**Zoho Inventory API v1: Integration Guide for Developers**

**1. Introduction to Zoho Inventory API v1 Integration**

**1.1. Purpose of the API and this Guide**

The Zoho Inventory API v1 serves as a programmatic interface to the comprehensive suite of functionalities available within the Zoho Inventory web client. It empowers developers to automate various aspects of inventory management, order processing, and other critical business workflows, thereby enhancing efficiency and enabling custom integrations.

This guide is specifically designed to provide developers with a consolidated understanding of key Zoho Inventory API v1 sections pertinent to common integration scenarios. It delves into authentication mechanisms, the management of items (including specific use cases like "crates"), procedures for stock adjustments, sales order processing (particularly for "dispatch orders"), handling of packages (in the context of "donation batches"), and the nuances of organization management. The information is structured to align with typical developer functions and use cases, such as those involving getZohoAccessToken, createZohoInventoryItem, getZohoInventoryItem, updateZohoInventoryStock, donationFunction, reportFunction, and makeZohoInventoryRequest.

**1.2. API Architecture: REST Principles and HTTP Rules**

The Zoho Inventory API v1 is architected based on Representational State Transfer (REST) principles. This architectural style ensures predictable and resource-oriented URLs, simplifying the development of applications that interact with the API. The API strictly adheres to standard HTTP rules and methods, such as GET for retrieving data, POST for creating resources, PUT for updating existing resources, and DELETE for removing resources. This adherence to HTTP standards allows for a broad range of HTTP clients and development languages to be used for interacting with the API, promoting flexibility and ease of integration.

**1.3. Data Centers and Base API URIs**

Zoho Inventory's global infrastructure is distributed across multiple data centers to ensure optimal performance and data residency. Consequently, the API is accessible via different base URIs, each corresponding to a specific data center. It is imperative for developers to use the correct base API URI that aligns with the data center where their Zoho Inventory organization is hosted.

The primary base API URIs are :

* United States: https://www.zohoapis.com/inventory/
* Europe: https://www.zohoapis.eu/inventory/
* India: https://www.zohoapis.in/inventory/
* Australia: https://www.zohoapis.com.au/inventory/
* Japan: https://www.zohoapis.jp/inventory/
* Canada: https://www.zohoapis.ca/inventory/
* China: https://www.zohoapis.com.cn/inventory/
* Saudi Arabia: https://www.zohoapis.sa/inventory/

The existence of multiple data centers has a direct implication for the design of any generalized API request function, such as a makeZohoInventoryRequest function. This function must be configurable to target the correct domain based on the organization's location. Hardcoding a specific domain could lead to persistent connection failures or data access issues if the target organization resides in a different data center. An incorrect base URI will result in requests being misrouted, potentially leading to authentication failures or "resource not found" errors, even if a valid organization\_id for a different region is provided. This makes the correct configuration of the base API URI a critical preliminary step for ensuring a stable and reliable integration.

**2. Authenticating with Zoho Inventory API (OAuth 2.0)**

**2.1. Overview of OAuth 2.0 Protocol**

The Zoho Inventory API v1 utilizes the OAuth 2.0 protocol for all authorization and authentication processes. OAuth 2.0 is an industry-standard framework that enables third-party applications, such as custom-developed Lambda functions, to access user data hosted on a service provider (Zoho Inventory, in this case) without requiring the application to handle or store the user's actual login credentials (username and password) directly. This delegation model significantly enhances security by limiting the exposure of sensitive credentials. The core of the OAuth 2.0 flow involves the client application obtaining an access token, which is then presented with each API request to authenticate and authorize the requested operation.

**2.2. Step 1: Client Registration**

Before an application can initiate the OAuth 2.0 authentication flow, it must be registered as a new client within the Zoho Developer Console, accessible at https://accounts.zoho.com/developerconsole. This registration process involves providing essential details about the application, such as a Client Name, Client Domain, and one or more Authorized Redirect URIs.

Upon successful registration, Zoho will generate a unique Client ID and a Client Secret. These two pieces of information are critical credentials for the OAuth 2.0 process and must be treated with the utmost confidentiality, stored securely, and never exposed in client-side code or insecure channels.

For server-side applications like AWS Lambda functions (e.g., a getZohoAccessToken function), parameters such as "Client Domain" and "Authorized Redirect URIs" are primarily relevant during the initial, often manual, setup phase required to obtain the first authorization code. Once this initial setup is complete and a refresh token is acquired, the Lambda function will predominantly use the Client ID and Client Secret for non-interactive token operations, such as the refresh token grant type. Therefore, the secure storage and management of the Client ID and Client Secret (for instance, using services like AWS Secrets Manager) are paramount for the operational security of the Lambda function.

**2.3. Step 2: Generating Authorization Code (Grant Token)**

The generation of an authorization code (also referred to as a grant token) is typically the first active step in the OAuth 2.0 flow, especially for flows involving user interaction or for the initial acquisition of a refresh token. The client application initiates this step by redirecting the user's browser to Zoho's authorization server endpoint: https://accounts.zoho.com/oauth/v2/auth?.

