# Extension Entwicklung

In diesem Kapitel werden wichtige Integration Events und Beispiele aufgeführt, die bei der Entwicklung einer Extension für SQUEEZE for BC helfen. Wichtig: Anpassungen dürfen nur durch entsprechend geschulte Berater/Entwickler durchgeführt werden. Diese befinden sich außerdem immer außerhalb des Standard-Supports.

# Hinzufügen eines Feldes in der Validierung (bis Squeeze BC APP Version 1.*)

Im Folgenden wird anhand einer Beispielextension dargestellt, wie man als Entwickler Felder in der Validierung hinzufügen kann.

## Benötigte Integration Events inkl. beispielhafter Prozeduren:

```
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.';
        }
    }
}
```

## Zielbeleg (hier: Einkaufsbeleg)

```
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.';
            }
        }
    }
}
```

## Zielbeleg (hier: Einkaufsbeleg)

```
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;
            }
        }
    }
}
```

## Optionale Hilfsobjekte:

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

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

# Hinzufügen eines Feldes in der Validierung (ab Version 2.0)

Im Folgenden wird anhand einer Beispielextension dargestellt, wie man als Entwickler Felder in der Validierung hinzufügen kann (hier: Dokumentenklasse Rechnung/Gutschrift).

## Benötigte Integration Events inkl. beispielhafter Prozeduren:

```
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 P. Inv/Crdt Memo Impl.", 'OnBeforeAddLineToDocumentJObj', '', false, false)]
    local procedure SQZPInvCrdtMemoImplOnBeforeAddLineToDocumentJObj(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 P. Inv/Crdt Memo Impl.", 'OnBeforeAddLineJObjToLineJArray', '', false, false)]
    local procedure SQZPInvCrdtMemoImplOnBeforeAddLineJObjToLineJArray(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 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;

    //
    // 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 also 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
```

```
        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.';
```

```
        }
    }
}
```

## Zielbeleg (hier: Einkaufsbeleg)

```
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.';
                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.';
                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.';
            }
        }
    }
}
```

## Zielbeleg (hier: Einkaufsbeleg)

```
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;
            }
        }
    }
}
```

## Optionale Hilfsobjekte:

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

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

# Hinzufügen eines Feldes in der Validierung (ab Version 2.10)

Im Folgenden wird anhand einer Beispielextension dargestellt, wie man als Entwickler Felder in der Validierung hinzufügen kann (hier: Dokumentenklasse Rechnung/Gutschrift).

## Optionale Integration Events inkl. beispielhafter Prozeduren:

```
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.';
        }
    }
}
```

## Zielbeleg (hier: Einkaufsbeleg)

```
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.';
            }
        }
    }
}
```

## Zielbeleg (hier: Einkaufsbeleg)

```
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;
            }
        }
    }
}
```

# Abweichende Verwendung der SQUEEZE Anhänge

Im Folgenden wird kurz erläutert, wie die durch SQUEEZE for BC heruntergeladenen Anhänge für ein Szenario verwendet werden können, das von der vorgesehenen Verwendung abweicht.

Nach der Erstellung des Einkaufsbeleges aus dem validierten SQUEEZE Beleg haben Sie die Möglichkeit, die Anhänge abzugreifen (sofern diese im Vorfeld heruntergeladen wurden).

```
    // 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;
```

# Implementierung eines benutzerdefinierten, automatischen Bestellabgleichs

## Einführung

Der automatische Bestellabgleich ist ein wichtiger Bestandteil der Dokumentenverarbeitung in Squeeze for Business Central. Im Standard vergleicht das System eingehende Belege mit vorhandenen Bestellungen und Wareneingängen. Diese Dokumentation zeigt Ihnen, wie Sie diesen Abgleichsprozess um Ihre eigenen Belegarten erweitern können.

Die Standardimplementierung verwendet einen dreistufigen Prozess:

1. Sammeln relevanter Belegnummern basierend auf Geschäftsregeln
2. Für jede gefundene Belegnummer wird ein detaillierter Abgleich durchgeführt
3. Im Anschluss werden die von Squeeze ausgelesenen Positionen um die gefundenen Daten angereichert

Diesen bewährten Ansatz werden wir in der folgenden, beispielhaften Implementierung beibehalten.

