

**EcoATM   
Integration Design Document**

**Document v1.0**

**Deposco, Inc.**1-877-770-1110 | www.deposco.com  
© Copyright 2025. All Rights Reserved.

# **Table Of Contents**

[**Table Of Contents 2**](#_po8yvhm44ya5)

[**Integrations 3**](#_poqmfd5p9nhl)

[Integration Diagram 3](#_tcsztbbi2oxm)

[ecoATM API Integration 3](#_cl3jxcfjztyc)

[Integration Overview 3](#_5ycr2hz809yj)

[Item API (2.0 API) 6](#_r1xtcrogtovk)

[Purchase Order API (2.0 API) 8](#_is414twt1dy2)

[Inbound Transfer Order API (2.0 API) 13](#_aw1da2dzlldg)

[Customer Return (RMA) - (2.0 API) 15](#_39yv82puclyc)

[Receipt Line - Search API - (legacy API) 19](#_q5umgnii774t)

[Sales Order API - (2.0 API) 22](#_9wtqpdryoigd)

[Ship Notice - Webhook API 26](#_ar40wwwtivsa)

[Shipment API - (legacy API) 35](#_w3fk8d2tlb76)

[Inventory API - (legacy API) 36](#_r2jgkcjyf73u)

[Inventory Adjustment - Search API (legacy API) 38](#_3gnavixu82k)

[Stock Inventory Modification API - (2.0) 41](#_ej92por92tx8)

[**Customer Agreement 52**](#_spssmwgabg43)

# 

# **Integrations**

## **Integration Diagram**

## 

## **ecoATM API Integration**

#### ***Integration Overview***

ecoATM uses Deposco’s open APIs and Data Exchange to establish the following connections between Oracle/FutureDial and Deposco for the following data:

Inbound Data Exchange Touchpoints - data uploaded to Deposco by ecoATM via Data Exchange

* Purchase Order:
  + Purchase Orders IMEI
    - A manifest is uploaded through Data Exchange using the following Data Exchange “**Manifest Upload**”
  + Purchase Orders Case Lots
    - A manifest is uploaded through Data Exchange using the following Data Exchange “**Case Lot Manifest Upload**”
* Inbound Transfer Order:
  + A manifest is uploaded through Data Exchange using the following Data Exchange “**Inbound Transfer Order Upload**”
* Customer Returns (RMA)
  + A manifest is uploaded through Data Exchange using the following Data Exchange **“Customer Return Upload”**

Inbound API Touchpoints - data posted to Depsoco by ecoATM via REST API.

* Item
* Purchase Order - Update
* Customer Return (RMA)
* Sales Order
* Sales Order - Update
* Inventory Modification - Update (new API)

Outbound API Touchpoints - data retrieved from Deposco by ecoATM via REST API

* Purchase Order
  + GET to retrieve PO information
* Transfer Order
  + GET to retrieve ITO information
* Receipt Line(s)
* Sales Order Picked
  + GET to retrieve SO information
* Ship Confirmation
  + Webhook shipNotice
  + No other information should be needed outside of the webhook response
* Inventory
  + InvAdjustment
  + Inventory Summary
* Search API - Various use cases
  + Stock Unit in Case Lot Location

**Please Note: API 2.0 - Throttling & Rate Limit Headers**

Please reference the **Deposco Developer Portal** at <http://developer.deposco.com> for “API (2.0) References” on **Throttling & Rate Limit Headers**.

* **Throttling:**
  + ecoATM “**Bright Cloud platform tier**” = 1, 2, or 3
    - **API request rate** = 4 requests per second
    - **Burst request size** = 20 requests
  + If the API continually receives more than 4 requests per second, then the burst bucket continues to grow until the burst request size limit is reached.
    - At that point, the HTTP 429 error code is returned, and the request must be resubmitted.
* **Rate Limit Headers:**
  + Each API response includes a set of HTTP headers that can help you identify what your current rate limit is and how many requests you have remaining in the current 1 second time window.
    - Please reference the **Deposco Developer Portal** for examples of “Rate Limit Headers”
  + The Deposco API gateway also has a 29 second timeout threshold for all requests. Any request that reaches this limit will fail with a HTTP error code of 400 and will need to be retried.

**Please Note:**

The only Deposco event that is maintained for external webhook use is the **shipNotice** event. While other events are available (such as **purchaseOrderCreated** and **orderCompleted**) and may work for the ecoATM current business case, these are not and will not be maintained by the Deposco product team for external usage. In the event that such events are deprecated, updated, or replaced by new versions, the ecoATM team will be responsible for adjusting their processes to accommodate for these changes as they arise. Any time spent by Deposco to assist with process updates related to such changes will be logged as billable time.

As discussed, the alternative to using these events as webhooks is for the ecoATM team to leverage base supported functionality using Deposco’s Search API on a scheduled cadence with specified filters to gather the information needed.

Full documentation on Deposco’s API requirements and best practices (including schema) are found on ([Help Site/Developer Portal](https://docs.deposco.com/docs/html/Content/API/API.htm?tocpath=API%7C_____0)). The following sections of this document outline API usage information specific to the ecoATM solution.

#### ***Item API (2.0 API)***

ecoATM leverages the Item API to create and update item level master data for items in bulk. Also included in this call is data specific to Packs (units of measure for each item) and Channel Listings.

##### Syntax

* POST:
  + Creates a new item or updates data for an existing item.
    - Production:
      * https://api.deposco.com/latest/items
    - UA:
      * https://api.deposco.com/beta/items

##### Cadence

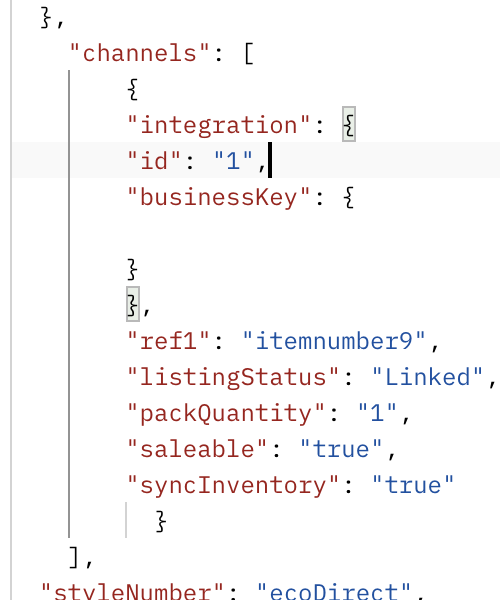
ecoATM posts item updates daily. Only items which have been updated in Oracle since the last POST are updated in the daily POST.

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Item
  + ***number*** = SKU or item number in Oracle
  + ***businessUnit***
    - ID = 73
    - businessKey
      * code = ECOATM
  + *salesEnabled* = true (Inventory Item Status)
  + *shippableFlag* = true
  + *inventoryTrackingEnabled* = true
  + *serialTrackingEnabled* = true
  + *cycleCount* =true
  + *originCountry*:
    - Populated via “Automations”
  + *customAttribute1* = modelYear
  + *customAttribute2* = carrier
  + *customAttribute3* = capacity
  + *customAttribute4* = condition
  + *customAttribute5* = Itemtype (i.e. INT, GSB, PWS)
  + *size* = grade
  + *styleName* = targetSalesChannel
  + *productCategory* = category
  + *brandName* = brand
  + *name* = model
  + *colorName* = colorName
  + *colorCode* = sortableColor
    - Populated via “Automations”
  + *longDescription* = longDescription
  + *shortDescription* = model-color
  + *externalImageUrl* = imageURL
* Pack
  + *type* = Each (required for all items), Case when applicable
  + *quantity* = 1 (required for all Each packs), Case Quantity when applicable
* Channel Listing
  + ***integration***
    - *ID = integrationId*
    - *businessKey*
      * *name = ECOATM API*
  + *listingStatus = Linked*
  + ***packQuantity*** *= 1*
  + *saleable = true*
  + *syncInventory = true*
  + ***ref1*** *= ecoATM itemNumber / sku*
  + *Example:*
    - *DEP Item* ***Number****: PWS1234*
    - *Premium Wholesale Item* ***Number****: PWS20250228*
      * Channel Listing > **REF1** = *PWS20250228*

**Note: Sample Payload example that includes the “Channel Listing” fields:**

****

#### ***Purchase Order API (2.0 API)***

ecoATM leverages the Purchase Order API to “update” Purchase Orders that will be received in Deposco. EcoATM posts a Purchase Order Update to change the unknown skus in the order lines to the ungraded skus.

ecoATM leverages the Purchase Order API to “retrieve” Purchase Orders via API requests to allow EcoATM to use their external systems to collect the metadata and incumbrances related to the IMEIs at the order line level.