This request must include several mandatory query parameters :

* scope: Defines the permissions the application is requesting (e.g., ZohoInventory.items.READ). Multiple scopes can be requested, separated by commas.
* client\_id: The Client ID obtained during client registration.
* response\_type: Must be set to code, indicating that an authorization code is being requested.
* redirect\_uri: One of the Authorized Redirect URIs specified during client registration. Zoho will redirect the user back to this URI after consent is given.

Crucially, for applications like a Lambda-based getZohoAccessToken function that require long-term, non-interactive access, the access\_type=offline parameter must be included in this authorization request. This parameter instructs Zoho to issue a refresh token alongside the access token, which is essential for maintaining access after the initial short-lived access token expires. The prompt=consent parameter can also be used to ensure the user is explicitly asked for consent, even if they have previously granted it.

If the user grants the requested permissions via the consent page, Zoho's authorization server redirects the user's browser back to the specified redirect\_uri. This redirection includes the authorization\_code as a query parameter. This code is short-lived (typically valid for 60 seconds or a configurable duration of up to 10 minutes) and single-use. For a getZohoAccessToken Lambda function designed for ongoing, unattended operation, this step is usually performed once manually during setup to obtain the initial refresh token. This refresh token is then securely stored and used by the Lambda for all subsequent token generation needs.

**2.4. Step 3: Generating Access and Refresh Tokens**

Once the client application has received the authorization\_code, it must exchange this code for an access\_token (and, if requested, a refresh\_token). This exchange is done by making a server-to-server POST request to Zoho's token endpoint: https://accounts.zoho.com/oauth/v2/token?.

The body of this POST request must include the following mandatory parameters :

* code: The authorization\_code obtained in the previous step.
* client\_id: The application's Client ID.
* client\_secret: The application's Client Secret.
* redirect\_uri: The same redirect\_uri used in the authorization code request.
* grant\_type: Must be set to authorization\_code, indicating that an authorization code is being exchanged for tokens.

If the request is successful, Zoho's token server will respond with a JSON payload containing :

* access\_token: The token used to authenticate API requests. Access tokens are typically short-lived, usually valid for one hour. The exact validity period is indicated by the expires\_in field in the response (in seconds or milliseconds).
* refresh\_token: If access\_type=offline was specified in the authorization code request, a refresh\_token will also be provided. Refresh tokens are long-lived and can be used to obtain new access tokens when the current one expires, without requiring further user interaction.
* expires\_in: The lifetime of the access token in seconds.
* token\_type: Typically Bearer.

The limited lifespan of the access token (commonly one hour) is a fundamental characteristic that necessitates a robust token management strategy within any application that interacts with the API over extended periods. A function like getZohoAccessToken must not only fetch an access token initially but also manage its lifecycle effectively. This includes mechanisms for detecting token expiry and utilizing the refresh token to obtain a new access token seamlessly.

**2.5. Step 4: Refreshing Access Tokens (Core for getZohoAccessToken Lambda)**

Given that access tokens expire, a critical function for any long-running integration, such as a getZohoAccessToken Lambda, is the ability to refresh the access token using the previously obtained refresh\_token. This process allows the application to maintain continuous authorized access to the Zoho Inventory API without requiring repeated user authentication.

To refresh an access token, the getZohoAccessToken function will make a POST request to Zoho's token endpoint (https://accounts.zoho.com/oauth/v2/token?) with the following parameters :

* refresh\_token: The stored refresh\_token.
* client\_id: The application's Client ID.
* client\_secret: The application's Client Secret.
* redirect\_uri: While the necessity of redirect\_uri for the refresh\_token grant type can vary across OAuth implementations, it is good practice to include the same URI that was used during the initial client registration and authorization code flow if the documentation implies or requires it.
* grant\_type: Must be set to refresh\_token.

The successful response to this request will provide a new access\_token and its expires\_in duration. In some OAuth 2.0 implementations, a new refresh\_token might also be issued (though often the original refresh token remains valid). Zoho's documentation indicates a limit of 20 active refresh tokens per user; if this limit is exceeded by repeatedly going through the full consent process, the oldest refresh tokens may be automatically invalidated. Therefore, it is crucial to securely store and consistently use the most recently obtained valid refresh token.

The getZohoAccessToken Lambda function should be designed to manage this refresh process proactively. It could, for example, store the expiry time of the current access token and automatically request a new one shortly before it expires. Alternatively, it could attempt a refresh operation upon receiving an authentication error (e.g., HTTP 401 Unauthorized) from a Zoho Inventory API call. Secure storage of the refresh\_token itself (e.g., in AWS Secrets Manager) is of paramount importance, as it represents long-term access to the user's data.

**2.6. Step 5: Using Access Tokens in API Requests**

Once a valid access\_token has been obtained (either directly or via a refresh), it must be included in every subsequent request to the Zoho Inventory API to authenticate the application. The access token is sent in the Authorization HTTP header.

The format for this header is : Authorization: Zoho-oauthtoken {access\_token} (Replace {access\_token} with the actual access token string).

Any function responsible for making API calls, such as the user-mentioned makeZohoInventoryRequest function, must be designed to correctly construct and include this Authorization header with every request sent to Zoho Inventory API endpoints.

