Oracle® Warehouse Management
Cloud

Rest API Guide
Update 20B
Part No. F29972-01
March 2020
<table>
<thead>
<tr>
<th>Response Statuses</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a Resource (POST)</td>
<td>33</td>
</tr>
<tr>
<td>Input Data</td>
<td>33</td>
</tr>
<tr>
<td>Data Structure</td>
<td>33</td>
</tr>
<tr>
<td>Related Resources</td>
<td>34</td>
</tr>
<tr>
<td>Response Statuses</td>
<td>34</td>
</tr>
<tr>
<td>Validations</td>
<td>34</td>
</tr>
<tr>
<td>Nested Related Objects</td>
<td>34</td>
</tr>
<tr>
<td>Supported Entities</td>
<td>35</td>
</tr>
<tr>
<td>Updating a Resource (PATCH)</td>
<td>35</td>
</tr>
<tr>
<td>Input Data</td>
<td>36</td>
</tr>
<tr>
<td>Response Statuses</td>
<td>36</td>
</tr>
<tr>
<td>Entities and Fields for PATCH</td>
<td>37</td>
</tr>
<tr>
<td>IB_shipment</td>
<td>37</td>
</tr>
<tr>
<td>IB_shipment_DTL</td>
<td>37</td>
</tr>
<tr>
<td>Item_characteristics</td>
<td>38</td>
</tr>
<tr>
<td>Load</td>
<td>38</td>
</tr>
<tr>
<td>Location</td>
<td>39</td>
</tr>
<tr>
<td>Order_hdr</td>
<td>39</td>
</tr>
<tr>
<td>Order_dtl</td>
<td>40</td>
</tr>
<tr>
<td>Purchase_order_hdr</td>
<td>41</td>
</tr>
<tr>
<td>Purchase_order_dtl</td>
<td>41</td>
</tr>
<tr>
<td>Work_order_hdr</td>
<td>41</td>
</tr>
<tr>
<td>Work_order_kit</td>
<td>42</td>
</tr>
<tr>
<td>Entity Operations (GET/POST)</td>
<td>42</td>
</tr>
<tr>
<td>Response Status</td>
<td>42</td>
</tr>
<tr>
<td>Bulk Operations</td>
<td>42</td>
</tr>
<tr>
<td>4. Supported Entity Operations</td>
<td>45</td>
</tr>
<tr>
<td>Inventory</td>
<td>46</td>
</tr>
<tr>
<td>Location</td>
<td>47</td>
</tr>
<tr>
<td>Update_active_inventory</td>
<td>47</td>
</tr>
<tr>
<td>Item</td>
<td>49</td>
</tr>
<tr>
<td>Image Upload</td>
<td>49</td>
</tr>
<tr>
<td>Trailer</td>
<td>50</td>
</tr>
<tr>
<td>First available</td>
<td>50</td>
</tr>
<tr>
<td>Locate_to_yard</td>
<td>51</td>
</tr>
<tr>
<td>Remove_from_yard</td>
<td>52</td>
</tr>
<tr>
<td>Load</td>
<td>53</td>
</tr>
<tr>
<td>Check_in</td>
<td>53</td>
</tr>
<tr>
<td>Check_out</td>
<td>55</td>
</tr>
<tr>
<td>Ship load</td>
<td>56</td>
</tr>
<tr>
<td>Container</td>
<td>58</td>
</tr>
<tr>
<td>Get sales orders</td>
<td>58</td>
</tr>
<tr>
<td>Lock container</td>
<td>58</td>
</tr>
</tbody>
</table>