##### Syntax: POST

* Create or update a purchase order
  + Production:
    - https://api.deposco.com/latest/orders/purchaseOrders
  + UA:
    - https://api.deposco.com/beta/orders/purchaseOrders
* Update an order line for a purchase order
  + Production:
    - https://api.deposco.com/latest/orders/purchaseOrders/{orderId}/orderLines/{orderLineId}
  + UA:
    - https://api.deposco.com/beta/orders/purchaseOrders/{orderId}/orderLines/{orderLineId}

##### Parameters: Path

##### Required:

* + - * {orderId} - ID of Order Header
      * {orderLineId} - ID of Order Line

##### Syntax: GET

* Get a list of Purchase Orders:
  + Production:
    - https://api.deposco.com/latest/orders/purchaseOrders/{orderId}
  + UA:
    - https://api.deposco.com/beta/orders/purchaseOrders/{orderId}

##### Parameters: Query:

* + - businessUnit - ECOATM
    - searchId
    - pageSize
    - createdAfter
    - updatedAfter
    - number - PO Number
    - orderStatus
* Get a Purchase Order:
  + Production:
    - https://api.deposco.com/latest/orders/purchaseOrders/{orderId}
  + UA:
    - https://api.deposco.com/beta/orders/purchaseOrders/{orderId}
* Get a list of order lines for a purchase order
  + Production:
    - https://api.deposco.com/latest/orders/purchaseOrders/{orderId}/orderLines
  + UA:
    - https://api.deposco.com/beta/orders/purchaseOrders/{orderId}/orderLines

##### Parameters: Path

##### Required:

* + {orderId} - ID of Order Header

##### Cadence

ecoATM uploads new Purchase Orders (manifest) via Data Exchange in Deposco.

ecoATM retrieves the Purchase Order in Deposco via API requests on a frequent cadence. ecoATM posts a Purchase Order Update hourly if an update was made in the Oracle.

Please reference the **Deposco Developer Portal** at <http://developer.deposco.com> for “API References” on **Throttling & Rate Limit Headers** or section “Integration Overview” > “**Please Note: API 2.0 - Throttling & Rate Limit Headers**”

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Purchase Order
  + *type* = Purchase Order
  + *businessUnit* = ECOATM
  + ***number*** = PO Number / Identifier
  + *customerOrderNumber* = ecoATM PO #
  + *otherReferenceNumber* = invoiceNumber
  + *consigneePartner* = vendor/customer
  + *otherReferenceNumber2* = trackingNumber
  + *shipVia = carrier*
  + *orderSource:*
    - *DDS (Device Source) = default*
  + *importType:*
    - *Unit (Grocery Scan)*
    - *Case (LPN)*
  + *orderStatus*
    - *Draft*
      * *ecoATM upload PO via DE*
    - *Hold*
      * *ecoATM pushes updates to set status = Hold when a PO is processing the IMEI*
    - *New*
      * *ecoATM pushes updates to set status = New when processing is complete*
  + *placedDate = dateShipped*
  + *plannedArrivalDate = plannedArrivalDate*
  + *channels:*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM PO # = PO1234 (ie customerOrderNumber)*

*ref1 = PO1234*

* Purchase Order Lines
  + *lineNumber*
  + *customerLineNumber*
  + *lineStatus*
    - *Draft*
      * *ecoATM upload PO via DE*
    - *Hold*
      * *ecoATM pushes updates to set status = Hold when a PO is processing the IMEI*
    - *New*
      * *ecoATM pushes updates to set status = New when processing is complete*
  + *itemNumber = sku*
  + *pack/type = Each*
  + *pack/quantity = 1*
  + *orderPackQuantity*
  + *customAttribute1 = acquisitionCost*
  + *customAttribute2 = targetSalesChannel*
  + *customAttribute3 = Encumbrance (Use of “Standardized Codes”*
    - *BlockList (BL)*
    - *FMI (FMI)*
    - *MDM (MDM)*
    - *CarrierFinancial (CF)*
    - *BlockList + FMI (BL-FMI)*
    - *Etc.*
  + *inventoryAttribute2 = Data Wipe Result*
  + ***inventoryAttribute1 = IMEI*** 
    - ***Please Note:*** 
      * ***API 2.0 does not include the “serialNumber” field on the OL schema and a reason for using “inventoryAttribute1”***
      * ***DEP will add logic to stamp the IMEI via “inventoryAttribute1” to the “serialNumber” field***
      * ***(ie serialNumber = IMEI)***
  + *exportReference = expectedGrade*
  + *inboundLPN = ecoATM Case Lot LPN*
  + *description = Ignore Encumbrance*
    - *True*
    - *False*
  + *priceCode = Order Type*
  + *unitCost*
  + *channels*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM Line # = 001*

*ref1 = 001*

#### ***Inbound Transfer Order API (2.0 API)***

ecoATM leverages the Inbound Transfer Order API to create and update Inbound Transfer Orders (if necessary) - which contain the fully graded, finished SKU (no encumbrance) - that will be received and is ready for putaway in Deposco.

##### Syntax: POST

* Create or update an Inbound Transfer Order
  + Production:
    - https://api.deposco.com/latest/orders/inboundTransferOrders
  + UA:
    - https://api.deposco.com/beta/orders/inboundTransferOrders

##### Cadence

ecoATM posts new Inbound Transfer Orders created in Oracle to Deposco hourly. Updates to existing Inbound Transfer Orders are posted hourly if an update was made in the Oracle.

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Inbound Transfer Order
  + *type = Inbound Transfer Order*
  + ***businessUnit*** *= ECOATM*
  + ***number*** *= ITO Number / Identifier*
  + *customerOrderNumber = ecoATM ITO #*
  + *otherReferenceNumber = invoiceNumber*
  + *consigneePartner = vendor/customer // Do we need for ITO? ECOATM*
  + *otherReferenceNumber2 = trackingNumber // Do we need for ITO?*
  + *shipVia = carrier // Do we need for ITO?*
  + *orderSource:*
    - *Retail (darryl)*
    - *Kiosk (Zach)*
    - *Trade-in*
  + *importType = Case (LPN) // Do we need for ITO always we use IMEI Number? Case*
  + *orderStatus = New*
  + *placedDate = dateShipped*
  + *plannedArrivalDate = plannedArrivalDate*
  + *channels:*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM ITO # = ITO1234 (ie customerOrderNumber)*