**2.7. Step 6: Revoking Refresh Tokens**

If a refresh\_token is compromised, or if the application's access to Zoho Inventory is no longer required (e.g., the integration is being decommissioned), the refresh\_token should be revoked to invalidate it and prevent any further unauthorized access. This is an essential security best practice.

A refresh token can be revoked by making a POST request to Zoho's token revocation endpoint: https://accounts.zoho.com/oauth/v2/token/revoke?. The request must include the token parameter, where the value is the refresh\_token to be revoked.

Implementing a process or standard operating procedure for revoking refresh tokens is a crucial component of the security lifecycle management for any integration. This ensures that access rights are properly terminated when they are no longer needed or if a security incident is suspected.

**2.8. Essential OAuth Scopes for Inventory Operations**

OAuth scopes are used to define the specific permissions that an access token grants to an application. When requesting an authorization code, the scope parameter must specify which parts of the user's Zoho Inventory data the application needs to access and what operations (create, read, update, delete) it needs to perform. Requesting only the necessary scopes adheres to the principle of least privilege, which is a fundamental security best practice.

Zoho Inventory provides a granular set of scopes for different modules and operations. Examples include :

* For Organizations: ZohoInventory.settings.CREATE, ZohoInventory.settings.READ, ZohoInventory.settings.UPDATE.
* For Items: ZohoInventory.items.CREATE, ZohoInventory.items.READ, ZohoInventory.items.UPDATE, ZohoInventory.items.DELETE.
* For Item Adjustments: ZohoInventory.inventoryadjustments.CREATE, ZohoInventory.inventoryadjustments.READ, ZohoInventory.inventoryadjustments.UPDATE, ZohoInventory.inventoryadjustments.DELETE.
* For Sales Orders: Scopes such as ZohoInventory.salesorders.CREATE, ZohoInventory.salesorders.READ, ZohoInventory.salesorders.UPDATE, ZohoInventory.salesorders.DELETE are implied by the available API actions.
* For Packages: ZohoInventory.packages.CREATE, ZohoInventory.packages.READ, ZohoInventory.packages.UPDATE, ZohoInventory.packages.DELETE.
* A general scope like ZohoInventory.FullAccess.all is also available, granting extensive permissions, but should be used judiciously.

The following table outlines essential OAuth scopes relevant to the user functions mentioned in the query, providing clarity for developers when configuring the scope parameter during the OAuth authorization process:

**Table: Essential OAuth Scopes for Zoho Inventory API**

|  |  |  |
| --- | --- | --- |
| **Scope Name** | **Description** | **Relevant User Functions** |
| ZohoInventory.items.CREATE | Allows the creation of new items in Zoho Inventory. | createZohoInventoryItem |
| ZohoInventory.items.READ | Allows the retrieval of item details and lists. | getZohoInventoryItem |
| ZohoInventory.inventoryadjustments.CREATE | Allows the creation of item adjustments for managing stock levels. | updateZohoInventoryStock, createZohoInventoryItem (if setting initial stock via adjustment) |
| ZohoInventory.salesorders.CREATE | Allows the creation of new sales orders (e.g., dispatch orders). | Creating dispatch orders |
| ZohoInventory.salesorders.READ | Allows the retrieval of sales order details and lists. | Managing/tracking dispatch orders |
| ZohoInventory.packages.CREATE | Allows the creation of new packages (e.g., donation batches linked to SOs). | donationFunction (for creating donation batches as packages) |
| ZohoInventory.packages.READ | Allows the retrieval of package details and lists. | reportFunction (for reporting on donation batches) |
| ZohoInventory.settings.READ | Allows reading organization settings and details (e.g., listing orgs). | makeZohoInventoryRequest (if needing to fetch organization details dynamically) |
| ZohoInventory.FullAccess.all | Grants comprehensive access to all modules within Zoho Inventory. | General purpose, but its use should be carefully evaluated against specific needs. |

Export to Sheets

This table directly supports developers in requesting appropriate permissions, ensuring that the getZohoAccessToken function can obtain tokens with the correct entitlements for the various inventory operations the application will perform.

**3. Working with Organizations**

**3.1. The organization\_id Parameter**

A fundamental requirement for interacting with the Zoho Inventory API is the inclusion of the organization\_id parameter in every API request. Zoho Inventory is designed to support multiple organizations under a single user account, and each organization functions as an independent entity with its own data, base currency, settings, contacts, and reports. The organization\_id is a unique, server-generated string that identifies the specific organization whose data the API request intends to access or modify.

Therefore, any function designed to make requests to the Zoho Inventory API, such as the makeZohoInventoryRequest function, must ensure that it appends the organization\_id as a query parameter to the request URL. For example: ...?organization\_id={your\_organization\_id}.

**3.2. Retrieving Organization IDs**

The organization\_id for a particular organization can be obtained in a few ways:

1. **Via API:** By making a GET request to the /organizations endpoint (e.g., https://www.zohoapis.com/inventory/v1/organizations). The response to this request will be a list of all organizations accessible to the authenticated user, with each entry including the organization\_id, name, is\_default\_org, contact\_name, email, and other relevant details. The OAuth scope required for this operation is typically ZohoInventory.settings.READ.
2. **Via Admin Console:** The organization\_id can also be found within the Zoho Inventory web application, usually on a "Manage Organizations" page in the admin console or settings section.