## Integrationspunkt

Der zentrale Integrationspunkt ist das `OnBeforePerformAutomaticOrdermatch`-Event in Codeunit 70954632 "DXP SQZ Document Mgt.". Dieses Event wird nach der Erstellung von Kopf- und Positionsdaten aufgerufen.

```
[IntegrationEvent(false, false)]
local procedure OnBeforePerformAutomaticOrdermatch(
```

```
    DocHeader: Record "DXP SQZ Document Header";
    var OrderNoList: List of [Code[20]];
    var IsHandled: Boolean)
```

# Parameter

- `DocHeader`: Der Dokumentenkopf mit den zu vergleichenden Daten
- `OrderNoList`: Eine Liste von Belegnummern für den Abgleich
- `IsHandled`: Steuert, ob die Standardverarbeitung übersprungen werden soll

# Implementierungsbeispiel

# Technische Umsetzung

## 1. Erweiterung des Dokumententyp-Enums

Zunächst erweitern wir die möglichen Dokumententypen um unseren eigenen Typ:

```
enumextension 50100 "Custom Order Match Doc. Type" extends "DXP Order Match Document Type"
{
    value(50000; "Custom")
    {
        Caption = 'Custom Document';
    }
}
```

## 2. Implementierung der Abgleichslogik

Der zentrale Punkt unserer Implementierung ist eine Codeunit, die den Abgleichsprozess steuert. Integrationspunkt ist :

```
//codeunit 50100 "Custom Document Matching"
```

```
//{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"DXP SQZ Document Mgt.",
'OnBeforePerformAutomaticOrdermatch', '', false, false)]
    local procedure OnBeforePerformAutomaticOrdermatch(
        DocHeader: Record "DXP SQZ Document Header";
        var OrderNoList: List of [Code[20]];
        var IsHandled: Boolean)
    var
        CustomSourceDoc: Record "Custom Source Document";
        CustomDocNo: Code[20];
    begin
        // Take control of the matching process and prevent standard processing
        IsHandled := true;

        // Find potential matching documents
        CustomSourceDoc.SetRange("Vendor No.", DocHeader."Buy-from Vendor No.");
        // Add your specific document status or type filters
        CustomSourceDoc.SetRange("Document Type", CustomSourceDoc."Document
Type"::Order);
        CustomSourceDoc.SetRange(Status, CustomSourceDoc.Status::Released);

        // Add matching document numbers to the list
        if CustomSourceDoc.FindSet() then
            repeat
                CustomDocNo := CustomSourceDoc."No.";
                if IsDocumentEligibleForMatching(CustomSourceDoc, DocHeader)
then
                    if not OrderNoList.Contains(CustomDocNo) then
                        OrderNoList.Add(CustomDocNo);
            until CustomSourceDoc.Next() = 0;

        // Process all collected documents
        ProcessMatchingDocuments(DocHeader, OrderNoList);
    end;

    local procedure IsDocumentEligibleForMatching(
        CustomSourceDoc: Record "Custom Source Document";
        DocHeader: Record "DXP SQZ Document Header"): Boolean
    begin
```

```
        // Implement business rules for document selection
        // For example:
        if CustomSourceDoc."Document Date" > DocHeader."Document Date" then
            exit(false);

        if CustomSourceDoc."Currency Code" <> DocHeader."Currency Code" then
            exit(false);

        // Check for open lines that can be matched
        if not HasOpenLinesToMatch(CustomSourceDoc) then
            exit(false);

        exit(true);
    end;

    local procedure HasOpenLinesToMatch(CustomSourceDoc: Record "Custom Source Document"):
Boolean
    var
        CustomSourceLine: Record "Custom Source Line";
    begin
        CustomSourceLine.SetRange("Document No.", CustomSourceDoc."No.");
        CustomSourceLine.SetFilter("Outstanding Quantity", '>0');
        exit(not CustomSourceLine.IsEmpty());
    end;
//}
```

# 3. Verarbeitung der gefundenen Belege

Nach dem Sammeln der relevanten Belegnummern erfolgt der eigentliche Abgleich:

```
local procedure ProcessMatchingDocuments(
    DocHeader: Record "DXP SQZ Document Header";
    OrderNoList: List of [Code[20]])
var
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary;
    OrderNo: Code[20];
begin
    foreach OrderNo in OrderNoList do begin
```

```
        // Get matching entries for current document
        GetMatchEntries(OrderNo, TempMatchEntry, DocHeader);

        if not TempMatchEntry.IsEmpty() then
            // Try to find and assign matching lines
            TryMatchDocumentLines(TempMatchEntry, DocHeader);
    end;
end;

local procedure GetMatchEntries(
    DocumentNo: Code[20];
    var TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary;
    DocHeader: Record "DXP SQZ Document Header")
var
    CustomSourceLine: Record "Custom Source Line";
begin
    TempMatchEntry.Reset();
    TempMatchEntry.DeleteAll();

    // Get lines from custom document
    CustomSourceLine.SetRange("Document No.", DocumentNo);

    if CustomSourceLine.FindSet() then
        repeat
            // Create match entry for each relevant line
            CreateMatchEntry(CustomSourceLine, TempMatchEntry);
        until CustomSourceLine.Next() = 0;
end;

local procedure CreateMatchEntry(
    CustomSourceLine: Record "Custom Source Line";
    var TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary)
begin
    TempMatchEntry.Init();
    TempMatchEntry."Document Type" := "DXP Order Match Document Type"::Custom;
    TempMatchEntry."Document No." := CustomSourceLine."Document No.";
    TempMatchEntry."Document Line No." := CustomSourceLine."Line No.";
    TempMatchEntry."No." := CustomSourceLine."Item No.";
    TempMatchEntry.Quantity := CustomSourceLine.Quantity;
```

```
    TempMatchEntry."Direct Unit Cost" := CustomSourceLine."Unit Price";
    TempMatchEntry."Line Amount" := CustomSourceLine.Amount;
    TempMatchEntry.Insert();
end;
```

