

Implementation of a new, generic document class

SQUEEZE ? Core ? Your processing

This guide is deliberately focused and pipeline-oriented: It shows how document data is downloaded from SQUEEZE, processed in queues, and persisted in DEXPRO Core—and where a third-party extension comes into play.

Term: In this guide, “*SQUEEZE app*” refers to the Business Central extension that is connected to the SQUEEZE API and controls imports to DEXPRO Core.

Assumptions:

- Third-party developers typically work with **published icons**, not with the entire source code.
 - The SQUEEZE app handles **downloading + import queue + creation of the core document as well as the rudimentary creation of the SQUEEZE document, taking field mapping into account**, as soon as setup is complete.
 - Your extension takes over the business logic once the core document exists: create or extend source records, validate them, generate “Processed JSON,” and optionally create your own target object or document.
-

1) The end-to-end process SQUEEZE ? Core

Once the document class and setup are configured, the SQUEEZE app's import pipeline typically runs as follows:

1. **DXP Document Class** **extend** and bind your implementation: **DXP ISource Document**.
2. Configure settings for this **Document class-Setup**.
 - When validating **DXP SQZ Document Class ID**, the SQUEEZE app automatically retrieves the required setup values from SQUEEZE (e.g., **DXP SQZ Export Interface ID** and **DXP SQZ Line Table ID**) and fills them in.

3. The SQUEEZE app job calls `DXP SQZ API Mgt.DownloadDocumentData()`. Can also be called via “Download documents”.
 - New entries are created in the DXP SQZ import queue (document ID + target client).
4. The SQUEEZE app calls `DXP SQZ Import Queue Mgt.TransferImportQueueEntries()`.
 - The orchestration code unit `DXP Sqz Create Document` is executed for the entries in the import queue.
5. `DXP SQZ API Mgt.CreateCoreAndSQUEEZEDocument(DocClass, DocumentId)` loads the complete JSON.
 - `DXP Document Mgt.AddDocument(DocClass, DocumentId, RawJson, ...)` is called up.
 - Result: A `DXP Document`-Record exists. Status = `Imported`, `JSON Raw` is filled.
6. The SQUEEZE app calls your implementation `ISourceDocument.CreateSource(Document)`.
 - Your implementation begins here.

After step 6, you will have:

- A Core-Container-Document (`DXP Document`).
 - A SQUEEZE document in the SQUEEZE app.
 - Your source records should exist once `CreateSource` has completed successfully.
-

2) What you implement (Interfaces)

Source contract, i.e., squeeze page: `DXP ISource Document`

You implement this to:

- Create your editable source data sets from `DXP Document.JSON Raw`
- Validate source data records
- Generate “Processed JSON” and transfer it to core processing

Target contract, i.e., target document: `DXP IDocument Processing`

You implement this if Core is to create/update its **own target data set** (your own table/document) from the “Processed JSON.”

3) Step 1 — Define document class and assign source implementation

Create an enum extension and bind your source implementation:

```
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) Step 2 — Configure document class setup (so that the import queue is filled)

Configure the SQUEEZE-specific fields in the document class setup for your new document class.

Important:

- Validating the `DXP SQZ Document Class ID` triggers the automatic loading of the required configuration from SQUEEZE (e.g., `DXP SQZ Export Interface ID` and `DXP SQZ Line Table ID`).
- If the call fails, these API-derived fields are cleared again.

Result: The SQUEEZE app can execute `DownloadDocumentData()` and create entries in the DXP SQZ Import Queue for your class.

5) Step 3 — Ensure field mapping (downloaded mapping + your default values)

The SQUEEZE app can download the field catalog of the connected Squeeze client and persist it in Core Mapping tables (per document class).

Your role as a document class implementer:

- Provide standard field mapping “Squeeze Field Name → AL Field No.” via `InitFieldMapping(...)`.
- Alternatively, map fields via the assignment pages (custom field assignment).

5.1 Implement `InitFieldMapping` (independent example)

```
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 Validation switch when assigning fields: Validate app field

When the SQUEEZE app or your source creation logic writes JSON values to “app/source fields,” you can control whether the **OnValidate trigger** is executed.

In `DXP Custom Field Mapping`:

- `Validate App Field` = Validate the app field during the assignment.
- `Validate (Target)` = Perform validation when transferring to the target field.