For server-side applications or automated scripts, such as Lambda functions, that are typically configured to work with a single, specific Zoho Inventory organization, the organization\_id is often best treated as a static configuration parameter. This ID can be determined once (either through the API or the admin console) and then stored securely as an environment variable or a configuration setting for the application. Dynamically fetching the organization\_id using the /organizations endpoint before every business-related API call (e.g., creating an item or sales order) would introduce unnecessary latency and consume API rate limits. The /organizations endpoint is more appropriate for initial setup, for applications that need to allow users to select from multiple organizations, or for administrative tools that manage organizations themselves.

**3.3. Making API Requests with organization\_id**

All API requests targeting specific resources within an organization (such as items, sales orders, item adjustments, etc.) must include the organization\_id as a query parameter. The makeZohoInventoryRequest function should be designed to systematically append this parameter to the base URL and resource path for any given API call.

For example, a request to list items would look like: GET https://www.zohoapis.com/inventory/v1/items?organization\_id={your\_organization\_id}

And a request to create an item would be: POST https://www.zohoapis.com/inventory/v1/items?organization\_id={your\_organization\_id} (with the item data in the request body).

Failure to include a valid organization\_id will result in API errors, as Zoho Inventory will not know which organization's data to operate on.

**4. Managing Items (including Crates)**

**4.1. Overview of the Items API**

The Items API in Zoho Inventory provides a comprehensive set of operations for managing products, services, or any other entities that a business buys, sells, or tracks. These are generically referred to as "items". Items can be physical goods (inventory items), non-inventory items, or services. They can have associated stock levels, purchase and sales prices, SKUs, and various other attributes.

The user's requirement to manage "crates" can be directly and effectively mapped to the "items" resource in Zoho Inventory. Each type of crate would be set up as an individual item record, allowing for distinct tracking of stock, attributes, and transactions related to that crate type.

**4.2. Creating an Item (for createZohoInventoryItem and "Crates")**

To create a new item, such as a new type of crate, a POST request is made to the /items endpoint. The OAuth scope required for this operation is ZohoInventory.items.CREATE.

When creating a "crate" as an item, several parameters are particularly relevant to ensure proper setup and tracking. The createZohoInventoryItem function would need to construct a JSON payload with these parameters. The locations array within the item creation payload is particularly noteworthy. It offers an efficient method to define initial stock levels for the new "crate" across one or more inventory locations simultaneously with the item's creation. This capability can simplify the initial provisioning process, potentially eliminating the need for a subsequent, separate stock adjustment transaction (via updateZohoInventoryStock or creating an item adjustment) immediately after the createZohoInventoryItem call. This is especially useful when a new crate type is introduced with a known starting quantity at specific warehouses.

The following table details key parameters from the item creation API, highlighting their relevance for defining "crates":

**Table: Key Parameters for Item (Crate) Creation (POST /items)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Data Type** | **Required?** | **Description & Relevance for "Crates"** |
| name | string | Yes | The descriptive name of the crate (e.g., "Standard Wooden Crate - Large", "Reusable Plastic Crate Type B"). |
| sku | string | No | Stock Keeping Unit. A unique identifier for the specific crate type. Highly recommended for accurate inventory management and barcode scanning. |
| item\_type | string | No | Defines how the item is handled. For crates whose stock is tracked, this should be set to inventory. Other options include sales, purchases. |
| product\_type | string | No | Specifies if the item is a physical good or a service. For crates, this would typically be goods. |
| unit | string | No | The unit of measurement for the crate (e.g., "pcs", "units", "each"). |
| is\_taxable | boolean | No | Indicates whether the crate is subject to sales tax. |
| description | string | No | A more detailed description of the crate, potentially including dimensions, material, capacity, or usage notes. |
| rate | double | No | The sales price per crate, if the crates are sold to customers or charged out. |
| purchase\_rate | double | No | The purchase cost per crate, if the crates are procured from external vendors. |
| reorder\_level | double | No | The minimum stock level for this crate type that, when reached, triggers a reorder or replenishment process. |
| inventory\_account\_id | long | No | Unique ID of the inventory asset account associated with this item. |
| locations | array | No | An array of location-specific stock information. Each object in the array can specify a location\_id, initial\_stock (the starting quantity at that location), and initial\_stock\_rate (the initial value per unit at that location). This is key for setting up initial stock levels during creation. |
| custom\_fields | array | No | Allows for storing additional, user-defined attributes specific to the crate (e.g., {"customfield\_id": "your\_cf\_id", "value": "Heavy Duty Rating"}). |

By populating these fields appropriately, the createZohoInventoryItem function can ensure that crates are accurately represented in Zoho Inventory, facilitating effective tracking and management.