*ref1 = ITO1234*

* Inbound Transfer Order Lines
  + *lineNumber*
  + *customerLineNumber*
  + *lineStatus = New*
  + *itemNumber = sku*
  + *pack/type = Each*
  + *pack/quantity = 1*
  + *orderPackQuantity*
  + *customAttribute1 = acquisitionCost*
  + *customAttribute2 = targetSalesChannel // leave blank*
  + ***inventoryAttribute1 = IMEI***
  + ***serialNumber = IMEI/LPN ??????***
  + *inventoryAttribute2 = Data Wipe Result // format always success*
  + ***Please Note:*** 
    - ***API 2.0 does not include the “serialNumber” field on the OL schema and a reason for using “inventoryAttribute1”***
    - ***DEP will add logic to stamp the IMEI via “inventoryAttribute1” to the “serialNumber” field***
    - ***(ie serialNumber = IMEI)***
  + *exportReference = expectedGrade (Cosmetic Grade) // ignore*
  + *inboundLPN = ecoATM Transfer LPN // Box Number*
  + *productCode = eco Unit LPN  
    customAttribute3 = Encumbrance (Use of “Standardized Codes” - (Draft > New “PO Push” to flag)*
    - *BlockList (BL)*
    - *FMI (FMI)*
    - *MDM (MDM)*
    - *CarrierFinancial (CF)*
    - *BlockList + FMI (BL-FMI)*
    - *Etc. // always CLEAN*
  + *unitCost*
  + *channels*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM Line # = 001*

*ref1 = 001*

#### ***Customer Return (RMA) - (2.0 API)***

ecoATM leverages the Customer Returns (RMA) API to create or update Customer Return (RMA) Orders after the RMA process has been started in the sales portal and then communicated to Oracle.

ecoATM leverages the Customer Returns (RMA) API to retrieve Customer Return (RMA) Orders that were uploaded via Data Exchange.

##### Syntax: POST

* Create or update a Customer Return
  + Production:
    - https://api.deposco.com/latest/orders/customerReturns
  + UA:
    - https://api.deposco.com/beta/orders/customerReturns

**Syntax: GET**

* Get a list of customer returns
  + Production:
    - https://api.deposco.com/latest/orders/customerReturns
  + UA:
    - https://api.deposco.com/beta/orders/customerReturns

##### Parameters: Query

* + - businessUnit - ECOATM
    - createdAfter
    - updatedAfter
    - searchId
    - pageSize
    - number = RMA #
    - orderStatus
* Get a list of order lines for a customer returns
  + Production:
    - https://api.deposco.com/latest/orders/customerReturns/{orderId}/orderLines
  + UA:
    - https://api.deposco.com/beta/orders/customerReturns/{orderId}/orderLines

##### Parameters: Path

##### Required:

* + {orderId} - ID of Order Header

##### Cadence

ecoATM posts new Customer Return (RMA) Orders after the RMA process has been started in the sales portal and then communicated to Oracle hourly.

ecoATM retrieves new Customer Return (RMA) Orders after the RMA is created in Deposco via CreateRMA or ecoATM uploads manifest via Data Exchange.

##### Mappings

The fields and definitions below represent relevant data available to ecoATM in the Customer Return (RMA) Search call that are used to update Oracle appropriately.

* Customer Return (RMA)
  + ***businessUnit*** *= ECOATM /*
  + ***Number*** *= RMA #*
  + *customerOrderNumber = ecoATM RMA #*
  + *consigneePartner = vendor/customer*
  + *type = Customer Return*
  + *orderStatus = New*
  + *shipToFacility = LVL*
  + *parentOrder*
  + *placedDate = shippedDate*
  + *plannedArrivalDate = plannedArrivalDate*
  + *otherReferenceNumber = invoiceNumber*
  + *otherReferenceNumber2 = trackingNumber*
  + *exportReference = Process Down Stream Check*
  + *shipVia = carrier*
  + *orderStatus*
  + *orderSource:*
    - *DDS (Device Source) = default*
  + *importType:*
    - *Unit (Grocery Scan)*
    - *Case (LPN)*
  + *channels:*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM RMA # = RMA1234 (ie customerOrderNumber)*

*ref1 = RMA1234*

* Customer Return (RMA) Order Lines
  + *lineNumber*
  + *customerLinenumber*
  + ***Item*** *= sku*
  + ***Pack*** *(packKey)*
  + *orderPackQuantity*
  + *lineStatus*
  + *importReference = ecoATM SO number #*
  + *inventoryAttribute2 = Data Wipe Result*
  + ***inventoryAttribute1 = IMEI*** 
    - ***Please Note:*** 
      * ***API 2.0 does not include the “serialNumber” field on the OL schema and a reason for using “inventoryAttribute1”***
      * ***DEP will add logic to stamp the IMEI via “inventoryAttribute1” to the “serialNumber” field***
      * ***(ie serialNumber = IMEI)***
  + *notes*
    - ***title = RMA Reason***
    - *body = note message**(ie ecoATM Reason Code)*
  + *exportReference = expectedGrade*
  + *relatedOrderLine*
  + *customAttribute1 = acquisitionCost*
  + *customAttribute2 = targetSalesChannel*
  + *customAttribute3 = Encumbrance (Use of “Standardized Codes”)*
    - *BlockList (BL)*
    - *FMI (FMI)*
    - *MDM (MDM)*
    - *CarrierFinancial (CF)*
    - *BlockList + FMI (BL-FMI)*
    - *Etc.*
  + *description = Ignore Encumbrance*
    - *True*
    - *False*
  + *unitCost*
  + *channels*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM Line # = 001*

*ref1 = 001*

#### ***Receipt Line - Search API - (legacy API)***

ecoATM leverages the Search API to retrieve Purchase Order, Inbound Transfer Order, & Customer Return (RMA) Receipts from Deposco and update their Oracle appropriately. Note: used to “increment” Stock in Oracle

**Requests**

* Production:
  + https://api.deposco.com/integration/ECOATM/...
* UA:
  + https://sandboxapi.deposco.com/ua/integration/ECOATM/….

**Syntax: GET**

* Search for entities with a specific field value
  + GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/{entity}?{fieldName}={fieldValue}
* Search for entities with a specific field value on a related entity
  + GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/{entity}?{relatedEntity.fieldName}={fieldValue}
* Search for entities where a field has one of the specified field values
  + GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/{entity}?{fieldName}={fieldValue1},{fieldValue1}
* Search for entities based on multiple field values
  + GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/{entity}?{fieldName1}={fieldValue1}&{fieldName2}={fieldValue2}
* Examples:
  + Find receipts that were created after a certain date
    - GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/receiptLine?receiptDateTimeOps=%3E%3D&receiptDateTime=2025-05-18T00:00:00
  + Find all receipts that were created on a certain date
    - GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/receiptLine?createdDate=2025-02-20

##### Parameters: Path

Please reference the Help Guide page “Notes for using the search API” to reference the Parameters, Searchable entities, Operators, etc.

* {code} set to ECOATM
* {entity} set to “receiptLine”
* {fieldName}, {relatedEntity.fieldName}
  + Example using {fieldName}
    - Example: {fieldName} = “orderType” with expected values of either: Purchase Order, Customer Return (RMA), or Inbound Transfer Order
  + Example: {relatedEntity.fieldName} - If the field is on a related entity to the entity that you are searching for, then add the name of the related entity and a period before the field name
    - For example:
      * To search for all shipments for order header number “12345” - the syntax is GET /integration/ECOATM/search/Shipment?orderHeaders.number=12345
* operator
  + When searching by more than one field, the logical condition between the fields is always AND.
  + For example:
    - **Between**: createdDateOps=BETWEEN&createdDate=2025-02-20T01:27:23,2025-03-08T21:29:23
    - **Greater than:** createdDateOps=>&createdDate=2025-02-20T01:27:23
* fieldValue
* orderBy
* pageSize
* pageNo

##### Cadence

ecoATM searches for Receipt Lines in Deposco via Search API hourly.