Use this, for example, if extension fields have business logic in `OnValidate` or if you want to assign deliberately trigger-free (higher performance).

6) Step 4 — Create Source data sets in CreateSource

`CreateSource(Document)` is called by the SQUEEZE app immediately after `DXP Document Mgt.AddDocument(...)` as soon as the core container data record exists.

Minimum responsibilities:

1. Read `DXP Document.JSON Raw`.
2. Create SQUEEZE-Source-Header/Lines (`DXP SQZ Document Header` / `DXP SQZ Document Line`).
3. Connect Core-document with SQUEEZE-Source-Header via `UpdateDocumentAfterTransfer`.

Optional (recommended if you want functionality equivalent to the standard SQUEEZE document classes): 4) Download attachments if `DXP Download Attachments` is active. 5) Perform automatic validation if enabled for the document.

6.1 Example: Create SQUEEZE?Header + Lines with generic helpers

What the helpers do (so you don't have to implement it twice):

- `SqzDocumentMgt.CreateSQUEEZEDocHeader(Document, DocHeader, RawJson)`
 - Creates exactly one data record in `DXP SQZ Document Header` for the Core-document (`DXP Document`).
 - Sets identification fields as `Core Document No.` and the SQUEEZE-`API Document ID`.
 - Assigns header fields from JSON based on the configured mappings:

- DXP Field Mapping (field mapping)
- DXP Custom Field Mapping (Custom field mapping)
- Respects DXP Custom Field Mapping.Validate App Field and thus decides whether the **OnValidate trigger** is executed when assigning.
- Executes the integration event OnBeforeModifySQUEEZEDocumentHeader(...) as an extension hook.
- Standard post-processing is also performed for the built-in standard SQUEEZE classes (e.g., alternative IBAN/VAT ID determination, supplier update, standard posting date, duplicate check).
- SqzDocumentMgt.CreateSQUEEZEDocLine(DocHeader, RawJson)
 - Creates the DXP SQZ Document Line -data sets based on the JSON-Rows.
 - Assign line fields using the same mapping principle (standard + custom; optionally with validation triggers).
 - Fill the Core Document No. in the lines.
 - For standard SQUEEZE classes, standard line post-processing is also applied (e.g., transfer of recognized values).
 - Persists line-related auxiliary data (e.g., coordinates/metadata that use the UI and matching logic).

```

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.AutoValidationActivated(DocHeader."No.");
if AutomaticValidation then
    SqzDocumentMgt.ValidateSQUEEZEDocument(DocHeader, AutomaticValidation);

exit(true);
end;

```

7) Step 5 — Validation + Handover „Processed JSON“

7.1 Validation: `IsSourceDataPlausible`

Use the Core Plausibility Framework to ensure that errors are displayed consistently:

```

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 Build „Processed JSON“ and pass it to the core: CreateProcessedJsonFromSource

Core stores “Processed JSON” in `DXP Document.JSON Processed` and transfers it to your target processing.

```

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) Step 6 — Create your own target document

If you want Core to create your own target data set, extend `DXP Target Document Process` and implement `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;
}

```

Note: The transfer of metadata and user-defined/additional fields is already included in the above example and is based on the core mapping tables.

9) Troubleshooting (quick checks)

- No entries in `DXP SQZ Import Queue` → Document class setup for SQUEEZE is incomplete.
- Core-`DXP Document` exists, but no source record → Your `CreateSource(Document)` has failed or returns `false`.
- Source exists, but target is not created → `CreateProcessedJsonFromSource` is not called or `DXP Target Document Process` is not configured.

- Incorrect trigger behavior when assigning → Check `DXP Custom Field Mapping.Validate App Field` and `Validate`.
 - `Validate` controls whether validation of the target field (`OnValidate`) is performed.
-

10) What Core does automatically

When you call up `DXP Document Mgt.UpdateDocument(...)`:

- Core stores the “Processed JSON” in `DXP Document.JSON Processed`.
 - Core sets `DXP Document.Status` to the transferred status.
 - Core linked/updated `DXP Document.Linked-to Record Id` with the record, which your `DXP IDocument Processing`-implementation returns.
 - Core transfers Core attachments to the returned data set (and typically appends the “Processed JSON” as a `.json` attachment).
-

Revision #5

Created 2026-01-29 09:15:36 UTC by Christoph Koepke-von Seth

Updated 2026-01-29 10:40:54 UTC by Christoph Koepke-von Seth