**4.3. Retrieving Item Details (for getZohoInventoryItem)**

The Zoho Inventory API provides several endpoints for retrieving item information, which would be utilized by a getZohoInventoryItem function:

* **Retrieve a specific item:** To fetch the details of a single item (crate) when its unique item\_id is known, a GET request is made to /items/{item\_id}. The required OAuth scope is ZohoInventory.items.READ. The response will contain all attributes of the specified item, including stock levels if it's an inventory item.
* **List all items:** To retrieve a list of all items (crates) in the organization, a GET request is made to /items. This also requires the ZohoInventory.items.READ scope. The list can be paginated and filtered using various query parameters (e.g., by status, item type, or custom fields) to refine the results.
* **Bulk fetch item details:** If details for multiple specific items are needed, and their item\_ids are known, a GET request to /itemdetails can be used. This endpoint allows fetching details for several items in a single API call by passing their IDs, which can be more efficient than making multiple individual requests. The scope is ZohoInventory.items.READ.

**4.4. Updating an Item**

Existing items (crates) can be modified using a PUT request to the /items/{item\_id} endpoint. The OAuth scope for this operation is ZohoInventory.items.UPDATE. The request body would contain the fields to be updated, similar in structure to the item creation payload. This allows for changes to attributes like name, SKU, description, rates, reorder levels, and custom fields.

**4.5. Other Item Management Operations**

Beyond creation, retrieval, and updates, the Items API supports other management functions :

* **Delete an item:** DELETE /items/{item\_id} (Scope: ZohoInventory.items.DELETE).
* **Mark as active:** POST /items/{item\_id}/active (Scope: ZohoInventory.items.CREATE). This makes an item available for transactions.
* **Mark as inactive:** POST /items/{item\_id}/inactive (Scope: ZohoInventory.items.CREATE). This prevents an item from being used in new transactions but retains its history.
* **Update custom field in existing items:** PUT /item/{item\_id}/customfields (Scope: ZohoInventory.items.UPDATE).
* **Delete an item image:** DELETE /items/{item\_id}/image (Scope: ZohoInventory.items.DELETE).

These operations provide a comprehensive toolkit for managing the lifecycle of items (crates) within Zoho Inventory.

**5. Making Item Adjustments (Stock Control)**

**5.1. Overview of Item Adjustments**

Item Adjustments in Zoho Inventory are a crucial mechanism for recording changes in stock levels that occur outside of standard sales or purchase transactions. These adjustments are used to account for various scenarios such as:

* Physical inventory count discrepancies (cycle counting).
* Damaged, lost, or stolen goods.
* Items used for internal consumption or samples.
* Donations of stock.
* Initial stock loading or corrections to existing stock quantities.

This functionality is directly relevant for implementing an updateZohoInventoryStock function, which aims to set or update the stock levels of items like "crates."

**5.2. Creating an Item Adjustment (for updateZohoInventoryStock)**

To make an adjustment to the stock level of one or more items (crates), a POST request is sent to the /inventoryadjustments endpoint. The required OAuth scope for this operation is ZohoInventory.inventoryadjustments.CREATE.

The updateZohoInventoryStock function would use this endpoint to reflect changes in crate quantities. A critical consideration when using this API is the interpretation of the quantity\_adjusted field within the line\_items array. This field typically represents the *change* in stock (delta) rather than the new absolute stock level. If the updateZohoInventoryStock function is designed to set the stock to a specific absolute value (e.g., "set stock of Crate A to 100 units"), it must first determine the current stock level of that item (potentially by querying the item's details via GET /items/{item\_id}, which often includes stock information per location). Then, it must calculate the difference: delta = desired\_absolute\_stock - current\_stock. This calculated delta is then used as the quantity\_adjusted value (positive for an increase, negative for a decrease). This two-step approach (fetch current stock, calculate delta, then post adjustment) ensures that the final stock level accurately reflects the intended absolute quantity. If the function's input is already the delta (e.g., "add 10 units of Crate A"), then this calculation is not needed.

The following table outlines the key parameters for creating an item adjustment, particularly relevant for the updateZohoInventoryStock function:

**Table: Key Parameters for Item Adjustment Creation (POST /inventoryadjustments)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Data Type** | **Required?** | **Description & Relevance for updateZohoInventoryStock** |
| date | string | Yes | The date on which the stock adjustment is recorded. Should reflect the actual date of the stock change. |
| reason | string | Yes | A descriptive reason for the stock adjustment (e.g., "Annual Stock Count Correction", "Initial Stock Load for New Crates", "Disposal of Damaged Crates", "Stock Update via Integration"). |
| adjustment\_type | string | Yes | Specifies the type of adjustment. For changing stock quantities, this must be set to quantity. The other option is value for value-based adjustments. |
| line\_items | array | Yes | An array of objects, where each object details a specific item (crate) being adjusted and the quantity of adjustment. |
| item\_id | long | Yes (within line\_items) | The unique ID of the crate (item) whose stock is being updated. |
| quantity\_adjusted | double | Yes (within line\_items if adjustment\_type is quantity) | The change in quantity for the specified crate. A positive value increases stock, a negative value decreases it. As discussed, if setting an absolute stock level, this should be NewAbsoluteStock - CurrentStock. |
| description | string | No | Optional additional notes or comments providing more context about the stock adjustment. |
| reference\_number | string | No | An optional internal reference number for the adjustment transaction, useful for auditing or tracking. |
| location\_id | string | No | The ID of the specific warehouse or location where the stock adjustment is occurring. This is critical if inventory is managed across multiple locations, as it ensures the adjustment impacts the correct location's stock record. |

Using these parameters correctly will enable the updateZohoInventoryStock function to accurately reflect changes in crate inventory within Zoho Inventory.