##### Mappings

The fields and definitions below represent relevant data available to ecoATM in the Receipt Line Search call that are used to update Oracle appropriately.

* ReceiptLine
  + *orderType = Purchase Order, Customer Return, or Inbound Transfer Order*
  + *orderNumber = PO Number*
  + *lineNumber = PO “LineNumber”*
  + *number = receiptLine “Number” (auto-generated based on ReceiptLine ID)*
  + *itemNumber*
  + *quantity (receivedPackQuantity)*
  + *packType = Each*
  + *packQty = 1*
  + *receivingFacility = LVL*
  + *receivingLocation = (ie Receiving, Returns, Device Testing)*
  + *receivingContainer = “generic” Receiving LPN*
  + *receivingContainerType*
  + *receivedDate*
    - *Ex: 2025-02-05T13:30:23-06:00*
  + *status = Received*
  + *unitPrice = acquisitionCost*
  + *disposition = (DEP - reasonCode)*
    - *Defect confirmed (no restocking fee)*
    - *No defect found (apply restocking fee)*
    - *Unexpected device (50% restock fee)*
  + ***inventoryAttribute1 = IMEI*** 
    - ***Please Note:*** 
      * ***API 2.0 does not include the “serialNumber” field on the OL schema and a reason for using “inventoryAttribute1”***
      * ***DEP will add logic to stamp the IMEI via “inventoryAttribute1” to the “serialNumber” field***
      * ***(ie serialNumber = IMEI)***
  + *createPO*
    - *true*
    - *false*
  + *createItem*
    - *true*

#### ***Stock Unit - Search API - (legacy API)***

ecoATM leverages the Search API to retrieve Stock Units that are tied to a Case Lot Location/Zone to identify the Quantity and associated LPN/Container that are not allocated to an Order.

**Requests**

* Production:
  + https://api.deposco.com/integration/ECOATM/...
* UA:
  + https://sandboxapi.deposco.com/ua/integration/ECOATM/….

**Syntax: GET**

* Please reference the section “Receipt Line - Search API - (legacy API) > Syntax: Get” for additional syntax references that the Search API supports.
* Examples:
  + Find Stock Units that are associated with a Case Lot Location “Zone”
    - GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/StockUnit?locationZoneName=CaseLot
  + Find Stock Units that are associated with a Case Lot Location “Zone” for a specific Item
    - GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/StockUnit?locationZoneName=CaseLot&item.number=SPB\_QA\_1

##### Parameters: Path

Please reference the Help Guide page “Notes for using the search API” to reference the Parameters, Searchable entities, Operators, etc.

##### Cadence

ecoATM searches for Case Lot Stock Units in Deposco via Search API hourly.

##### Mappings

The fields and definitions below represent relevant data available to ecoATM in the Stock Unit Search call that are used to update Oracle appropriately.

* Stock Unit
  + *quantity*
  + *lpnNumber = Container/LPN*
  + *serialNumber = IMEI*
  + *locationZoneName = DEP Location “Zone” (ie CaseLot)*
  + *unitPrice*
  + *itemNumber = sku*
  + *location*
  + inventoryAttribute1 = “Original” IMEI
  + inventoryAttribute2 = (4) *Encumbrance (Use of “Standardized Codes”)*
    - *BlockList (BL)*
    - *FMI (FMI)*
    - *MDM (MDM)*
    - *CarrierFinancial (CF)*
    - *BlockList + FMI (BL-FMI)*
    - *etc.*
  + *receiptLineNumber*
  + *allocatedOrderNumber*
  + *businessUnit = ECOATM*
* Pack:
  + *itemNumber = sku*
  + *type = Each*
  + *quantity = # of Items in a Pack*
  + *weight*
  + *dimensions:*
    - *length*
    - *width*
    - *height*

#### ***Sales Order API - (2.0 API)***

ecoATM leverages the Sales Order API to create and update Sales Orders that will be received in Deposco.

ecoATM retrieves Sales Orders via API request to allow ecoATM to use their external systems to collect the metadata related to fulfillment orders that are waiting for Payment Approval.

##### Syntax: POST

* Create or update a Sales Order
  + Production:
    - https://api.deposco.com/latest/orders/salesOrders
  + UA:
    - https://api.deposco.com/beta/orders/salesOrders

**Syntax: GET**:

* Get a list of sales orders
  + Production:
    - https://api.deposco.com/latest/orders/salesOrders
  + UA:
    - https://api.deposco.com/beta/orders/salesOrders

##### Parameters: Query

* + - businessUnit - ECOATM
    - createdAfter
    - updatedAfter
    - searchId
    - pageSize
    - Number
    - orderStatus
* Get a list of order lines for a sales order
  + Production:
    - https://api.deposco.com/latest/orders/salesOrders/{orderId}/orderLines
  + UA:
    - https://api.deposco.com/beta/orders/salesOrders/{orderId}/orderLines

##### Parameters: Path

* + - Required:
      * {orderId} - ID of Order Header

##### Cadence

ecoATM posts new Sales Orders once an Order is available from ecoATM’s PWS Sales Platform.

ecoATM retrieves Sales Orders via API requests on a frequent cadence. ecoATM posts a Sales Order Update hourly if an update was made in the Oracle for sending a Payment Approval.

Please reference the **Deposco Developer Portal** at <http://developer.deposco.com> for “API References” on **Throttling & Rate Limit Headers** or section “Integration Overview” > “**Please Note: API 2.0 - Throttling & Rate Limit Headers**”

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Sales Order
  + *type = Sales Order*
  + ***businessUnit*** *= ECOATM*
  + ***number*** *= Oracle SO #*
  + *orderStatus = New*
  + *customerOrderNumber = ecoATM Customer Reference Number*
  + *consigneePartner*
    - ***ID***
    - ***businessKey***
      * *code = tradingPartner.code*
      * *For example - TP (Verizon)*
        + *TP “ID” = 4 (in the UA env)*
        + *TP “Code” = Verizon*
  + *freightTermsType*
    - *Prepaid*
    - *Third Party*
  + *customAttribute1 = paymentStatus*
    - *~~Hold~~*
    - *~~Ready to Ship~~*
    - *Payment Pending*
    - *Payment Received  
      export enum PaymentStatus {*
    - *Pending = 'Payment Pending',*
    - *Received = 'Payment Received'*
    - *}*
  + *shipVia - set via automations*
  + *orderTotal*
  + *orderShippingTotal*
  + *shipFromFacility = LVL*
  + *placedDate*
  + *paymentTerms (informational)*
  + *exportReference = Process Down Stream Check*
  + *notes*
    - *Title = Packing Note*
    - *Body = message*
  + *orderSource = salesChannel*
    - *Premium Wholesale (PWS)*
  + *channels:*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM Order # = SO107427*

*ref1 = SO107427*

* Sales Order Lines
  + ***lineNumber*** *- Identification number for the order line, including a next-up line number starting at 1.*
    - *Example:*
      * *SO Number = SO107427*
        + *SO LineNumber = SO107427--1, SO107427--2…etc.*
  + *customerLineNumber*
  + *lineStatus = New*
  + ***item*** *= itemNumber*
  + ***pack*** *= packKey*
  + ***orderPackQuantity***
  + *canceledPackQuantity*
  + *unitPrice*
  + *channels*
    - ***integration***
      * ***ID*** *= integrationId*
      * ***businessKey***
        + ***name = ECOATM API***
    - ***ref1*** *=**Internal reference for the entity in the external system.* 
      * *This field should be set to the ID or number for the entity.*
        + *Example:*

*ecoATM Line # = 001*

*ref1 = 001*

#### ***Ship Notice - Webhook API***

When a shipment is closed via a WMS process, WMS raises the shipNotice event. If there are partial shipments for an order, multiple shipNotices will be raised, one per shipment.

