Implementierung einer neuen, generischen Dokumentenklasse

SQUEEZE ? Core ? Ihre Verarbeitung

Dieser Leitfaden ist bewusst fokussiert und pipeline-orientiert: Er zeigt, wie Belegdaten aus **SQUEEZE** heruntergeladen, in Warteschlangen verarbeitet und in **DEXPRO Core** persistiert werden – und an welcher Stelle sich eine Drittanbieter-Extension einklinkt.

Begriff: In diesem Leitfaden meint „SQUEEZE-App“ die Business-Central-Erweiterung, die mit der SQUEEZE API verbunden ist und den Import nach DEXPRO Core steuert.

Annahmen:

- Drittanbieter-Entwickler arbeiten typischerweise mit **veröffentlichten Symbolen**, nicht mit dem gesamten Quellcode.
- Die SQUEEZE-App übernimmt **Herunterladen + Importwarteschlange + Erstellung des Core-Belegs sowie die rudimentäre Erstellung des SQUEEZE-Belegs unter Berücksichtigung der Feldzuordnung**, sobald die Einrichtung abgeschlossen ist.
- Ihre Extension übernimmt die Fachlogik, nachdem das Core-Dokument existiert: Quell-Datensätze erstellen bzw. erweitern, validieren, „Processed JSON“ erzeugen und optional ein eigenes Zielobjekt oder einen eigenen Beleg erstellen.

---

1) Der Ende?zu?Ende?Ablauf SQUEEZE ? Core

Sobald Dokumentklasse und Einrichtung konfiguriert sind, läuft die Importpipeline der SQUEEZE-App typischerweise so ab:

1. `DXP Document Class` **erweitern** und Ihre Implementierung binden: `DXP ISource Document`.
2. **Dokumentenklassen-Einrichtung** für diese Klasse konfigurieren.
   - Beim Validieren von `DXP SQZ Document Class ID` holt die SQUEEZE-App automatisch die benötigten Einrichtungswerte aus SQUEEZE (z. B. `DXP SQZ Export Interface ID` und `DXP SQZ Line Table ID`) und füllt sie.

3. Der Job der SQUEEZE-App ruft `DXP SQZ API Mgt.DownloadDocumentData()` auf. Auch via "Belege herunterladen" aufrufbar
   - Dabei werden neue Einträge in `DXP SQZ Import Queue` angelegt (Dokument-ID + Zielmandant).
4. Die SQUEEZE-App ruft `DXP SQZ Import Queue Mgt.TransferImportQueueEntries()` auf.
   - Dabei wird die Orchestrierungs-Codeunit `DXP Sqz Create Document` für die Einträge in der Importwarteschlange ausgeführt.
5. `DXP SQZ API Mgt.CreateCoreAndSQUEEZEDocument(DocClass, DocumentId)` lädt das vollständige JSON.
   - Es wird `DXP Document Mgt.AddDocument(DocClass, DocumentId, RawJson, …)` aufgerufen.
   - Ergebnis: Ein `DXP Document`-Datensatz existiert. Status = `Imported`, `JSON Raw` ist gefüllt.
6. Die SQUEEZE-App ruft Ihre Implementierung `ISourceDocument.CreateSource(Document)` auf.
   - Ab hier beginnt Ihre Implementierung.

Nach Schritt 6 haben Sie:

- Ein Core-Container-Dokument (`DXP Document`).
- Ein SQUEEZE-Dokument auf Seite der SQUEEZE-App.
- Ihre Source-Datensätze sollten existieren, sobald `CreateSource` erfolgreich abgeschlossen ist.

---

2) Was Sie implementieren (Interfaces)

Quell?Vertrag, also Squeeze Seite: `DXP ISource Document`

Das implementieren Sie, um:

- Ihre editierbaren Source-Datensätze aus `DXP Document.JSON Raw` zu erstellen
- Source-Datensätze zu validieren
- „Processed JSON“ zu erzeugen und an Core-Processing zu übergeben

Ziel?Vertrag, also Zielbeleg: `DXP IDocument Processing`

Das implementieren Sie, wenn Core aus dem „Processed JSON“ **einen eigenen Ziel-Datensatz** (Ihre eigene Tabelle / Ihr eigenes Dokument) erstellen/aktualisieren soll.