**5.3. Retrieving and Updating Item Adjustments**

The API also allows for the management of existing item adjustments :

* **List all item adjustments:** GET /inventoryadjustments (Scope: ZohoInventory.inventoryadjustments.READ). This can be used to retrieve a history of adjustments, filterable by various criteria.
* **Retrieve a specific item adjustment:** GET /inventoryadjustments/{inventory\_adjustment\_id} (Scope: ZohoInventory.inventoryadjustments.READ). Fetches details of a single adjustment.
* **Update an item adjustment:** PUT /inventoryadjustments/{inventory\_adjustment\_id} (Scope: ZohoInventory.inventoryadjustments.UPDATE). Allows modification of certain fields of an existing adjustment, though restrictions may apply depending on the adjustment's status.
* **Delete an item adjustment:** DELETE /inventoryadjustments/{inventory\_adjustment\_id} (Scope: ZohoInventory.inventoryadjustments.DELETE). Removes an adjustment record.

These operations provide control over the lifecycle of item adjustment records.

**6. Managing Sales Orders (Dispatch Orders)**

**6.1. Overview of Sales Orders API**

In Zoho Inventory, Sales Orders are documents that confirm a sale to a customer, detailing the items ordered, quantities, prices, and shipping information. This resource maps directly to the user's concept of "dispatch orders," representing an instruction to prepare and send out goods (crates, in this context) to a specified destination or customer. The Sales Orders API allows for the creation and management of these dispatch orders programmatically.

**6.2. Creating a Sales Order (Dispatch Order)**

To create a new Sales Order, which will represent a dispatch order, a POST request is made to the /salesorders endpoint. The implied OAuth scope for this operation would be ZohoInventory.salesorders.CREATE.

When creating a dispatch order, several parameters are crucial for accurate record-keeping and inventory management. Of particular importance is the location\_id parameter. If the business operates with multiple warehouses or inventory storage locations, specifying the location\_id from which the items (crates) will be dispatched is essential. This ensures that stock levels are correctly depleted from the appropriate source location, maintaining the integrity of location-specific inventory data. If location\_id is omitted, Zoho Inventory might deduct stock from a default location, which could lead to inaccuracies if the dispatch actually originates elsewhere.

The table below highlights key parameters for creating a Sales Order intended as a dispatch order:

**Table: Key Parameters for Sales Order (Dispatch Order) Creation (POST /salesorders)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Data Type** | **Required?** | **Description & Relevance for "Dispatch Orders"** |
| customer\_id | string | Yes | The unique ID of the customer or entity to whom the crates are being dispatched. |
| date | string | No (often defaults to current) | The date the dispatch order is created or recorded. |
| shipment\_date | string | No | The planned or actual date on which the crates are scheduled to be shipped or dispatched. |
| line\_items | array | Yes | An array of objects, where each object details a type of crate being dispatched. Each line item typically includes item\_id, quantity, rate (if applicable), unit, and tax information. |
| item\_id | string | Yes (within line\_items) | The unique ID of the specific crate (item) being included in the dispatch. |
| quantity | double | Yes (within line\_items) | The number of units of this particular crate type being dispatched. |
| location\_id | string | No | **Crucial for accurate stock:** The ID of the warehouse or inventory location from which the crates are being dispatched. Ensures correct stock deduction from the source. |
| shipping\_address\_id | string | No | The unique ID of the shipping address (from the customer's contact record) where the crates are to be sent. If not provided, the default shipping address for the customer might be used. |
| billing\_address\_id | string | No | The unique ID of the billing address for this order. |
| delivery\_method | string | No | The method of delivery or dispatch (e.g., "Internal Fleet Transfer", "Third-Party Courier", "Customer Pickup"). |
| notes | string | No | Any special instructions, comments, or notes relevant to this dispatch order (e.g., handling instructions, delivery contact). |
| terms | string | No | Terms and conditions associated with the dispatch or order. |
| salesorder\_number | string | Yes (if ignore\_auto\_number\_generation is true) | A unique number for the dispatch order. If ignore\_auto\_number\_generation is false or omitted, Zoho Inventory typically auto-generates this number. |

By carefully constructing the payload with these parameters, applications can programmatically create dispatch orders that accurately reflect the intended shipment and integrate seamlessly with Zoho Inventory's stock tracking.

**6.3. Retrieving and Updating Sales Orders**

The API also provides functionalities to manage existing Sales Orders :

* **List all Sales Orders:** GET /salesorders (Scope: ZohoInventory.salesorders.READ). Retrieves a list of sales orders, which can be filtered by status, customer, date range, etc.
* **Retrieve a specific Sales Order:** GET /salesorders/{salesorder\_id} (Scope: ZohoInventory.salesorders.READ). Fetches the complete details of a single sales order.
* **Update a Sales Order:** PUT /salesorders/{salesorder\_id} (Scope: ZohoInventory.salesorders.UPDATE). Allows modification of existing sales orders, for example, to change quantities, shipment dates, or add notes before dispatch.

