an origin file (`IsOriginFile = true`)
3. The system has set the origin file name and `DXP Is Origin File` flag
4. Just before the `DocumentAttachment.Modify(true)` call

Use Cases

This integration event is useful for:

- **Custom Field Population**: Setting additional custom fields on the document attachment
- **File Name Transformation**: Applying custom naming conventions or formatting
- **Metadata Enhancement**: Adding custom metadata or tags to attachments
- **Validation Logic**: Implementing custom validation before the record is saved
- **Integration Requirements**: Preparing data for external system integrations

Implementation Example

Change The Filename Before Modify

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ API Mgt.",
'OnAfterSetOriginFileNameOnBeforeModifyDocumentAttachment', '', false, false)]
local procedure OnAfterSetOriginFileNameOnBeforeModifyDocumentAttachment(var
DocumentAttachment: Record "DXP Document Attachment")
var
    CoreDocument: Record "DXP Document";
    SQZDocumentHeader: Record "DXP SQZ Document Header";
    Vendor: Record Vendor;
    NewFileName: Text[250];
    VendorPrefix: Text[50];
begin
    // Get the core document using the document attachment's Document No.
    if not CoreDocument.Get(DocumentAttachment."Document No.") then
        exit;
```

```
    // Check if the core document has a linked SQUEEZE document
    if IsNullGuid(CoreDocument."Linked-to Record Id") then
        exit;

    // Get the SQZ Document Header using the SystemId from Linked-to Record Id
    if not GetSQZDocumentHeaderFromRecordId(CoreDocument."Linked-to Record Id",
SQZDocumentHeader) then
        exit;

    // Check if Buy-from Vendor No. is populated
    if SQZDocumentHeader."Buy-from Vendor No." = '' then
        exit;

    // Get the vendor record
    if not Vendor.Get(SQZDocumentHeader."Buy-from Vendor No.") then
        exit;

    // Create vendor prefix from Search Name (fallback to Name if Search Name is empty)
    if Vendor."Search Name" <> '' then
        VendorPrefix := Vendor."Search Name"
    else
        VendorPrefix := Vendor.Name;

    // Clean the vendor prefix (remove invalid filename characters and limit length)
    VendorPrefix := CleanFilenameText(VendorPrefix, 30);

    // Check if vendor prefix already exists in filename to avoid duplicates
    if DocumentAttachment."File Name".StartsWith(VendorPrefix + '_') then
        exit;

    // Create new filename with vendor prefix
    NewFileName := VendorPrefix + '_' + DocumentAttachment."File Name";

    // Update the attachment filename
    DocumentAttachment."File Name" := NewFileName;
end;

local procedure GetSQZDocumentHeaderFromRecordId(LinkedRecordId: Guid; var SQZDocumentHeader:
Record "DXP SQZ Document Header"): Boolean
```

```
var
    RecRef: RecordRef;
    SystemIdFieldRef: FieldRef;
begin
    // Method 1: Try to get the record directly if LinkedRecordId is actually a SystemId
    if SQZDocumentHeader.GetBySystemId(LinkedRecordId) then
        exit(true);

    // Method 2: If that fails, we need to find the record another way
    // This assumes the Linked-to Record Id might be stored differently
    SQZDocumentHeader.Reset();
    SQZDocumentHeader.SetRange(SystemId, LinkedRecordId);
    exit(SQZDocumentHeader.FindFirst());
end;

local procedure CleanFilenameText(InputText: Text; MaxLength: Integer): Text
var
    CleanText: Text;
begin
    // Remove invalid filename characters
    CleanText := DelChr(InputText, '=', '<>|"/\:*?');

    // Replace spaces with underscores for better filename compatibility
    CleanText := CleanText.Replace(' ', '_');

    // Limit length
    CleanText := CopyStr(CleanText, 1, MaxLength);

    exit(CleanText);
end;
```

Important Considerations

Data Integrity

- The `DocumentAttachment` record is passed by reference, so any changes will be persisted

- Consider field length limitations when modifying text fields

Performance

- Keep processing lightweight as this event is called for each origin file attachment
- Consider caching frequently accessed data

Error Handling

- Implement proper error handling to prevent the attachment save process from failing
- Use try-functions for risky operations

Related Events

- `OnAfterSaveAttachment`: Triggered after each attachment is completely saved
- Consider using this alternative event if you need to perform actions after the record is saved

Troubleshooting

If your event subscriber isn’t being triggered:

1. Test with origin files specifically (non-origin files won’t trigger this event)