---

3) Schritt 1 — Dokumentklasse definieren und Quell?Implementierung zuweisen

Erstellen Sie eine Enum-Extension und binden Sie Ihre Source-Implementierung:

```
enumextension 50100 "MY Document Class Ext." extends "DXP Document Class"
{
    value(50100; "MY Generic Document")
    {
        Caption = 'My Generic Document';
        Implementation = "DXP ISource Document" = "MY Source Document Impl.";
    }
}
```

---

4) Schritt 2 — Dokumentenklassen? Einrichtung konfigurieren (damit die Importwarteschlange gefüllt wird)

Konfigurieren Sie in der **Dokumentenklassen-Einrichtung** für Ihre neue Dokumentklasse die SQUEEZE-spezifischen Felder.

Wichtig:

- Das Validieren von `DXP SQZ Document Class ID` triggert das automatische Laden der erforderlichen Konfiguration aus SQUEEZE (z. B. `DXP SQZ Export Interface ID` und `DXP SQZ Line Table ID`).
- Falls der Abruf fehlschlägt, werden diese API-abgeleiteten Felder wieder geleert.

Ergebnis: Die SQUEEZE-App kann `DownloadDocumentData()` ausführen und Einträge in `DXP SQZ Import Queue` für Ihre Klasse anlegen.

---

5) Schritt 3 — Feldzuordnung sicherstellen (heruntergeladene Zuordnung + Ihre Standardwerte)

Die SQUEEZE-App kann den Feldkatalog des angeschlossenen Squeeze Mandanten herunterladen und in Core-Mapping-Tabellen persistieren (pro Dokumentklasse).

Ihre Rolle als Dokumentklassen-Implementierer:

- Standard-Feldzuordnung „Squeeze Field Name → AL Field No.“ via `InitFieldMapping(...)` bereitstellen.
- Alternativ Felder über die Zuordnungs-Pages mappen (Benutzerdefinierte Feldzuordnung).

5.1 `InitFieldMapping` implementieren (eigenständiges Beispiel)

```
codeunit 50101 "MY Source Document Impl." implements "DXP ISource Document"
{
    procedure InitFieldMapping(var HeaderMapping: Dictionary of [Text, Integer]; var
LineMapping: Dictionary of [Text, Integer])
    var
        Header: Record "DXP SQZ Document Header";
        Line: Record "DXP SQZ Document Line";
    begin
        Clear(HeaderMapping);
        Clear(LineMapping);

        // Header fields
        HeaderMapping.Add('vendorNo', Header.FieldNo("Buy-from Vendor No."));
        HeaderMapping.Add('documentDate', Header.FieldNo("Document Date"));
        HeaderMapping.Add('postingDate', Header.FieldNo("Posting Date"));
        HeaderMapping.Add('reference', Header.FieldNo("Document Reference"));

        // Line fields
        LineMapping.Add('description', Line.FieldNo(Description));
        LineMapping.Add('quantity', Line.FieldNo(Quantity));
        LineMapping.Add('unitPrice', Line.FieldNo("Direct Unit Cost"));
```

```
    end;

    // other interface methods omitted here...
}
```

5.2 Validierungs?Schalter beim Zuweisen von Feldern: App?Feld validieren

Wenn die SQUEEZE-App oder Ihre Quell-Erstelllogik JSON-Werte in „App-/Quellfelder“ schreibt, können Sie steuern, ob der **OnValidate-Trigger** ausgeführt wird.

In `DXP Custom Field Mapping`:

- `Validate App Field` = App-Feld während des Assignments validieren.
- `Validate` (Ziel) = Validierung beim Transfer in das Ziel-Feld ausführen.

Nutzen Sie das z. B. wenn Erweiterungsfelder Business-Logik in `OnValidate` haben oder wenn Sie bewusst trigger-frei (performanter) zuweisen wollen.

---

6) Schritt 4 — Source?Datensätze in `CreateSource` erstellen

`CreateSource(Document)` wird von der SQUEEZE-App unmittelbar nach `DXP Document Mgt.AddDocument(...)` aufgerufen, sobald der Core-Container-Datensatz existiert.