**6.4. Status Management for Dispatch Workflow**

Managing the status of a Sales Order is critical for tracking its progress through the dispatch workflow. The API supports several status update operations :

* **Mark as Confirmed:** POST /salesorders/{salesorder\_id}/status/confirmed. This action typically signifies that the order is acknowledged and ready for further processing, such as picking, packing, and dispatch. It can be a key step before a dispatch order is considered active for fulfillment.
* **Bulk Confirm Sales Orders:** POST /salesorders/status/confirmed (with a comma-separated list of salesorder\_ids in the request). Allows confirming multiple orders in one call.
* **Mark as Void:** POST /salesorders/{salesorder\_id}/status/void. This action cancels the sales order, typically used if the order is annulled before dispatch.

Further status changes (e.g., to "Shipped" or "Delivered") are often linked to the creation of related entities like Packages and Shipments within Zoho Inventory, which build upon the Sales Order.

**7. Handling Packages (Donation Batches)**

**7.1. Overview of Packages API**

The Packages API in Zoho Inventory is primarily designed to facilitate the process of grouping line items from confirmed Sales Orders into one or more packages, assigning unique package slip numbers, and preparing these packages for shipment. This functionality is integral to the order fulfillment and shipping stages.

The user intends to manage "donation batches" using this Packages API, with specific functions (donationFunction for creation and reportFunction for retrieval). A significant aspect to understand here is the Packages API's inherent linkage to Sales Orders. The documentation indicates that creating a package requires a salesorder\_id as a query parameter, and the line items within a package refer to so\_line\_item\_id (Sales Order Line Item ID). The available documentation does not describe a direct mechanism for creating packages independently of sales orders or specifically for "donation batches" as a standalone concept.

This structural dependency implies that to use the Packages API for "donation batches," a "donation" must first be conceptualized and recorded as a Sales Order within Zoho Inventory. This Sales Order would detail the items (crates) being donated and their quantities, potentially linked to a specific customer record representing the donation recipient or a generic internal entity for tracking donations. Once this Sales Order exists, the donationFunction can then proceed to create a Package associated with it. This two-step process (Create Sales Order for donation -> Create Package linked to that Sales Order) is a necessary adaptation to align the "donation batch" concept with the existing capabilities of the Packages API.

**7.2. Creating a Package (for donationFunction)**

To create a new package (representing a "donation batch" derived from a Sales Order), a POST request is made to the /packages endpoint. The required OAuth scope is ZohoInventory.packages.CREATE.

As established, this operation requires the salesorder\_id of the parent Sales Order (which represents the donation) to be passed as a query parameter in the request URL (e.g., POST /packages?salesorder\_id={id\_of\_donation\_sales\_order}).

The body of the request should be a JSON payload containing details for the package. The table below outlines key parameters:

**Table: Key Parameters for Package (Donation Batch) Creation (POST /packages)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter Type** | **Parameter Name** | **Data Type** | **Required?** | **Description & Relevance for "Donation Batches" (assuming linked to a Sales Order representing the donation)** |
| Query Parameter | salesorder\_id | string | Yes | The unique ID of the Sales Order that contains the items designated for this donation batch. This Sales Order must be created prior to creating the package. |
| Body Parameter | package\_number | string | No (often auto-generated) | A unique identifier for this specific package or donation batch. Can be user-defined or system-generated. |
| Body Parameter | date | string | Yes | The date on which the package (donation batch) is prepared or recorded. |
| Body Parameter | line\_items | array | Yes | An array of objects, each specifying which line items from the parent Sales Order (and in what quantity) are included in this particular package. This allows a single Sales Order to be split into multiple packages if needed. |
| Body Parameter | so\_line\_item\_id | long | Yes (within line\_items) | The unique ID of a specific line item from the associated Sales Order (identified by salesorder\_id). This links the package content directly to the items on the donation Sales Order. |
| Body Parameter | quantity | float | Yes (within line\_items) | The quantity of the specified so\_line\_item\_id that is being included in *this* package. This must not exceed the unfulfilled quantity of that line item on the Sales Order. |
| Body Parameter | notes | string | No | Any notes or special instructions specific to this donation batch package (e.g., "Fragile Contents - Handle with Care", "Donation for XYZ Initiative"). |
| Body Parameter | custom\_fields | array | No | An array for including any custom field data relevant to the package. Each object would contain customfield\_id and value. This can be used to store additional donation-specific information (e.g., {"customfield\_id": "your\_cf\_id", "value": "Urgent Humanitarian Aid"}). |

The donationFunction would thus first ensure the creation or identification of the relevant Sales Order representing the donation, retrieve its salesorder\_id and the so\_line\_item\_id(s) for the items being donated, and then use these to construct the payload for the POST /packages request.

**7.3. Retrieving Package Details (for reportFunction)**

To retrieve information about packages (donation batches), which would be the role of a reportFunction, the API offers the following endpoints :

* **Retrieve a specific package:** GET /packages/{package\_id} (Scope: ZohoInventory.packages.READ). Fetches the details of a single package using its unique package\_id. The response includes information about the associated sales order, customer, line items in the package, status, and custom fields.
* **List all packages:** GET /packages (Scope: ZohoInventory.packages.READ). Retrieves a list of all packages. This endpoint supports various query parameters for filtering and searching, such as:
  + salesorder\_id: To list packages associated with a specific Sales Order.
  + customer\_id: To list packages for a particular customer/recipient.
  + filter\_by: Filter by status (e.g., NotShipped, Shipped, Delivered).
  + date\_start, date\_end: Filter by package creation date range.
  + search\_text: For general text-based search.