# Die Matching-Logik im Detail

Der eigentliche Abgleich der Belegzeilen erfolgt nach definierten Geschäftsregeln:

```
local procedure TryMatchDocumentLines(
    var TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary;
    DocHeader: Record "DXP SQZ Document Header")
var
    DocLine: Record "DXP SQZ Document Line";
    MatchSetup: Record "Custom Match Setup";
    HasCustomSetup: Boolean;
begin
    // Get configuration
    MatchSetup.Get();

    // Process each potential match entry
    if TempMatchEntry.FindSet() then
        repeat
            DocLine.Reset();
            DocLine.SetRange("Document No.", DocHeader."No.");
            DocLine.SetRange("Buy-from Vendor No.", TempMatchEntry."Buy-from Vendor
No.");
            DocLine.SetFilter("Allocated Document Line No.", '%1', 0); // Only unallocated
lines

            // Try matching strategies in order of precision
            if TryExactMatch(DocLine, TempMatchEntry) then
                CheckTolerancesAndAllocate(TempMatchEntry, DocLine, MatchSetup)
            else
                if TryItemReferenceMatch(DocLine, TempMatchEntry) then
                    CheckTolerancesAndAllocate(TempMatchEntry, DocLine,
MatchSetup)
                else
```

```
if TryDescriptionMatch(DocLine, TempMatchEntry) then
                        CheckTolerancesAndAllocate(TempMatchEntry, DocLine,
MatchSetup)
                    else
                        if TryBasicValuesMatch(DocLine, TempMatchEntry)
then
                            CheckTolerancesAndAllocate(TempMatchEntry, DocLine,
MatchSetup);
        until TempMatchEntry.Next() = 0;
end;

local procedure TryExactMatch(
    var DocLine: Record "DXP SQZ Document Line";
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary): Boolean
begin
    DocLine.SetRange(Type, TempMatchEntry.Type);
    DocLine.SetFilter("No.", '%1|%2', TempMatchEntry."No.", TempMatchEntry."Item Reference
No.");
    DocLine.SetRange("SQZ Quantity", TempMatchEntry.Quantity);

    exit(not DocLine.IsEmpty());
end;

local procedure TryItemReferenceMatch(
    var DocLine: Record "DXP SQZ Document Line";
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary): Boolean
begin
    DocLine.SetRange(Type);  // Clear previous filters
    DocLine.SetRange("Item Reference No.", TempMatchEntry."Item Reference No.");

    exit(not DocLine.IsEmpty());
end;

local procedure TryDescriptionMatch(
    var DocLine: Record "DXP SQZ Document Line";
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary): Boolean
begin
    DocLine.SetRange("Item Reference No."); // Clear previous filter
    DocLine.SetFilter("SQZ Description", '@*' + TempMatchEntry.Description + '*');
```

```
    exit(not DocLine.IsEmpty());
end;

local procedure TryBasicValuesMatch(
    var DocLine: Record "DXP SQZ Document Line";
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary): Boolean
begin
    DocLine.SetRange("SQZ Description");  // Clear previous filter
    DocLine.SetRange("SQZ Quantity", TempMatchEntry.Quantity);
    DocLine.SetRange("SQZ Unit Price", TempMatchEntry."Direct Unit Cost");

    exit(not DocLine.IsEmpty());
end;

local procedure CheckTolerancesAndAllocate(
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary;
    var DocLine: Record "DXP SQZ Document Line";
    MatchSetup: Record "Custom Match Setup")
begin
    DocLine.FindFirst();  // We know there is at least one line

    if IsMatchWithinTolerances(DocLine, TempMatchEntry, MatchSetup) then
        AssignMatchData(DocLine, TempMatchEntry);
end;

local procedure IsMatchWithinTolerances(
    DocLine: Record "DXP SQZ Document Line";
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary;
    MatchSetup: Record "Custom Match Setup"): Boolean
var
    QtyTolerance: Decimal;
    AmountTolerance: Decimal;
begin
    QtyTolerance := MatchSetup."Quantity Tolerance";
    AmountTolerance := MatchSetup."Amount Tolerance";

    if Abs(DocLine."SQZ Quantity" - TempMatchEntry.Quantity) > QtyTolerance then
        exit(false);

    if Abs(DocLine."SQZ Unit Price" - TempMatchEntry."Direct Unit Cost") > AmountTolerance
```

```
then
        exit(false);

    exit(true);
end;

local procedure AssignMatchData(
    var DocLine: Record "DXP SQZ Document Line";
    TempMatchEntry: Record "DXP SQZ Order Match Entry" temporary)
begin
    // Assign the matched data to the document line
    DocLine.Validate("Allocated Document Type", "DXP Order Match Document Type"::Custom);
    DocLine.Validate("Allocated Document No.", TempMatchEntry."Document No.");
    DocLine.Validate("Allocated Document Line No.", TempMatchEntry."Document Line No.");
    DocLine.Validate("Allocated Quantity", TempMatchEntry.Quantity);
    DocLine.Validate("Allocated Unit Price", TempMatchEntry."Direct Unit Cost");
    DocLine.Validate("Allocated Line Amount", TempMatchEntry."Line Amount");
    DocLine.Validate("Allocated Line Discount %", TempMatchEntry."Line Discount %");
    DocLine.Modify(true);
end;
```