Note: “decrement” Stock in Oracle

##### Configuration

* From Event Definition - activate the shipNotice event.
  + IsActive = true
  + doEventLog = true
  + activating Socket = true
    - Connects the event to a webhook > Socket subscribes to the event.
* Set up webhook via the Event Stream UI
  + Click the “Add” button at top-right
  + Set “Type” = Webhook.
  + The callback URL is the URL that the payload will be sent to

##### Functionality Notes

* **sendASN** flag
  + Some customers do not want to receive shipment information unless the order has been fully fulfilled.
    - In order to support this via the shipNotice event, there is an attribute on both the orderHeader and tradingPartner called sendASN
  + If this is set to Fully Shipped, then the shipNotice generated with partial shipments will sit in a pending state until the order is fully shipped.
    - At that point, the last shipNotice raised will contain all fulfillment
  + information for the entire order and that is used to send the fulfillment to Oracle.
* Regenerating shipNotice Events
  + Sometimes an external system might be down when Deposco tries to send a fulfillment via a shipNotice.
    - Other times, there might be a data issue in Deposco that causes a failure in the external system.
    - Because of this - there are multiple ways to regenerate ship notice events.
  + In the case that there are just a few shipNotices that need to be regenerated - there is a ‘Regenerate Ship Notice’ “Action” available.
    - This action is available on the OrderHeader, Shipment, or Trip
    - The user has the option to regenerate all shipNotice events or just the ones that failed.
  + The other way to regenerate the shipNotice events in bulk is via a schedulerTask approach.
* Sample Payload:
  + Provide sample payload request

{

"ShipmentNotice": {

"Header": {

"Address": [

{

"Address1": "7213 Global Dr",

"Address2": "",

"AddressLocationNumber": "LVL",

"AddressName": "Louisville Facility",

"AddressTypeCode": "SF",

"City": "Louisville",

"Country": "US",

"LocationCodeQualifier": "92",

"PostalCode": "40258",

"State": "KY"

},

{

"Address1": "11605 Haynes Bridge Rd",

"Address2": "",

"Address3": "",

"Address4": "",

"AddressLocationNumber": "92",

"AddressName": "Daniel Stenger",

"AddressTypeCode": "ST",

"City": "ALPHARETTA",

"Contact": [

{

"ContactEmail": "",

"ContactFax": "",

"ContactName": "",

"ContactPhone": "0000000000",

"ContactTypeCode": "CH"

}

],

"Country": "US",

"LocationCodeQualifier": "92",

"PostalCode": "30009",

"State": "GA"

},

{

"Address1": "",

"Address2": "",

"AddressLocationNumber": "92",

"AddressName": "Daniel Stenger",

"AddressTypeCode": "BT",

"City": "",

"Contact": [

{

"ContactEmail": "",

"ContactFax": "",

"ContactName": "",

"ContactPhone": "",

"ContactTypeCode": "CH"

}

],

"Country": "",

"LocationCodeQualifier": "92",

"PostalCode": "",

"State": ""

}

],

"CarrierInformation": {

"CarrierAlphaCode": "FEDEX\_GROUND",

"CarrierEquipmentNumber": "",

"CarrierRouting": "Ground",

"CarrierTransMethodCode": "FEDEX\_GROUND",

"EquipmentDescriptionCode": "",

"SealNumber": "",

"ShipVia": "FedEx Ground",

"StatusCode": "CL"

},

"Date": [],

"QuantityAndWeight": {

"LadingQuantity": "1",

"PackingMaterial": "25",

"PackingMedium": "CTN",

"Weight": "1.0",

"WeightQualifier": "G",

"WeightUOM": "LB"

},

"ShipmentHeader": {

"ASNStructureCode": "0001",

"BillOfLadingNumber": "",

"CarrierProNumber": "285509003108",

"CreatedBy": "jonathan.yang",

"CurrentScheduledDeliveryDate": "2025-02-17",

"CurrentScheduledDeliveryTime": "10:32:00",

"CustomFields": [],

"RateZone": "4",

"RoutingNumber": "",

"ShipDate": "2025-02-17 10:32:06",

"ShipMethod": "FEDEX\_GROUND",

"ShipNoticeDate": "2025-02-17 10:32:00",

"ShipVendor": "FEDEX2",

"ShipmentId": "14",

"ShippingCost": "7.59",

"TotalPackages": "1",

"TrackingNumber": "285509003108",

"TradingPartnerId": "",

"TripNumber": "AUTO-FEDEX2-250217",

"TsetPurposeCode": "00"

},

"TrackingUrl": [

"https://www.fedex.com/fedextrack/?trknbr=285509003108"

]

},

"OrderLevel": [

{

"Address": {

"Address1": "11605 Haynes Bridge Rd",

"Address2": "",

"AddressLocationNumber": "481",

"AddressName": "Daniel Stenger",

"AddressTypeCode": "BY",

"City": "ALPHARETTA",

"Country": "US",

"LocationCodeQualifier": "92",

"PostalCode": "30009",

"State": "GA"

},

"OrderHeader": {

"AccountNumber": "",

"Channels": [],

"ContractNumber": "",

"CustomAttribute1": "",

"CustomAttribute2": "",

"CustomAttribute3": "",

"CustomAttribute4": "",

"CustomAttribute5": "",

"CustomAttribute6": "",

"CustomAttribute7": "",

"CustomerOrderNumber": "",

"Department": "",

"ExportReference": "",

"FreightTermsType": "Prepaid",

"ImportReference": "",

"ImportType": "Warehouse to Store",

"InternalOrderDate": "2025-02-17 10:19:51",

"InternalOrderId": "46",

"InternalOrderNumber": "46",

"InvoiceDate": "2025-02-17 10:32:00",

"InvoiceNumber": "14",

"OrderDiscountSubtotal": 0.0,

"OrderShipTotal": 0.0,

"OrderShippingTotal": 0.0,

"OrderSource": "",

"OrderSubtotal": 0.0,

"OrderTaxTotal": 0.0,

"OrderTaxableTotal": 0.0,

"OrderTotal": 0.0,

"OrderUntaxableTotal": 0.0,

"OtherReferenceNumber": "",

"PartnerInvoiceNumber": "",

"PlannedShipDate": "",

"PurchaseOrderDate": "2025-02-17 10:19:51",

"ShipVendor": "",

"ShipVia": "FedEx Ground",

"ShippingStatus": "20",

"Type": "Sales Order",

"Vendor": ""

},

"OrderLineLevel": [

{

"BornOnDate": "",

"BuyerPartNumber": "",

"CanceledPackQuantity": "0.0",

"Channels": [],

"ChargeAllowances": [],

"ConsumerPackageCode": "",

"CustomAttribute1": "",

"CustomAttribute2": "",

"CustomAttribute3": "",

"CustomerLineNumber": "",

"Date": [],

"DiscountAmount": "0.0",

"EAN": "",

"ExpirationDate": "",

"GTIN": "",

"ISBN": "",

"ItemLevel": {

"ColorCode": "",

"ColorName": "",

"Hazmat": "false",

"ItemId": "13",

"ItemNumber": "Ipad Mini",

"PriceInformation": [

{

"PriceTypeIDCode": "RTL",

"UnitPrice": "0.0"

}

],

"ProductOrItemDescription": {

"ItemDescriptionType": "08",

"ProductDescription": "Ipad Mini(A17 Pro)"

},

"Size": "",

"StyleName": "",

"StyleNumber": "",

"Taxable": "false",

"Weight": "0.0",

"WeightUOM": ""