The reportFunction can leverage these endpoints, particularly the list endpoint with appropriate filters, to gather data on donation batches for reporting purposes.

**8. API Rate Limits**

**8.1. Overview of Rate Limits**

Zoho Inventory, like most cloud-based API services, imposes rate limits on the number of API requests an application can make within a given period. These limits are typically defined on a per-day basis and vary depending on the Zoho Inventory subscription plan of the organization.

The documented daily API request limits per plan are :

* Free Plan: 1500 API requests/day
* Standard Plan: 2500 API requests/day
* Professional Plan: 5000 API requests/day
* Premium Plan: 7500 API requests/day
* Enterprise Plan: 10000 API requests/day

It is important to note that these limits usually apply to the organization as a whole, meaning all API integrations and applications accessing that organization's data share this common pool of requests.

**8.2. Implications for Developers**

Developers integrating with the Zoho Inventory API must be acutely aware of these rate limits and design their applications accordingly to prevent service disruptions caused by exceeding these thresholds. Key considerations include:

* **Efficient API Usage:** Applications should be designed to use the API efficiently. This includes:
  + **Avoiding Unnecessary Calls:** Cache data that does not change frequently rather than re-fetching it repeatedly. For example, the getZohoAccessToken function should cache the active access token and its expiry time to avoid making redundant calls to the OAuth token endpoint.
  + **Using Bulk Operations:** Where available, utilize API endpoints that support bulk operations. For instance, fetching details for multiple items using GET /itemdetails with a list of IDs is more efficient than making individual GET /items/{item\_id} requests in a loop. Similarly, Bulk delete Sales Orders is preferable to deleting one by one if applicable.
* **Batch Processing:** For tasks involving large numbers of records (e.g., initial data synchronization or large updates), design the process to run in batches, respecting the API limits. Spread out requests over time if necessary.
* **Error Handling and Retry Mechanisms:** Implement robust error handling that can identify rate limit errors (often indicated by an HTTP 429 "Too Many Requests" status code). When such errors occur, the application should employ a retry mechanism with exponential backoff (i.e., wait for progressively longer intervals before retrying). However, be cautious that aggressive retries during a rate-limited state do not exacerbate the problem.
* **Monitoring:** Monitor API usage if Zoho provides tools or dashboards for this. Alternatively, implement logging within the application to track the number of API calls being made, which can help anticipate potential rate limit issues.

The getZohoAccessToken function, while essential, also contributes to API call consumption, as each token refresh is an API call. If this function is called excessively (e.g., by many other functions or services, or if it refreshes tokens too aggressively without proper caching), it can significantly impact the daily rate limit. Therefore, optimizing the logic within getZohoAccessToken to minimize token requests is crucial for the overall health and sustainability of the integration, especially in environments with high transaction volumes or multiple integrated processes.

**9. Conclusion**

Integrating with the Zoho Inventory API v1 offers powerful capabilities for automating and extending inventory and order management processes. This guide has detailed the essential components for such an integration, focusing on OAuth 2.0 authentication, organization handling, and the management of items (crates), item adjustments (stock control), sales orders (dispatch orders), and packages (donation batches).

Key considerations for developers embarking on this integration include:

* **Authentication:** A thorough understanding and correct implementation of the OAuth 2.0 flow, particularly the refresh token mechanism for server-side applications like Lambda functions (getZohoAccessToken), is paramount for secure and continuous API access. Secure storage of Client ID, Client Secret, and refresh\_token is critical.
* **Organization ID and Data Centers:** All API requests must correctly specify the organization\_id and target the appropriate data center URI for the organization. These are fundamental for routing requests correctly.
* **Resource Mapping:** User-specific concepts like "crates," "dispatch orders," and "donation batches" can be effectively mapped to Zoho Inventory's native resources: Items, Sales Orders, and Packages, respectively. However, the mapping for "donation batches" to Packages requires an intermediate step of creating a Sales Order to represent the donation, due to the Packages API's dependency on Sales Orders.
* **API Semantics:** Developers must pay close attention to the specific semantics of API parameters, such as quantity\_adjusted in Item Adjustments (which typically represents a delta) and the importance of location\_id in Sales Orders and Item Adjustments for accurate multi-location inventory tracking.
* **Rate Limits:** Adherence to API rate limits is crucial for uninterrupted service. Efficient API usage, caching, bulk operations, and careful design of token management strategies are necessary to operate within these limits.

By addressing these aspects systematically, developers can build robust, secure, and efficient integrations with Zoho Inventory API v1, leveraging its capabilities to meet diverse business requirements. The provided tables for key parameters in item creation, item adjustments, sales order creation, and package creation serve as practical references for constructing API request payloads accurately. Careful planning and adherence to the API's design principles will lead to successful and scalable solutions.