EvolutionX API Integration Guide

Updated September 21, 2018

This is a high-level guide to integrating back-office ERP or accounts systems with EvolutionX Webstores (https://evolutionx.io). Its focus is the processes necessary for integration.

EvolutionX API technical details are found at https://docs.evoapi.io. This latest version of this document will be available from the API documentation page.

TERMS USED 1

EVOLUTIONX WEBSTORE AND INTEGRATION OPPORTUNITIES 2

B2B SALES PROCESS 2

EXAMPLE USE CASES: 2

USING THE WEBSTORE API INTEGRATION BY MODULE 3

MINIMAL VIABLE PRODUCT FOR API INTEGRATION WITH AN ERP 3
SPECIAL FIELDS AND VALUES 4
PROCESSING WEBSTORE ORDERS 4
ADDING ERP CUSTOMERS TO THE WEBSTORE 6
LIVE PRICING INTEGRATION 7
LIVE INVENTORY INTEGRATION 8
An example Request to the ERP *8*
An example Response from the ERP *8*

Terms Used

ERP – refers to the back-office system which is integrating with EvolutionX. This could in fact be an Accounts system or any other similar software which usually handles the Distributor's records of customers and orders and is used by the Distributor's staff.

Distributor – refers to the company selling products through the EvolutionX Webstore and who has the ERP system for their order records management.

Webstore – refers to the Distributor's branded EvolutionX Webstore which is processing online orders and offering products and prices branded for the Distributor.

API – refers to the EvolutionX API as documented at docs.evoapi.io.

EvolutionX Webstore and Integration Opportunities

EvolutionX is a SAAS Webstore platform and the API integration is the main focus of this document. There are no individual servers or software installations per Distributor store. There is no direct database access or server-installed software possible on EvolutionX for the purpose of integrating with an ERP.

The following methods can be used for getting information in and out of the Webstore:

- Webstore Admin – Staff can login to the Webstore admin and manage customers, products, pricing, orders and more (documentation at https://docs.evolutionx.io).
- Flat File Import and Export – Imports and exports can be done in the Webstore admin.
- Import Feeds – Timed/scheduled imports of inventory stock updates and product updates can be retrieved from a Distributor's FTP location.
- Product Feeds – Product data can be published e.g. for Google product feeds.
- API – Software developers can read and update Webstore records using the RESTful API.

B2B Sales Process

Typical business to business sales starts with staff securing a new customer and agreeing pricing and terms. The customer should expect to be able to order online and see the products and pricing they've negotiated reflected on the Webstore (when logged in). The staff should be able to manage the records in the ERP and know that the update to the Webstore will be automatic.

