**Xero-Salesforce Integration Documentation**

**Overview**

This integration pushes Invoice\_\_c records from Salesforce to Xero Accounting when the invoice's X\_Status\_\_c is set to **"Synced"**. It uses:

* OAuth 2.0 (Authorization Code Flow)
* Named Credentials + External Credentials
* Custom Metadata for tenant ID
* Apex callouts via @future method
* Line item and error validation

**Manual Authentication (Every 30 Minutes)**

**Notes:**

* Xero OAuth tokens expire every 30 minutes.
* You must reauthenticate to get a fresh Access Token.

**1. Manual Steps:**

1. Go to Salesforce → **Named Credentials**
2. Create a Auth Provider :
   1. Label - Xero Auth Provider
   2. Provide – Authentication protocol (OAuth 2.0), Client ID, Client Secret, Authorization end point URl, Token Endpoint URL
   3. It will generate Call back URL
3. Create a External Credential:
   1. Label - Xero External Credential
   2. Provide – Authentication protocol (OAuth 2.0), Authentication Flow Type – Browser Flow, Scope: ‘openid profile email accounting.transactions’
   3. Create Named Principles : label - Xero Principal, Identity Type – Named Principal and give scope (This is the authentication status that we need to authenticate for every 30 mins)
4. Create a Named Credential:
   1. Label - Xero Named Credential
   2. URL - <https://api.xero.com>
   3. Allow - Generate Authentication header and Allow formulas in HTTP Header
5. Create a permission set
   1. Label - Xero External Credential Access
   2. Navigate to External Credential Principal Access – edit and add your named principal - Xero\_External\_Credential - Xero Principal
6. After creating above steps go to you external credentials – named principals and authenticate it
7. Allow access and connection successful
8. Create a custom metadata for storing tenantID that generated from the Authtentication response
   1. Setup – Custom Metadata Types – New
   2. Label – Xero Configuration
   3. After saved, click new custom fields : dField type : Text, label: Tenant ID, Name: Tenant\_ID\_\_c, length : 255
   4. Save and add a record to it.
   5. Setup – Custom Metadata Types – Xero Configuration – Manage Records – New
   6. Label – Default Xero Org, Tenant ID: paste the tenant ID (You can use postman and Xero developer documentation to get the ID).

**2**. **Salesforce Objects & Fields**

**Invoice\_\_c**

| **Field Label** |  | **API Name** | **Notes** |
| --- | --- | --- | --- |
| X Status |  | X\_Status\_\_c | Picklist: "Synced", "Error" |
| Xero Invoice ID |  | Xero\_Invoice\_ID\_\_c | Stores returned Invoice ID |
| Xero Status |  | Xero\_Status\_\_c | Text status from Xero |
| Xero Response |  | Xero\_Response\_\_c | Debug/logging full response |

**Invoice Line (Invoice\_Line\_\_c)**

| **Field Label** | **API Name** | **Notes** |
| --- | --- | --- |
| X Description | X\_Description\_\_c | Required |
| Quantity | Quantity\_\_c | Required |
| X Unit Amount | X\_Unit\_Amount\_\_c | Required |
| AccountCode | (hardcoded to "200") | Required for Xero |
| TaxType | (hardcoded to "NONE") | Required for Xero |

**3**. **Apex Code for integration**

Create XeroInvoiceService.apxc

public with sharing class XeroInvoiceService {

@future(callout=true)

public static void sendInvoiceToXeroAsync(Id invoiceId) {

sendInvoiceToXero(invoiceId);

}

public static void sendInvoiceToXero(Id invoiceId) {

try {

Invoice\_\_c inv = [

SELECT Id, Name, X\_Status\_\_c, Due\_Date\_\_c, Invoice\_Date\_\_c,

Xero\_Invoice\_ID\_\_c, Xero\_Status\_\_c, Xero\_Response\_\_c,

(SELECT X\_Description\_\_c, Quantity\_\_c, X\_Unit\_Amount\_\_c, AccountCode\_\_c, TaxType\_\_c, Product\_Name\_\_c, Product\_Name\_\_r.Id

FROM Invoice\_Lines1\_\_r)

FROM Invoice\_\_c

WHERE Id = :invoiceId

LIMIT 1

];

if (inv.X\_Status\_\_c != 'Synced') {

System.debug('Invoice not marked as "Synced".');

return;

}

if (inv.Invoice\_Lines1\_\_r.isEmpty()) {

inv.Xero\_Status\_\_c = 'Error';

inv.Xero\_Response\_\_c = 'No Invoice Line Items found. Xero requires at least one.';

update inv;

return;

}

Xero\_Configuration\_mdt\_\_mdt config = [

SELECT Tenant\_ID\_\_c FROM Xero\_Configuration\_mdt\_\_mdt LIMIT 1

];

String tenantId = config.Tenant\_ID\_\_c;

List<Object> lineItems = new List<Object>();

for (Invoice\_Line\_\_c line : inv.Invoice\_Lines1\_\_r) {

if (line.Quantity\_\_c != null && line.X\_Unit\_Amount\_\_c != null) {

Map<String, Object> itemLine = new Map<String, Object>{

'Description' => line.X\_Description\_\_c != null ? line.X\_Description\_\_c : 'Item',