Copyright © 2019, 2020, Oracle and/or its affiliates.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BULK LOCK CONTAINER</td>
<td>58</td>
</tr>
<tr>
<td>UNLOCK CONTAINER</td>
<td>59</td>
</tr>
<tr>
<td>BULK UNLOCK CONTAINER</td>
<td>59</td>
</tr>
<tr>
<td>PALLETIZE CONTAINER</td>
<td>59</td>
</tr>
<tr>
<td>DEPALLETIZE INBOUND/OUTBOUND LPN</td>
<td>61</td>
</tr>
<tr>
<td>PICK-PACK</td>
<td>62</td>
</tr>
<tr>
<td>PICK CONFIRM</td>
<td>62</td>
</tr>
<tr>
<td>CLOSE LPN</td>
<td>65</td>
</tr>
<tr>
<td>WAVE COMPLETE</td>
<td>66</td>
</tr>
<tr>
<td>TASK</td>
<td>67</td>
</tr>
<tr>
<td>NEXT TASK</td>
<td>67</td>
</tr>
<tr>
<td>IBLPN</td>
<td>69</td>
</tr>
<tr>
<td>DIRECT_CONSUME</td>
<td>69</td>
</tr>
<tr>
<td>MODIFY_ITEM_QTY</td>
<td>73</td>
</tr>
<tr>
<td>OBLPN</td>
<td>75</td>
</tr>
<tr>
<td>CREATE_FROM_IBLPN</td>
<td>75</td>
</tr>
<tr>
<td>LINK OBLPN WITH ASSET</td>
<td>80</td>
</tr>
<tr>
<td>PALLET</td>
<td>82</td>
</tr>
<tr>
<td>SORT LPN</td>
<td>82</td>
</tr>
<tr>
<td>SORT LPN/CLOSE PALLET</td>
<td>84</td>
</tr>
<tr>
<td>REPLENISHMENT</td>
<td>86</td>
</tr>
<tr>
<td>REPLENISH TO ACTIVE</td>
<td>86</td>
</tr>
<tr>
<td>SALES ORDER HEADER</td>
<td>87</td>
</tr>
<tr>
<td>PRINT</td>
<td>88</td>
</tr>
<tr>
<td>PRINT SHIPPING LABEL</td>
<td>88</td>
</tr>
<tr>
<td>EXAMPLE QUERY STRING FOR GET</td>
<td>89</td>
</tr>
<tr>
<td>RESPONSE BODY DATA</td>
<td>90</td>
</tr>
<tr>
<td>PRINT LPN LABEL</td>
<td>90</td>
</tr>
<tr>
<td>API FILTERS</td>
<td>91</td>
</tr>
<tr>
<td>RESPONSE BODY DATA</td>
<td>92</td>
</tr>
<tr>
<td>API FILTERS</td>
<td>93</td>
</tr>
<tr>
<td>RESPONSE BODY DATA</td>
<td>93</td>
</tr>
<tr>
<td>REPORT</td>
<td>93</td>
</tr>
<tr>
<td>CUSTOM INVENTORY SUMMARY</td>
<td>94</td>
</tr>
<tr>
<td>COMPANY PARAMETER</td>
<td>94</td>
</tr>
<tr>
<td>FACILITY PARAMETER</td>
<td>95</td>
</tr>
<tr>
<td>LOCATION SIZE TYPE</td>
<td>95</td>
</tr>
<tr>
<td>PUTAWAY PRIORITY</td>
<td>96</td>
</tr>
</tbody>
</table>
Send Us Your Comments

Oracle® Warehouse Management Cloud Rest API Guide, Update 20B

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: owms-cloud-comms_us@oracle.com

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, contact Support at https://support.oracle.com or find the Support phone number for your region at http://www.oracle.com/support/contact.html.
Preface

This document provides an overview of REST APIs which can also be used as a tool by individual users for working with the application, but is primarily designed for system integration.

Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Document Revision</th>
<th>Summary of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/2018</td>
<td>-01</td>
<td>Initial release.</td>
</tr>
<tr>
<td>09/2018</td>
<td>-02</td>
<td>Updates for 18C.</td>
</tr>
<tr>
<td>03/2019</td>
<td>-03</td>
<td>Added details on the following Supported Entity Operations: IBLPN: direct consume, modify_item_qty, Trailer: locate_to_yard, remove_from_yard Load: check_in, check_out OBLPN: create_from_iblpn Location: update_active_inventory</td>
</tr>
<tr>
<td>06/2019</td>
<td>-04</td>
<td>Added the following in section 4: Supported Entity Operations: Palletize Container, Depalletize Container, Link OBLPN with Asset Print Shipping, LPN, Pallet Label, Custom Inventory Summary Location Size Type, Putaway Type, Putaway Type Calculation Rule Replenishment Zone, SQL Selection (Rule Tree)</td>
</tr>
<tr>
<td>08/2019</td>
<td>-05</td>
<td>Updated Print APIs in section 4: Supported Entity Operations:</td>
</tr>
<tr>
<td>08/2019</td>
<td>-06</td>
<td>In section 4: Supported Entity Operations, changed Depalletize Container to Depalletize Inbound/Outbound LPN and updated information details.</td>
</tr>
<tr>
<td>02/2020</td>
<td>-08</td>
<td>Added Putaway Priority to section 4: Supported Entity Operations</td>
</tr>
<tr>
<td>03/2020</td>
<td>-09</td>
<td>Added stop_ship_flag to Order Hdr table in Updating a Resource (PATCH) section.</td>
</tr>
</tbody>
</table>
1. Overview

Oracle Warehouse Management (WMS) Cloud update 18C introduces a new set of RESTful API resources designed for interacting with and manipulating application data using HTTP requests. The purpose of these APIs is to expose application data to external systems in a controlled and predictable manner. REST APIs can be used as a tool for individual users working with the application, but they are primarily designed for system integration.