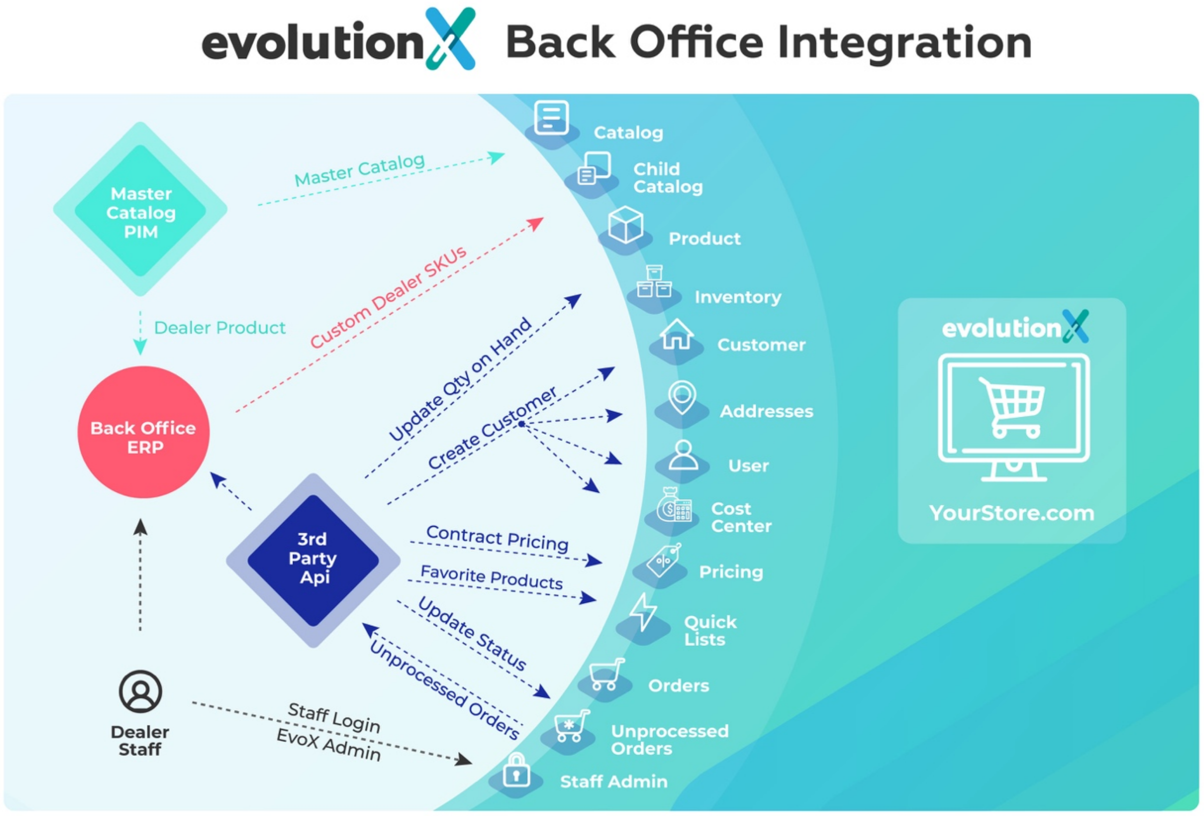
New customers ordering online should be seamlessly added to the ERP and flagged for staff follow-up. Identifying new opportunities for a B2B account with specific pricing on common products be a main goal as it increases repeat business.

Example Use Cases:

1. Sales staff create a customer business account in the ERP and add the contract pricing for the agreed products. Customer billing and shipping addresses are added along with the initial user(s).
   Integration Points include:
   i. Account
   ii. Addresses
   iii. Cost centers
   iv. Users
   v. Pricing (contract pricing)
2. Sales staff set custom/contract pricing for an existing customer's business account.
   Integration Points include:
   a. Pricing
      OR
   b. Live Pricing Integration (refer to section on pricing)

3. Sales staff add new users to an existing customer's business account and setup approval controls, budgets, and cost centers.
   Integration Points include:
   a. Customer Users
   b. Customer User budget
   c. Customer Cost center

Using the Webstore API Integration by Module



Minimal Viable Product for API Integration with an ERP

The following should be the minimum goal of a two-way integration between an ERP and EvolutionX Webstore.

- Customers – including addresses, users, cost centers
  - Create and update EvolutionX customer, address and user records from the ERP
  - Retrieve customer details based on orders received where no matching customer is in the ERP and create a matching customer record in the ERP.
- Orders – new orders polled at regular intervals
  - retrieve new unprocessed orders
  - update unprocessed orders as processed or other status

- Prices – live pricing by customer and product when browsing Webstore pages and ordering.
    - Where live pricing isn't possible, uploaded pricing through the API can be used.
- Inventory – live inventory by product when browsing Webstore product views and placing orders.
    - Where live pricing is not possible an API integration or scheduled FTP feed can be used.

Additional optional features are also described below but the above list is considered MVP. Note that Live Pricing and Live Inventory integrations are not part of the API and are Developed separately between the partners.

Special Fields and Values

Throughout the API there are certain recurring themes.

- Seller_reference is used for your third-party ERP ID and is supported in most records.
    - When reading Customer Addresses you can determine which ones were added by the user (if allowed by store permissions) because the seller_reference will be null.
- You can read a record using your ERP ID by filtering with the seller_reference in the query string but create and update requests must use the EvolutionX ID.
- Some objects may arrive without an ID such as customer details in a guest order.
- Bulk update options are specifically mentioned in the API documentation where they are supported.

