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)

Revision #4
Created 28 May 2025 07:18:51 by Bernd Feddersen
Updated 28 May 2025 09:58:02 by Bernd Feddersen