},

"ItemNumber": "Ipad Mini",

"LineSequenceNumber": "1",

"LineTotal": "0.0",

"LotNumber": "",

"NatlDrugCode": "",

"OrderLineId": "119",

"OrderQty": "1.0",

"OrderQtyUOM": "",

"PickDetails": [

{

"ActualSource": "Shipping",

"LotNumber": "",

"PickedItem": "Ipad Mini",

"Quantity": "1",

"SerialNumber": "909090"

}

],

"ProductSizeCode": "",

"QtyInPack": "1",

"Reference": [],

"SerialNumber": "",

"ShipQty": "1.0",

"ShipQtyUOM": "EA",

"ShipmentLineLevel": [

{

"BornOnDate": "",

"ContainerLevel": {

"CODAmount": 0.0,

"CartonType": "",

"ContainerHistId": "10",

"Height": "7.0",

"InsuredValue": 0.0,

"LPN": "Cart1",

"Length": "4.0",

"MasterLPN": "",

"Number": "Cart1",

"ReturnTrackingNumber": "",

"SecondaryType": "",

"ShippingCost": 7.59,

"TrackingNumber": "285509003108",

"TrackingNumber2": "",

"Type": "Shipping",

"Volume": "0.0",

"Weight": "1.0",

"WeightUOM": "LB",

"Width": "6.0"

},

"ExpirationDate": "",

"InventoryAttribute1": "",

"InventoryAttribute2": "",

"InventoryCondition": "",

"ItemLevel": {

"ColorCode": "",

"ColorName": "",

"ItemId": "13",

"ItemNumber": "Ipad Mini",

"PriceInformation": [

{

"PriceTypeIDCode": "RTL",

"UnitPrice": "0.0"

}

],

"ProductOrItemDescription": {

"ItemDescriptionType": "08",

"ProductDescription": "Ipad Mini(A17 Pro)"

},

"Size": "",

"StyleName": "",

"StyleNumber": "",

"Weight": "0.0",

"WeightUOM": ""

},

"LotNumber": "",

"Number": "14--1",

"SerialNumber": "909090",

"ShipQty": "1.0",

"ShipQtyUOM": "EA",

"ShipmentLineId": "10",

"TrackingNumber": "285509003108",

"TrackingURL": "https://www.fedex.com/fedextrack/?trknbr=285509003108"

}

],

"SortOrder": "0",

"UPCCaseCode": "",

"UnitCost": "0.0",

"UnitPrice": "0.0",

"VendorPartNumber": ""

}

],

"QuantityAndWeight": {

"LadingQuantity": "1",

"PackingMaterial": "25",

"PackingMedium": "CTN",

"Weight": "0.0",

"WeightQualifier": "G",

"WeightUOM": "LB"

},

"Reference": [],

"TrackingDetails": [

{

"ContainerHistId": "10",

"OrderLineLevel": [

{

"OrderLineId": "119",

"ShipQty": "1.0"

}

],

"TrackingNumber": "285509003108",

"TrackingURL": "https://www.fedex.com/fedextrack/?trknbr=285509003108"

}

]

}

],

"Summary": {

"TotalLineItems": "1",

"TotalQuantity": "1.0"

}

}

}

#### ***Shipment API - (legacy API)***

ecoATM leverages the Shipment Search API to retrieve all Shipment Line Details from Deposco to get all IMEI(s) tied to the associated Sales Order Lines.

**Syntax: GET**

* Search for entities with a specific field value on a related entity. (Search API)
  + Production:
    - https://api.deposco.com/integration/{code}/search/{entity}?{relatedEntity.fieldName}={fieldValue}
  + UA:
    - https://sandboxapi.deposco.com/ua/integration/{code}/search/{entity}?{relatedEntity.fieldName}={fieldValue}
  + Example: Find all shipments for a specific order:
    - Production:
      * https://api.deposco.com/integration/ECOATM/search/shipment?orderHeaders.Number=xxx
    - UA:
      * https://sandboxapi.deposco.com/ua/integration/ECOATM/search/shipment?orderHeaders.Number=xxx
* Retrieve a shipment: (Shipment API)
  + Production:
    - https://api.deposco.com/integration/ECOATM/shipments/{shipmentNumber}
  + UA:
    - https://sandboxapi.deposco.com/ua/integration/ECOATM/shipments/{shipmentNumber}

##### Parameters:

* **{code}** = ECOATM
* shipmentNumber

##### Cadence

ecoATM searches for Shipped Shipments in Deposco via Search API hourly.

#### 

#### ***Inventory API - (legacy API)***

ecoATM leverages the Inventory API to retrieve **Inventory Summary** values from Deposco to get full inventory levels and update the Inventory in Oracle appropriately.

ecoATM also leverages the Inventory API to increment/decrement inventory levels and update the Inventory in PWS Sales Platform appropriately.

**Syntax: GET**

* Retrieve item inventory grouped by facility (Full Sync)
  + Production:
    - https://api.deposco.com/integration/ECOATM/inventory/full
  + UA:
    - https://sandboxapi.deposco.com/ua/integration/ECOATM/inventory/full

##### Parameters:

* **{code}** = ECOATM
* **pageNo**
* businessUnit = ECOATM
* facilityNumber
* pageSize
* startActivityTime / endActivityTime
  + Date range for when the inventory quantity was last created or updated.
  + Enter the values in yyyy-mm-ddThh:mm:ssZ format. For example:
  + GET /integration/{code}/inventory/full?facilityNumber=Warehouse&startActivityTime=2025-06-19T07:00:00Z&endActivityTime=2025-06-20T07:00:00Z

**Syntax: GET**

* Retrieve item inventory by facility and location
  + Production:
    - https://api.deposco.com/integration/integration/{code}/inventory/facility/{facilityNumber}/location/{locationNumber}
  + UA:
    - https://sandboxapi.deposco.com/ua/integration/ECOATM/inventory/facility/LVL/location/Receiving

##### Parameters:

* **{code}** = ECOATM
* businessUnit
* facilityNumber
* locationNumber

##### Cadence

ecoATM retrieves Inventory Summary in Deposco at the end of day to get full inventory levels and update the Inventory in Oracle appropriately.

ecoATM retrieves Inventory Summary in Deposco every 1-3 minutes to get inventory levels to increment/decrement in PWS Sales Platform

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Inventory Summary
  + *itemNumber = sku*
  + *facility = LVL*
  + *total = On-Hand Qty*
  + *availableToPromise*
  + *unallocated*
  + *Allocated*

#### ***Inventory Adjustment - Search API (legacy API)***

ecoATM leverages the Search API to retrieve Inventory Adjustment values from Deposco and update the Inventory in Oracle appropriately based on the Adjustment Type and Reason Code.

Note: Graded Inventory Adjustments are tracked via IMEI / Serial #

**Syntax: GET**

* Search for entities with a specific field value
  + GET https://sandboxapi.deposco.com/ua/integration/ECOATM/search/{entity}?{fieldName}={fieldValue}
* Example:
  + Find all inventory adjustments for a specific item
    - GET /integration/{code}/search/invAdjustment?itemSku=item1
  + Find inventory adjustments that were created after a certain date
    - GET /integration/{code}/search/invAdjustment?createdDate=2025-01-01&createdDateOps=>
* *Please reference the section “Receipt Line - Search API - (legacy API)” or Help Guide page “Notes for using the search API” to reference the additional options for: Syntax, Parameters, Searchable entities, Operators, etc.*

##### Parameters: Path

* {code} set to ECOATM
* {entity} set to “invAdjustment”
* {fieldName}, {relatedEntity.fieldName}
* operator
* fieldValue
* orderBy
* pageSize
* pageNo

##### Cadence

ecoATM searches for Inventory Adjustments in Deposco via Search API hourly.