Processing Webstore Orders

Orders are poled on a schedule using the unprocessed orders endpoint. This automatically filters for orders not yet processed by the ERP. It delivers the order objects and an array of IDs so that you can send a follow-up request to update the unprocessed orders as processed to prevent their inclusion in the next call.

The order object includes all data relevant to the order including the addresses at the time. If you separately wish to add or update the ERP with all of the known customer details (such as all of the addresses available on the Webstore customer account) you can make separate requests for these.

The recommend use of the Webstore API is as follows:

- Get unprocessed orders – Filtered for either "New" or "In Progress" Status
    - /orders/unprocessed?status=1
      This will filter for "New" unprocessed orders and **this is the recommended request**. Without a filter this request responds with all status except "Cancelled" but this can cause an issue when using the "Update Unprocessed Orders" request which will reset all orders status and this could cause unwanted behavior.
    - Store the order details in the ERP. Errors should be logged for staff. If you remove orders from the "ids" list before updating the API this will attempt the orders again.

    This is useful if the ERP is down during processing and the orders should be attempted again.
  - Note that the customer id field will be null for guest orders since they have no customer account record. This is also true of other elements such as addresses. The “guest” = true is the indicator for these orders.
  - https://docs.evoapi.io/#list-unprocessed-orders
- Update unprocessed orders – “In Progress” status
  - Make a Put request to the Update unprocessed orders endpoint with the “ids” provided in the last GET (except for ids of unprocessed orders). This will change the status of the Order to “In Progress” (status 2) in the Webstore.
  - https://docs.evoapi.io/#update-unprocessed-orders
- Update order shipping (optional).
  - A put request to update order shipping can describe an array of tracking codes for the packages shipped for the order. You can also opt to update the customer by email from the Webstore.
  - https://docs.evoapi.io/#update-order-shipping
- Update Order Status (optional).
  - A put request to the update order request will allow an update of the order status beyond the “in progress” status supported above in the Update Unprocessed Orders request. For example you can mark an order as complete you will set the status_id = 7.
  - An optional query parameter “bypass” = true can be used to prevent event driven updates from firing when the order status changes (assuming there are any). This can be left out usually unless you are bulk updating historical records.
  - https://docs.evoapi.io/#update-order
  - https://docs.evoapi.io/#order-statuses
- Update Order Payment status (optional).
  - The payment object within the order has a status indicating if the payment method is considered paid, refunded etc. Online payments are automatically updated with the relevant status but there are many instances where you may want to set the payment status through the API.
  - https://docs.evoapi.io/#update-order-payment
  - The full list of payment status codes: https://docs.evoapi.io/#payment-statuses
- Update Order Timeline with order notes (optional)
  - The order timeline is a feature which shows the history of changes to an order. Events in the admin and through the API are automatically recorded there so that changes can be tracked. If you want to add custom notes to the order timeline based on changes in the ERP or notes from the Distributor’s staff you can add a custom note using a Post request to the Update Order Timeline request. The note is automatically recorded with a date and time of the entry. In order to add a different date or time to the message simply add it as part of the note. There is a limit of 255 characters for timeline notes but no limit on the number of note records per order.

    - https://docs.evoapi.io/#update-order-timeline
- To optionally update the ERP customer record with all of the information the Webstore maintains (such as all stored addresses), read the objects as separate calls.
    - Note that where the order object field “guest” = true then the only field values available are from the Order object and the following requests are not possible.
    - Customer Account records can include:
        - List Customer object https://docs.evoapi.io/#customers
        - List all Addresses for a customer https://docs.evoapi.io/#the-customer-address-object
        - List all Users for the customer (note that budgets, user settings, and user cost centers are all available as separate requests). https://docs.evoapi.io/#list-users-for-a-customer
        - List all Cost Centers for a customer (note that cost center budgets and users are also available as separate requests) https://docs.evoapi.io/#list-all-cost-centers
        - List all Quicklists for a Customer (these are favorites and other lists, note that Quicklist items can be requested separately) https://docs.evoapi.io/#list-all-quicklists

Adding ERP Customers to the Webstore

As new customers are added to the ERP by staff you can optionally push them to the Webstore ensuring that their account details, addresses, users, Quicklists, etc. are available for use.