Minimale Verantwortlichkeiten:

1. `DXP Document.JSON Raw` lesen.
2. SQUEEZE-Source-Header/Lines erstellen (`DXP SQZ Document Header` / `DXP SQZ Document Line`).
3. Core-Dokument mit dem SQUEEZE-Source-Header via `UpdateDocumentAfterTransfer` verknüpfen.

Optional (empfohlen, wenn Sie Gleichwertigkeit der Funktionen zu den Standard-SQUEEZE-Dokumentklassen möchten): 4) Anhänge herunterladen, wenn `DXP Download Attachments` aktiv ist. 5) Automatische Validierung ausführen, wenn für das Dokument aktiviert.

6.1 Beispiel: SQUEEZE?Header + Lines mit den generischen Helpern erstellen

Was die Helper machen (damit Sie es nicht doppelt implementieren):

- `SqzDocumentMgt.CreateSQUEEZEDocHeader(Document, DocHeader, RawJson)`
  - Erstellt genau einen Datensatz in `DXP SQZ Document Header` für das Core-Dokument (`DXP Document`).
  - Setzt Identifikationsfelder wie `Core Document No.` und die SQUEEZE-`API Document ID`.
  - Weist Header-Felder aus JSON anhand der konfigurierten Zuordnungen zu:
    - `DXP Field Mapping` (Feldzuordnung)
    - `DXP Custom Field Mapping` (Benutzerdefinierte Feldzuordnung)
  - Respektiert `DXP Custom Field Mapping.Validate App Field` und entscheidet damit, ob beim Zuweisen der **OnValidate-Trigger** ausgeführt wird.
  - Führt das Integration-Event `OnBeforeModifySQUEEZEDocumentHeader(...)` als Extension-Hook aus.
  - Für die eingebauten Standard-SQUEEZE-Klassen wird zusätzlich Standard-Post-Processing ausgeführt (z. B. alternative IBAN/USt-ID-Ermittlung, Lieferantenaktualisierung, Standard-Buchungsdatum, Duplikatsprüfung).
- `SqzDocumentMgt.CreateSQUEEZEDocLine(DocHeader, RawJson)`
  - Erstellt die `DXP SQZ Document Line`-Datensätze basierend auf den JSON-Rows.
  - Weist Line-Felder nach dem gleichen Mapping-Prinzip zu (Standard + Custom; optional mit Validierungstriggern).
  - Befüllt `Core Document No.` auf den Lines.
  - Für Standard-SQUEEZE-Klassen wird zusätzlich Standard-Line-Post-Processing angewendet (z. B. Übertragung erkannter Werte).
  - Persistiert line-bezogene Hilfsdaten (z. B. Koordinaten/Metadaten, die UI und Matching-Logik nutzen).

```
procedure CreateSource(Document: Record "DXP Document"): Boolean
var
    DocClassSetup: Record "DXP Document Class Setup";
    AutoValidationMgt: Codeunit "DXP Automatic Validation Mgt.";
    JsonHelper: Codeunit "DXP Json Helper";
    DocMgt: Codeunit "DXP Document Mgt.";
    SqzDocumentMgt: Codeunit "DXP SQZ Document Mgt.";
    SqzApiMgt: Codeunit "DXP SQZ API Mgt.";
    RawJson: JsonObject;
    DocHeader: Record "DXP SQZ Document Header";
    AutomaticValidation: Boolean;
begin
    if (not Document."JSON Raw".HasValue()) or (Document.Status <> Document.Status::Imported)
```

```
then
        exit(false);

    // 1) Unverarbeitetes JSON lesen
    RawJson := JsonHelper.JObjectFromBlob(Document.RecordId(), Document.FieldNo("JSON Raw"));

    // 2) SQUEEZE-Source-Dokument mit den generischen Hilfsfunktionen erstellen
    //    (Feldzuordnung + optionales Validierungs-Trigger-Verhalten wird im Helper behandelt)
    SqzDocumentMgt.CreateSQUEEZEDocHeader(Document, DocHeader, RawJson);
    SqzDocumentMgt.CreateSQUEEZEDocLine(DocHeader, RawJson);

    // 3) Core → SQUEEZE-Source-Header verknüpfen
    DocMgt.UpdateDocumentAfterTransfer(Document."No.", Document.Status::Transferred,
DocHeader."API Document ID", DocHeader.RecordId());

    // 4) Optional: Anhänge herunterladen und speichern
    //    (gesteuert über die Dokumentenklassen-Einrichtung)
    if DocClassSetup.Get(DocHeader."Document Class") then
        if DocClassSetup."DXP Download Attachments" then
            SqzApiMgt.DownloadAndSaveAttachments(DocHeader);

    // 5) Optional: Automatische Validierung
    //    Wenn aktiviert, führt die Validierung die Plausibilitätsprüfungen aus und erstellt
das „Processed JSON“.
    AutomaticValidation := AutoValidationMgt.AutomaticValidationActivated(DocHeader."No.");
    if AutomaticValidation then
        SqzDocumentMgt.ValidateSQUEEZEDocument(DocHeader, AutomaticValidation);

    exit(true);
end;
```