**Schema**

The structure of a response for a GET request with the search API depends on the type of entity that you are searching for. Schemas are available at **http://{code}.deposco.com/schema/…**.

* Example: https://ecoatm-ua.deposco.com/schema/InvAdjustment.xsd

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Inv Adjustment
  + *invAdjustmentId*
  + *businessUnit = ECOATM*
  + *actionType = ADJ*
  + *facilityNumber = LVL*
  + *location = locationNumber*
  + *itemSku = itemNumber*
  + *packKey = pack*
  + *reasonCode*
    - *TW = Transfer to Wholesale*
    - *TFW = Transfer from Wholesale*
    - *WO = Writeoff*
    - *SF = Stock Found*
    - *IC = Item Change*
  + *lotNumber*
    - *“Case Lot” Orders - will display a value if the item is flagged as Lot Tracking Enabled*
  + *channels*
  + *quantity*
  + *Details (references associated IGL)*
    - *company = ECOATM*
    - *actionType = ADJ*
    - *code*
      * *CR - Credit*
      * *DR - Debit*
    - *price*
    - *quantity*
    - *itemSku*
    - *facilityNumber*
    - *inventoryLineNumber*
    - *packKey*

This API returns scalar response when quantity is 1 and an array when quantity is greater than 1. This is a bug in Deposco which is going to be handled via a workaround in the ecoATM codebase.

#### ***Stock Inventory Modification API - (2.0)***

ecoATM leverages the Inventory Modification API to update the previously Received and Staged Stock Unit(s) after Grading and Testing is complete.

*Note: Please refer below from sections “Assumptions - Write Activity and Inventory Tracking records”*

##### Syntax: POST

* Production:
  + https://api.deposco.com/latest/inventory/facility/{facilityid}/location/{locationid}/inventoryModification
* UA:
  + https://api.deposco.com/beta/inventory/facility/{facilityid}/location/{locationid}/inventoryModification

##### Cadence

ecoATM posts Inventory modifications daily for Items that are successfully Graded and passed Device Testing once Receiving is complete and the Inventory is Staged at a “Grading/Testing” Location.

##### Mappings

The field mappings below while not all data requirements for a successful API call are required for business flows defined for ecoATM.

* Stock Unit
  + businessUnit = ECOATM
  + *item* = sku
  + *pack* 
    - *packQuantity*
      * *(1) = Each*
      * *(x) = Case*
  + **serialNumber = IMEI**
  + **inventoryAttribute1 = “Original” IMEI**
  + **inventoryAttribute2 = (4)** *Encumbrance (Use of “Standardized Codes”)*
    - *BlockList (BL)*
    - *FMI (FMI)*
    - *MDM (MDM)*
    - *CarrierFinancial (CF)*
    - *BlockList + FMI (BL-FMI)*
    - *etc.*
  + ***inventoryCondition*** *=* ***Sort Type / Grouping***
  + *status - On Hold / Damaged / Inspection / Quarantine*
  + *unitPrice =* acquisitionCost
  + quantity
* Inventory Ledger
  + *reasonCode = ecoATM Summary Reason Code*
  + ledgerCustomAttribute1 = acquisitionCost
  + ledgerCustomAttribute2
    - *Data Wipe (text)*
      * *Passed*
      * *Failed*
    - *Functional Test (text)*
      * *Passed*
      * *Failed*
    - *Cosmetic Grade*
      * *01*
      * *02*
      * *03*
      * *A*
      * *B*
      * *C*
    - *Functional Failure Reason*
  + *ledgerCustomAttribute3 = Battery Health*
  + *Xx =* “last” WorkStation Location (reporting purposes - not required in Oracle)
    - Priority #3
* Activity Log
  + *transactionNumber (optional field) - populates in Activity Log*
  + *activityName (optional field) - populates in Activity Log*
  + *adjustmentUser (optional field) - populates in Activity Log*

**Assumptions:**

* ecoATM will utilize the “Inventory Modification” only.
* IF a Stock Unit requires any additional updates (ie “re-grade” / “passing updates”) - then there is potential for multiple calls for the same unit to be made

**Concept: Inventory Modification:**

* Does not change “quantity” - focuses on modifying stock attributes as either a correction or as a further specification of stock attributes (originally receive an item but don’t inspect its condition, later find out its damaged so set inventory condition to damaged)
* Does have the ability to change the item number and other stock attribute(s) on an existing stock unit.

**Overview:**

* Scenario for ecoATM:
  + An item with an IMEI is received and set aside for the grading process.
  + Upon completion of unit grading (Data Wipe & Functional Test), a third party system calls the Inventory Modification API after each step. The payload includes a section of fields to be used to lookup an existing stock unit as well as a section dedicated to outlining desired updates to be made to the existing stock unit. EcoATM provides the IMEI, Item Number, and Location in the lookup section, and may change Item, IMEI, and other static stock unit attributes (see mapping section for specific details.)
  + Deposco finds that stock unit that matches the criteria provided in the lookup section of the payload and performs updates to the existing stock unit if it is found. If no matching stock unit is found on the lookup criteria OR multiple matching stock units are found on the lookup criteria, no updates will be performed and an error response will be provided. Additionally if there is an outstanding pick task against the stock unit found on lookup criteria, the update will be rejected.
  + Next, Deposco will record the inventory grading change with the appropriate inventory and activity tracking records for inventory reporting and auditing purposes.

**Inventory Modification Endpoint and Payload example:**

An API that works synchronous - giving back success or fail info - one at a time.

* **Inventory Modification Payload Concept:**

A screenshot of a computer program

AI-generated content may be incorrect.

* **Inventory Modification example with facility and location in Path parameters:**
  + POST /inventory/facility/{facilityid}/location/{locationid}/inventoryModification

* **Additional example for using Inventory Modification with facility and location in Path parameters:**

A screenshot of a computer

AI-generated content may be incorrect.

**Core Service Business Logic:**

* Given a specific Location in Deposco > match to find the existing inventory record(s) (ie the StockUnit records tied to a Location)
  + Note:  the inputs to this service does give the specific location but does not give the specific StockUnit identifier(s), no outside system is able to keep up with how Deposco merges and splits StockUnit records.
  + Instead - they will tell you the item/pack info for you to use to find the right StockUnit records.
* Once the “correct” StockUnit records are found for the item to modify > attempt to make any of the modifications requested.
  + Note: there is a validation step to verify the request can be initiated
    - For example: You may modify a specified quantity of the item - meaning StockUnit merge/split logic has to be performed in some cases (existing SU of 100 units > only (14 of 100) are modified).
* Lastly - write Inventory and Activity Tracking records

**Validation:**

* Do not allow to set or change Location or Facility
  + Inventory modification API does NOT allow a location change (or subsequently a facility change). This should be its own separate action
    - The intended use case is to modify Inventory “attributes” only.
  + ecoATM will use a field to display the staged/graded/testing Location
* Do not allow to set or change Container, any fields related to LPN numbers or container numbers
  + Do not allow to set or change the allocated order field on StockUnit
* Do not allow to set or change the su.number (auto-generated) field
* Do not allow to set or change related entity fields like su.receiptLine, su.allocatedOrderLine, su.allocatedOrder, su.holdEventDetail, su.company, su.createdby, su.updatedby, su.createddate, su.updateddate:
  + Regarding the “createdDate & updatedDate” - the date/time of the API call will populate these fields since DEP doesn’t allow “random” date time values.