'Quantity' => line.Quantity\_\_c,

'UnitAmount' => line.X\_Unit\_Amount\_\_c,

'AccountCode' => line.AccountCode\_\_c != null ? line.AccountCode\_\_c : '200',

'TaxType' => line.TaxType\_\_c != null ? line.TaxType\_\_c : 'NONE'

};

if (line.Product\_Name\_\_r != null && String.isNotBlank(line.Product\_Name\_\_r.Id)) {

itemLine.put('ItemCode', line.Product\_Name\_\_r.Id);

}

lineItems.add(itemLine);

}

}

if (lineItems.isEmpty()) {

inv.Xero\_Status\_\_c = 'Error';

inv.Xero\_Response\_\_c = 'All Invoice Lines were empty or invalid.';

update inv;

return;

}

Map<String, Object> invoiceMap = new Map<String, Object>{

'Type' => 'ACCREC',

'Contact' => new Map<String, Object>{ 'Name' => inv.Name },

'Date' => String.valueOf(inv.Invoice\_Date\_\_c),

'DueDate' => String.valueOf(inv.Due\_Date\_\_c),

'LineItems' => lineItems,

'Status' => 'AUTHORISED'

};

Map<String, Object> body = new Map<String, Object>{

'Invoices' => new List<Object>{ invoiceMap }

};

HttpRequest req = new HttpRequest();

req.setEndpoint('callout:Xero\_Named\_Credential/api.xro/2.0/Invoices');

req.setMethod('POST');

req.setHeader('Content-Type', 'application/json');

req.setHeader('Accept', 'application/json');

req.setHeader('xero-tenant-id', tenantId);

req.setBody(JSON.serialize(body));

Http http = new Http();

HttpResponse res = http.send(req);

if (res.getStatusCode() == 200 || res.getStatusCode() == 201) {

// Check if the response is XML (error) or JSON (success)

if (res.getBody().startsWith('<')) {

inv.Xero\_Status\_\_c = 'Error - Non-JSON response';

inv.Xero\_Response\_\_c = res.getBody();

update inv;

return;

}

Map<String, Object> resMap = (Map<String, Object>) JSON.deserializeUntyped(res.getBody());

List<Object> invoices = (List<Object>) resMap.get('Invoices');

if (!invoices.isEmpty()) {

Map<String, Object> result = (Map<String, Object>) invoices[0];

String xeroInvoiceId = (String) result.get('InvoiceID');

String xeroStatus = (String) result.get('Status');

inv.Xero\_Invoice\_ID\_\_c = xeroInvoiceId;

inv.Xero\_Status\_\_c = xeroStatus;

inv.X\_Status\_\_c = 'Synced';

inv.Xero\_Response\_\_c = 'Invoice sent successfully.';

update inv;

} else {

inv.Xero\_Status\_\_c = 'Error';

inv.Xero\_Response\_\_c = 'Invoice sent, but no response returned.';

update inv;

}

} else {

inv.Xero\_Status\_\_c = 'Error ' + res.getStatusCode();

inv.Xero\_Response\_\_c = res.getBody();

update inv;

}

} catch (Exception e) {

System.debug('Exception: ' + e.getMessage());

Invoice\_\_c errorInvoice = new Invoice\_\_c(Id = invoiceId);

errorInvoice.Xero\_Status\_\_c = 'Exception';

errorInvoice.Xero\_Response\_\_c = e.getMessage();

update errorInvoice;

}

}

}

**Invoice Trigger**

trigger InvoiceTrigger on Invoice\_\_c (after insert, after update) {

for (Invoice\_\_c inv : Trigger.new) {

Invoice\_\_c oldInv = Trigger.isUpdate ? Trigger.oldMap.get(inv.Id) : null;

Boolean isSynced = inv.X\_Status\_\_c == 'Synced';

Boolean wasNotSynced = oldInv != null && oldInv.X\_Status\_\_c != 'Synced';

if ((Trigger.isInsert && isSynced) || (Trigger.isUpdate && isSynced && wasNotSynced)) {

XeroInvoiceService.sendInvoiceToXeroAsync(inv.Id);

}

}

}

**Testing & Debugging**

**To Test Manually:**

1. Authenticate in Named Credential
2. Insert or update Invoice\_\_c with X\_Status\_\_c = "Synced"
3. Check:
   * Fields: Xero\_Status\_\_c, Xero\_Response\_\_c, Xero\_Invoice\_ID\_\_c
   * Xero portal for invoice creation

**Common Errors & Fixes**

| **Error Message** | **Fix** |
| --- | --- |
| ValidationError: AccountCode or TaxType... | Ensure you hardcode or fetch these values in Apex |
| No Invoice Line Items found | Ensure child records are present and valid |
| Callout not supported from trigger | Use @future(callout=true) method like implemented |
| Unexpected character '<' | Add Accept: application/json header |
| Invalid Picklist Value: Synced | Add "Synced" to the X\_Status\_\_c picklist values |