---

7) Schritt 5 — Validierung + Übergabe „Processed JSON“

7.1 Validierung: `IsSourceDataPlausible`

Nutzen Sie das Core-Plausibility-Framework, damit Fehler konsistent angezeigt werden:

```
procedure IsSourceDataPlausible(Source: Variant; var TempPlausibilityCheckEntry: Record "DXP Plausibility Check Entry" temporary): Boolean
var
    PlausibilityCheck: Codeunit "DXP Plausiblity Check Mgt.";
    Header: Record "DXP SQZ Document Header";
begin
    if not Source.IsRecord then
        exit(false);

    Header := Source;
    TempPlausibilityCheckEntry.DeleteAll();

    PlausibilityCheck.FieldIsEmpty(Header, Header.FieldNo("Buy-from Vendor No."), 0, false, false);
    PlausibilityCheck.CheckPostingDate(Header."Posting Date", Header.FieldNo("Posting Date"));
    PlausibilityCheck.GetPlausibilityCheckEntries(TempPlausibilityCheckEntry);

    exit(TempPlausibilityCheckEntry.IsEmpty());
end;
```

7.2 „Processed JSON“ bauen und an Core übergeben: `CreateProcessedJsonFromSource`

„Processed JSON“ wird von Core in `DXP Document.JSON Processed` gespeichert und an Ihre Zielverarbeitung übergeben.

```
procedure CreateProcessedJsonFromSource(Source: Variant)
var
    CoreTokenMgt: Codeunit "DXP Core Token Mgt.";
    DocMgt: Codeunit "DXP Document Mgt.";
    JsonHelper: Codeunit "DXP Json Helper";
    DocTransferMgt: Codeunit "DXP Document Transfer Mgt.";
    Header: Record "DXP SQZ Document Header";
    Line: Record "DXP SQZ Document Line";
    HeaderJObj: JsonObject;
    LinesJArr: JsonArray;
    LineJObj: JsonObject;
```

```
begin
    Header := Source;

    HeaderJObj.Add(CoreTokenMgt.GetVendorNoTok(), Header."Buy-from Vendor No.");
    HeaderJObj.Add(CoreTokenMgt.GetDocDateTok(), Header."Document Date");
    HeaderJObj.Add(CoreTokenMgt.GetDocReferenceTok(), Header."Document Reference");

    // Benutzerdefinierte Felder (Erweiterungsfelder) aus den Source-Tabellen hinzufügen
    HeaderJObj.Add(CoreTokenMgt.GetCustomFieldsTok(), JsonHelper.Rec2JsonFieldArray(Header,
true, true, Header."Document Class", Enum::"DXP Field Type"::Header));

    Line.SetRange("Document No.", Header."No.");
    if Line.FindSet() then
        repeat
            Clear(LineJObj);
            LineJObj.Add(CoreTokenMgt.GetDescriptionTok(), Line.Description);
            LineJObj.Add(CoreTokenMgt.GetQtyTok(), Line.Quantity);
            LineJObj.Add(CoreTokenMgt.GetCustomFieldsTok(),
JsonHelper.Rec2JsonFieldArray(Line, true, true, Header."Document Class", Enum::"DXP Field
Type"::Line));
            LinesJArr.Add(LineJObj);
        until Line.Next() = 0;

    DocTransferMgt.AddDocLinesToHeaderJson(HeaderJObj, LinesJArr);

    // An Core-Zielverarbeitung übergeben
    DocMgt.UpdateDocument(
        Header."Core Document No.",
        Enum::"DXP Document Status"::Transferred,
        Enum::"DXP Target Document Process"::"MY Create Custom Document",
        Enum::"DXP Next process step"::"Standard document",
        HeaderJObj);
end;
```