The minimum records for a Customer Account which can be used by EvolutionX are:

- Customer record
- User record
- Address record

A single request to the Create Customer (https://docs.evoapi.io/#create-customer) can include all three entries but can only contain one user and one address. Note that when the request has all three elements it processes each in turn and if any fail it will report the failure and stop processing the request. If this happens the previous requests that were successful will remain as successful records.

The Create Customer can be used to only request the customer record to be created without the user and address record information but then follow-up requests must be made to add the users and addresses to EvolutionX.

Example of for creating customers in EvolutionX:

- Use a combined Create Customer request and include the first address and user details as well as the account level values. If the ERP supports multiple addresses and users per customer then add the additional addresses and users using separate requests.
    - Create Customer – include the ERP ID as the seller_reference field (account_number is used by EvoLink). Include the first address and user details if desired (if there is only one address and one user then the create address and create user requests are not separately required).
      https://docs.evoapi.io/#create-customer
    - Create Addresses for the Customer – reference the customer ID and submit a separate post request for each address being created. Use the seller_reference field to store the ERP ID. Each address is a separate post request.
      https://docs.evoapi.io/#create-address-for-a-customer
    - Create Users for the Customer – reference the customer ID and submit the user's details. A password can be provided, or one will be generated if it's not provided. The seller_reference is used for storing a third-party ID such as the ERP or CRM ID. Note that the **username field should not be used except** in cases where two or more users are required to use the same group email address, for example:
        - Email: sales@bigcompany.tld, user: stan
        - Email: sales@bigcompany.tld, user: jenny
        - Email: sales@bigcompany.tld, user: bill
- Update a customer
    - The Webstore customer ID must be used. If it's not stored in the ERP then it can be retrieved using the following request.
- Find a Customer using the ERP ID
    - Use the List all Customers request and, in the query string, reference the seller_reference = ERP ID.
      https://docs.evoapi.io/#list-all-customers

Live Pricing Integration

ERP price rules are not re-created on the Webstore. Instead the special pricing agreed per customer and product (where it differs from the public pricing) are produced by the ERP as flat pricing data which can either be uploaded to the Webstore using the Pricing end point or can be requested by the Webstore on demand (Live Pricing). Live pricing is preferred.

Live Pricing integration is developed between the software partners and usually consists of a request from the Webstore to the ERP API using the Distributor id, product id, quantity, location code, and customer id.

Alternatives to Live Pricing are to push pricing using the Webstore API (https://docs.evoapi.io/#pricing).

Live Inventory Integration

A Live Inventory integration is the best way to provide up-to-date stock quantities on products when users are viewing or ordering them. This is developed between the software partners and usually consists of a request from the Webstore to the ERP API with Distributor id and product code fields. The webstore supports a single quantity value for live pricing.

Live Inventory integration is developed between the software partners.

An example Request to the ERP

```
<StockPriceEnquiryRequest>
  <AccountNumber>00000</AccountNumber>
  <DeliveryType>Standard</DeliveryType>
   <LatestAcceptableDate>2017-02-17</LatestAcceptableDate>
  <Products>
    <Product>
      <Code>100001</Code>
    </Product>
  </Products>
</StockPriceEnquiryRequest>
```

An example Response from the ERP

```
<?xml version="1.0"?>
<StockPriceEnquiryResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SupplierName>Supplier Name</SupplierName>
<AccountNumber>00000</AccountNumber>
<Products>
    <Product>
        <Code>100001</Code>
        <Description>Product Name Pk100</Description>
        <PackSize>1</PackSize>
        <Quantity>110600</Quantity>
        <IsDiscontinued>false</IsDiscontinued>
        <Errors/>
    </Product>
</Products>
</StockPriceEnquiryResponse>
```

An alternative to Live Inventory is to use the bulk update inventory request which is currently limited to 10 products per request. https://docs.evoapi.io/#bulk-create-or-update-inventory