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.

Revision #4
Created 2024-02-06 17:38:16 UTC by Bernd Feddersen
Updated 2025-01-28 14:59:45 UTC by Bernd Feddersen