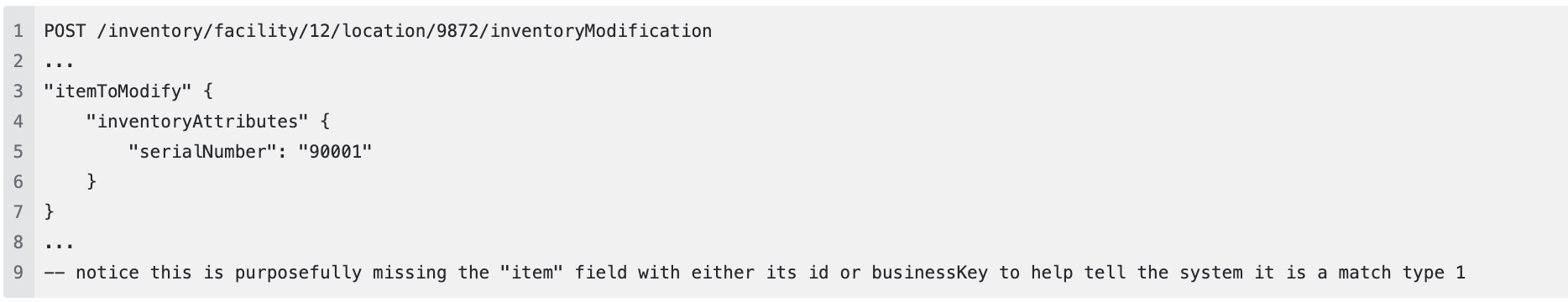
**Match to find the existing inventory record to change:**

Important: If you cannot find a match, do not create new StockUnits. Send back error message “inventory records not found for item <item> in location <location>”

* Two types of logic to match stock in a location
  + **Type 1** - Location and Serial Number with NO Item identifier
  + **Type 2** - Location, Item, and Item Attributes

**Logic:**

* Start with **type 1** matching: Check if the inventory modification payload has only provided the Serial Number attribute and did NOT provide the item identifier field (you get location from the path)
* Otherwise, move to **type 2** - assume you have to use the item identifier, the item attributes, and the location
* example of **type 1:**



In **type 1**, you know the exact location from the path parameter and you know the serial number to look for in that location. ie (select \* from STOCK\_UNIT where location\_id=9872 and serial\_nbr=90001)

* IF the stock unit record is matched that has that location and serial number - you know which stock unit record to proceed to modify
  + If it is a **type 1** match and the location is not found and/or the serial number not found.
  + Give back error message: “Error: inventory records not found for serial number <serial> in location <location locator>”
* Note: you can’t use match **type 1** to modify a stock unit with NO serial number to set a serial number value. There would be no serial number or item number to identify a match.
  + Instead, you’ll use match **type 2** for this scenario, but in that case, all other fields have to match

A **type 2** match - Deposco expects the “item” identifier, the pack quantity, then any additional fields/attributes that is required.

* Example of matching criteria:
  + (2) Records exist in the system today:
* Location=PickLoc3, Item=ITEMA, quantity=3, inventoryCondition=poor
* Location=PickLoc3, Item=ITEMA, quantity=1, inventoryCondition=<null>

**Inv modification API call 1:**

* Modify (2) units of ITEMA, inventoryCondition=poor, location=PickLoc3
  + Should match with the “first” record above and update it

**Inv modification API call 2:**

* Modify (2) units of ITEMA, inventoryCondition=excellent, location=PickLoc3 to change the “inventoryCondition” from excellent to poor.
  + This will NOT find any existing “inventoryCondition” = excellent records
  + Return error that no inventory found

**Inv modification API call 3:**

* Modify (1) unit of ITEMA, inventoryCondition=<null>, location=PickLoc3
  + Should match with the “second” record above and update it.
  + IF it asked for a quantity of (2 or higher)
    - Return error message because only a SU with qty (1) is found

The field “**inventoryCondition**” is used here as an example. Here are all of the matching criteria fields to decide if an existing stock unit is updated or a new one is created:

* adjustmentFacility, adjustmentLocation, number (item number), serialNumber, lotNumber, inventoryAttribute1, inventoryAttribute2, inventoryCondition, expirationDate, bornOnDate
* WMS should already have similar logic on how it treats stock units and stock unit splitting based on matching all of these fields

**Type 2** match WITH a Serial Tracked Item with a specific serial number

* The **type 1** match is specifically when the item identifier is NOT provided but the specific serial number is provided
* It is valid for a user to use **type 2** with a serial tracked item - where they provide the item id, the pack qty, and the specific serial number in the inventory attributes
  + However, in this case with **type 2**, ALL of the inventory attributes have to match between the payload and actual stock unit.
  + So if the actual stock unit is (su.id=8844, item=ITEMA, packQty=1, serialNumber=800002, inventoryCondition=Poor) - then the payload needs to exactly match these conditions.
    - For example - Including “inventoryCondition”: “Poor” in the payload.
    - This is the value of having a **type 1** match, it lets a caller give less info to match as a shortcut.

**Attempt to make any of the Modifications:**

Any combination of all of these fields might be requested to be changed

* **Quantity**
  + Need to accept a quantity so you can tell Deposco how much quantity to modify.
  + For example: A StockUnit with Qty (100) but you are only changing (14) qty
    - Deposco has to understand how to split the stockunit: (1) with 86 qty and (2) with 14 qty that included with the modifications.
* **Status**
  + Change the StockUnit.status value of ‘On Hold’, ‘Damaged’, ‘Inspection’, ‘Quarantine’, or blank
* **Item and pack**
  + Allow to change from one item/pack combo to a different item/pack combo - used to resolve errors
* **Serial number**
  + This might be changing a StockUnit record that has serialnumber=89048464363 to a different number
  + Or - changing a stockunit record that has serialnumber=blank to a non-blank value
* **inventoryAttribute1, inventoryAttribute2, inventoryCondition, expirationDate, lotNumber, bornOnDate, unitPrice**
  + Same findings as “serial number” change from a non-blank value to another value or a blank value to a non-blank value.

**Write Activity and Inventory Tracking records:**

* In Deposco - if the inventory modification is successful - it should act similar as if a user changed the StockUnit record from the UI.

* Write inventory ledger transactions:
  + Basically a DR and CR pair showing the from and to from the modification action:
    - actionType=ADJ
    - actionName=Manual Adjustment
    - reasonCode=from API reasonCode field passed in string value
    - transactionNumber= transactionNumber pulled from API data to help with troubleshooting between systems (if blank, generate our own internal transactionNumber)
    - customField1 (in IGL) or customAttribute1 (in InvLedger) = adjustmentUser pulled from API data

* Write Activity Log records:
  + Basically a minus quantity and a plus quantity pair showing the from and to from the modification action, that links to inventory ledger record
    - activityDefinition=MAD
    - processName=DeposcoAPI
    - activityName= the activityName pulled from API data (optional field on API so populate with ‘MAD’ if not provided)
    - transactionNumber= transactionNumber pulled from API data to help with troubleshooting between systems

* On the inventory ledger and activity log records:
  + The activity log minus quantity record and the inventory ledger DR record should capture the “from” or “before” item and item attributes before the modification
  + The activity log plus quantity records and the inventory ledger CR record should capture the “to” or “after” item and item attributes after the modification.

# **Customer Agreement**

Your signature is required as proof of understanding and acceptance of the design document as described above. Upon acknowledgement of your approval, Deposco will engage in development of the work stated in this document and the design specification. Please review this document in its entirety. Upon approval, please sign, date and return.

| **Deposco, Inc.** 11605 Haynes Bridge Road Suite 200 Alpharetta, GA 30009  By: | |
| --- | --- |
| Name: |  |
| Title: |  |
| Date: |  |

| **ecoATM**  10121 Barnes Canyon Rd.  San Diego, CA 92121  By: | |
| --- | --- |
| Name: |  |
| Title: |  |
| Date: |  |