This document includes an overview of the newer style APIs introduced in version 9.0.0. The WMS legacy APIs available in earlier WMS versions will still continue to be available. For more details on legacy APIs, refer to the Oracle Warehouse Management Cloud Integration API Guide.

Audience

This audience is intended for REST API software developers with customers or system implementors. While the document includes a reasonable overview of REST concepts, the assumption is that the audience understands REST, HTTP communication, response codes, and related topics.

RESTful Web Services

Representational State Transfer (REST) is a web standards-based architecture utilizing the HTTP protocol for data communication. RESTful web services are a light weight, scalable, and maintainable way to allow web-based system-to-system communication, irrespective of the respective application platforms (interoperability).

RESTful web services use HTTP methods in combination with a Universal Resource Identifier (URI) to implement the REST architecture. For reference, a URL is a type of URI. This combination allows consumers to interact with application data via a set of controlled, stateless, and idempotent methods.

OCWMS has had REST API’s prior to update 18C, however they were not designed to provide fine grained access. These legacy API’s continue to be available. Once all the functionality provided by these API’s are incorporated into the newer APIs, the legacy ones will be retired with sufficient notice. The new APIs also adhere to RESTful practices better and simplify some of the data encoding requirements.

HTTP Requests

RESTful web services are built on top of the HTTP protocol, which carries some important implications. First, each request is stateless. This means that each request is independent of any other requests and the request itself must contain all relevant data to fulfill the request. Second, certain types of requests should be idempotent; making identical requests should yield the same result on the server. This is a safety measure that also provides consistency. For example, when reading data the same request should always yield the same result assuming the resource’s state on the server has not changed between requests.

HTTP Methods

The APIs may utilize the following five HTTP methods in order to provide users with Create-Read-Update-Delete (CRUD) functionality. Note that not all APIs support all methods.

GET

Return a read-only representation of the selected resource(s) in the response body.
**HEAD**
Read-only check for resource existence and/or modification. Does not return a response body.

**POST**
Create resources or submit data to be processed by a resource operation.

**PATCH**
Modify existing resource(s).

**DELETE**
Remove/deactivate existing resource.

**URI Format**
The lgfapi URI structure is broken down into several components.

In general, lgfapi URIs following the following schema:

```
https://xxxxx.logfireapps.com/myenv/wms/lgfapi/v9/entity/company/1/
```

The first portion of the URI (protocol, domain, environment, and app) is consistent with the URL of the environment’s UI accessed via a web browser. The remaining pieces after “lgfapi” are specific to the lgfapi and designate the version and path to any child modules and/or resources.

**Versioning**
Lgfapi requires a version number in all URIs. The format is “v#”, starting with “v9” as the first release. New versions are created only for major releases of the OCMWS application, not for minor versions. For example, the release of OCWMS 9.0.0 included the lgfapi v9 release, but there will not be a new lgfapi version number with the release of OCWMS 9.0.1. However, the APIs will continue to be updated with new features and improvements along with the minor releases of OCWMS.

The purpose of version control is to give customers some ability to remain on their current integrations until they can complete any changes required to handle the newest lgfapi version. It is strongly encouraged that all customers use the latest version of lgfapi. Version control is a tool to assist with upgrades and testing, it is not meant to be used in production for extended periods of time. The previous versions of lgfapi will unavoidably become out of sync with newer versions of OCWMS, and eventually will no longer be compatible. Oracle will not make changes to previous versions of lgfapi in order to maintain expired functionality or compatibility. Therefore, it is always in the best interest to use the latest version. New API versions are planned approximately once a year. Older API versions will be supported approximately one year after a newer one is released.

**Lgfapi Modules**
Lgfapi contains modules that can be utilized by customers. These are groupings of functionality that may have their own formats and requirements. For example, lgfapi’s “entity” module is designed to allow customers to examine and interact with OCWMS business resources from outside the application.
Resource Path

The final component to the URI is the resource path. This may take many different forms depending on the HTTP method and any module-specific requirements.