8) Schritt 6 — Eigenes Ziel?Dokument erstellen

Wenn Core Ihren eigenen Ziel-Datensatz erstellen soll, erweitern Sie `DXP Target Document Process` und implementieren Sie `DXP IDocument Processing`:

```
enumextension 50102 "MY Target Doc Process Ext." extends "DXP Target Document Process"
{
    value(50102; "MY Create Custom Document")
    {
        Caption = 'Create My Custom Document';
        Implementation = "DXP IDocument Processing" = "MY Custom Doc Creation";
    }
}

codeunit 50103 "MY Custom Doc Creation" implements "DXP IDocument Processing"
{
    procedure ProcessStandardDocument(JObject: JsonObject): RecordId
    var
        CoreTokenMgt: Codeunit "DXP Core Token Mgt.";
        DocTransferMgt: Codeunit "DXP Document Transfer Mgt.";
        JsonHelper: Codeunit "DXP Json Helper";
        MyDoc: Record "MY Custom Document";
        TargetVariant: Variant;
    begin
        MyDoc.Init();
        MyDoc.Validate("Vendor No.", JsonHelper.ValAsTxt(JObject,
CoreTokenMgt.GetVendorNoTok(), false));
        MyDoc.Validate("Document Date", JsonHelper.ValAsDate(JObject,
CoreTokenMgt.GetDocDateTok(), false));
        MyDoc.Insert(true);

        // Überträgt Metadaten sowie benutzerdefinierte/zusätzliche Felder anhand der
Core-Zuordnungen in den Ziel-Datensatz.
        TargetVariant := MyDoc;
        DocTransferMgt.ProcessAdditionalInformation(JObject, TargetVariant);
        MyDoc := TargetVariant;

        exit(MyDoc.RecordId());
    end;
}
```

Hinweis: Die Übertragung von Metadaten sowie benutzerdefinierten/zusätzlichen Feldern ist im obigen Beispiel bereits enthalten und basiert auf den Core-Zuordnungstabellen.

9) Fehlerbehebung (schnelle Prüfungen)

- Keine Einträge in `DXP SQZ Import Queue` → Dokumentenklassen-Einrichtung für SQUEEZE ist unvollständig.
- Core-`DXP Document` existiert, aber kein Source-Datensatz → Ihr `CreateSource(Document)` ist fehlgeschlagen oder gibt `false` zurück.
- Source existiert, aber Ziel wird nicht erstellt → `CreateProcessedJsonFromSource` wird nicht aufgerufen oder `DXP Target Document Process` ist nicht konfiguriert.
- Falsches Trigger-Verhalten beim Zuweisen → `DXP Custom Field Mapping.Validate App Field` und `Validate` prüfen.
  - `Validate` steuert, ob die Validierung des Ziel-Feldes (OnValidate) ausgeführt wird.

---

10) Was Core automatisch macht

Wenn Sie `DXP Document Mgt.UpdateDocument(...)` aufrufen:

- Core speichert das „Processed JSON“ in `DXP Document`.`JSON Processed`.
- Core setzt `DXP Document`.`Status` auf den übergebenen Status.
- Core verknüpft/aktualisiert `DXP Document`.`Linked-to Record Id` mit dem Datensatz, den Ihre `DXP IDocument Processing`-Implementierung zurückgibt.
- Core überträgt Core-Attachments auf den zurückgegebenen Datensatz (und hängt typischerweise das „Processed JSON“ als `.json`-Attachment an).

Revision #2
Created 2026-01-15 15:37:04 UTC by Bernd Feddersen
Updated 2026-01-29 09:10:04 UTC by Bernd Feddersen