Bei Fragen zur Implementierung stehen wir Ihnen gerne zur Verfügung.

# Squeeze-Ursprungsanhang anpassen

# Übersicht

Das `OnAfterSetOriginFileNameOnBeforeModifyDocumentAttachment` Integration Event ermöglicht es Drittanbieter-Entwicklern, Dokumentenanhang-Datensätze anzupassen, bevor sie während des SQUEEZE-Anhang-Verarbeitungsworkflows modifiziert werden. Ein Ursprungsanhang ist der relevante Anhang, der zur Erstellung eines Belegs führt (z.B. die Rechnung des Lieferanten).

# Event-Deklaration

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

# Event-Parameter

| Parameter | Typ | Beschreibung |
| --- | --- | --- |
| DocumentAttachment | `Record "DXP Document Attachment"` (var) | Der Dokumentenanhang-Datensatz, der modifiziert werden soll. Als Referenz übergeben, wodurch Änderungen möglich sind. |

# Wann wird dieses Event ausgelöst?

Dieses Event wird während der `SaveAttachments`-Prozedur ausgelöst, wenn:

- Ein Anhang von SQUEEZE verarbeitet wird
- Der Anhang als Ursprungsdatei identifiziert wird (`IsOriginFile = true`)
- Das System den Ursprungsdateinamen und das `DXP Is Origin File` Flag gesetzt hat
- Kurz vor dem `DocumentAttachment.Modify(true)`-Aufruf

# Anwendungsfälle

Dieses Integration Event ist nützlich für:

- **Benutzerdefiniertes Füllen von Feldern**: Setzen zusätzlicher benutzerdefinierter Felder im Dokumentenanhang
- **Dateinamen-Transformation**: Anwenden benutzerdefinierter Namenskonventionen oder Formatierung
- **Metadaten-Erweiterung**: Hinzufügen benutzerdefinierter Metadaten oder Tags zu Anhängen
- **Validierungslogik**: Implementierung benutzerdefinierter Validierung vor dem Speichern des Datensatzes
- **Integrationsanforderungen**: Vorbereitung von Daten für externe Systemintegrationen

# Implementierungsbeispiel

## Dateinamen vor Modifikation ändern

```
[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;
```

# Wichtige Überlegungen

## Datenintegrität

- Der `DocumentAttachment`-Datensatz wird als Referenz übergeben, daher werden alle

Änderungen gespeichert
- Berücksichtigen Sie Feldlängenbegrenzungen beim Ändern von Textfeldern

# Performance

- Halten Sie die Verarbeitung schlank, da dieses Event für jeden Ursprungsdatei-Anhang aufgerufen wird
- Erwägen Sie das Zwischenspeichern häufig verwendeter Daten

# Fehlerbehandlung

- Implementieren Sie ordnungsgemäße Fehlerbehandlung, um zu verhindern, dass der Anhang-Speichervorgang fehlschlägt
- Verwenden Sie try-Funktionen für riskante Operationen

# Verwandte Events

- `OnAfterSaveAttachment`: Wird ausgelöst, nachdem jeder Anhang vollständig gespeichert wurde.
- Erwägen Sie die Verwendung dieses alternativen Events, wenn Sie Aktionen nach dem Speichern des Datensatzes durchführen müssen.

# Fehlerbehebung

Wenn Ihr Event Subscriber nicht ausgelöst wird:

- Testen Sie speziell mit Ursprungsdateien (Nicht-Ursprungsdateien lösen dieses Event nicht aus)