Optional Trailing Slashes

A trailing slash at the end of and Lgfapi URIs is optional and does not affect functionality.

Login and Authentication

Since each HTTP request is stateless, every request requires information to authenticate the user. Lgfapi supports several types of user authentication:

- BasicAuth – Classic username and password.
- OAuth2 – A token based authorization framework.

Application Permissions

Making a request to Lgfapi not only requires user authorization, but also one or more of the CRUD application-level permission to access the supported HTTP methods. These are configurable in the user’s group-level permissions.

- “lgfapi_read_access” – GET, HEAD
- “lgfapi_create_access” – POST
  - Note – this access is also required in order to run resource operations.
- “lgfapi_update_access” – PATCH
- “lgfapi_delete_access” – DELETE

It’s recommended to create dedicated user(s) with appropriate Lgfapi permissions and different facility/company eligibility to protect the integrity of your data. For instance, it is safe to give users read access but may not be appropriate to grant them permission to create or modify data.

The legacy API permission, “can_run_ws_stage_interface”, has been replaced by the new permission, “lgfapi_update_access”. This permission now applies to both Lgfapi and the legacy APIs. For legacy API’s, this is the singular permission required to access all APIs. For Lgfapi, this is one of several new permissions used to control user access. Any existing users with the legacy permission are automatically granted the appropriate new API permission(s) upon upgrading to OCWMS 9.0.0.

Data Input Methodology

Lgfapi allows for transmission of data in one of two ways, based on the HTTP method being used.

GET/HEAD

These read-only HTTP methods allow the user to pass additional information about the request in the URI. This data is sent as key-value pairs and starts with a question mark (“?”) at the end of the main URI. This section of the URI is known as the “query string”. Each key-value pair is known as a “parameter”. It is used to provide additional information to the resource. Parameters are delimited by an equals sign (“=”), and multiple parameters are delimited by an ampersand (“&”). The order of the parameters does not matter.
URL Encoding

In general, URIs only allow ASCII values, however there are specific cases like with internationalized domain names (IDN) where non-ASCII characters may be used in the domain name. For the purposes of communicating data using query string parameters in lgfapi, you cannot directly send non-ASCII (unsafe) characters. Also, some characters like spaces, “=”, and “&” have a specific meaning when sent in the query string section of the URI and are reserved. In order to handle unsafe characters and to distinguish between data and reserved characters that have special meaning in a URI, the URI must be “URL Encoded”. This encoding replaces non-ASCII and reserved characters parameter data with ASCII equivalents. This is also known as “Percent Encoding” since each unsafe character is replaced with a value starting with percent sign (“%”). All parameter values should be URL encoded to ensure correct transmission.

For example, the query string: "foo=Mañana" is URL encoded as "foo= %20Ma%C3%B1ana". A URI cannot have a space so that is encoded to the value "%20". The Spanish letter “ñ” is not a valid ASCII value and is encoded as “%C3%B1". Once the data reaches the server, it is decoded back to the original characters. The key portion of each parameter is determined by the application and therefore will never contain unsafe characters.

See [https://www.w3schools.com/tags/ref_urlencode.asp](https://www.w3schools.com/tags/ref_urlencode.asp) for more information.

It is possible to repeat the same parameter within the query string. However, lgfapi will only observe the final occurrence of the parameter in order to obtain a value. For example, given the query string “?code=A&code=B”, the interpreted value of the “code” parameter will be “B”. The “A” value is discarded. There is no use case for transmitting repeated parameters as the desired result is achieved through other module-specific query string mechanisms.

POST

A POST request is used to pass data to the server similar to pressing a “Submit” button on a web page to submit form data to the server. In the context of lgfapi, when making a POST request, the user is passing data to either create a resource or invoke a resource operation, such as cancelling an order. Unlike GET and HEAD requests, POST allows for text data to be passed in the free-form body of the request. Request body data must be in a supported format (JSON or XML) and follow the required structure of the API being invoked.

**Content-Type HTTP Header**

This HTTP header is required when using a method like POST, PATCH, and DELETE that allow transmitting data in the body of the request. It describes the data format so it can be correctly parsed server-side. Lgmapi supports JSON and XML input and therefore requires one of the two content-type values:

- application/json
- application/xml

*The Content-Type "application/x-www-form-urlencoded" is not supported in lgfapi, but is still required for legacy OCWMS APIs.*
Content Encoding

By default, lgfapi will use UTF-8 to decode the request body as this handles the majority of characters for languages supported in OCWMS. However, for situations where customers choose to use a different encoding, it can be specified in the Content-Type header's optional "charset" parameter:

Content-Type: application/json; charset=latin-1

Lgfapi will use the provided charset to decode the request body data. It is up to the customer to ensure that their data is properly encoded using the desired charset before transmission to lgfapi. Failure to do so may result in incorrect characters or an inability to process the request.

It is also important to note that this only applies to the encoding of the request body and does not apply to the encoding used in any response body data from lgfapi.

Request Body Data – Repeated Keys

Lgfapi does not restrict users from repeating data in the request body for a single request. Rather, it will use only the final occurrence in the body when processing the request.

For example, if one were to send a request with the key “code” multiple times in the same request body:

```json
{  
  "code": "A",  
  "code": "B"
}
```

The value used to process the request will be "B". "A" is ignored and is never used. There is no lgfapi use case for need to pass repeating data in the same request.

Request Body Data List Formatting

JSON and XML data follow language standards except for the case of lists of items in XML. This is a unique concern for XML since there is no standard methodology for how to handle lists whereas JSON supports lists by default.

XML Lists

A list of items in XML is represented by the wrapper tag, followed by a wrapper for each item’s value with the special tag name "list-item". For example, representing a list of serial numbers under the wrapper "serial_nbr_list", in JSON is represent as:

```json
{
  "serial_nbr_list": [
    "SN1",
    "SN2"
  ]
}
```

The equivalent XML list would be represented as the following. Note the use of “list-item” for each entry in the list to allow for correct parsing.

```xml
<serial_nbr_list>
  <list-item>SN1</list-item>
</serial_nbr_list>
```
<list-item>SN2</list-item>
</serial_nbr_list>
2. HTTP Response

Every valid HTTP request receives a response that is comprised of three main components:

(1) A 3-digit response status code that gives information about the success or failure of the request, the returned content, and other information specific to the request.

(2) The response header(s), which vary by request. These headers contain metadata information about the request, the response, the response data, and/or attributes of the server.

(3) The response body where free-form text information can be returned to the requester in either JSON (default) or XML format and in a standard defined by the application. This is where application-specific data pertaining to representation, success, and errors is returned to the requester.

Status Codes

Comprehensive list of HTTP status codes: [https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)

Lgfapi uses many of the available HTTP response status codes to convey success or failure of the request back to the user. All response status codes fall into 1 of 4 categories:

- **1xx** – Informational
- **2xx** – Success
- **3xx** – Redirection
- **4xx** – Failure

The following is a list of commonly used response status codes for Lgfapi:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Status Message</th>
<th>HTTP Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Ok</td>
<td>HEAD, GET, POST</td>
<td>GET - The request was successful. HEAD - The resource exists. POST - Resource exists and/or has been modified.</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>POST</td>
<td>Resource successfully created.</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
<td>POST</td>
<td>The request was successful, but no content is being returned in the response body.</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
<td>HEAD</td>
<td>The resource has not been updated since the target date-time.</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>HEAD, GET, POST</td>
<td>Invalid data or request structure.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>HEAD, GET, POST</td>
<td>Invalid login credentials.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>HEAD, GET, POST</td>
<td>User lacks permission.</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>HEAD, GET, POST</td>
<td>The resource does not exist.</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
<td>-</td>
<td>HTTP method is not supported for the requested resource.</td>
</tr>
<tr>
<td>409</td>
<td>Conflict</td>
<td>HEAD, GET, POST</td>
<td>Record Changed - The resource was modified by a concurrent operation before the request could be fulfilled. Try again.</td>
</tr>
<tr>
<td>500</td>
<td>Server Error</td>
<td>HEAD, GET, POST</td>
<td>An unhandled error occurred or the application was unable to formulate a valid response. Please contact support</td>
</tr>
</tbody>
</table>
Response Formats

Lgfapi supports JSON (default) and XML formats for data returned in the body of the response. This applies to all HTTP methods that return a response body.

The requester is able to specify the response format in several ways:

1. Making a request without specifying the response format will result in the default JSON format.
2. Using the reserved "format" query string parameter in the URI when making a request.
   
   You can set the format to XML by adding "format=xml" to the query string portion of the request (the key-value pair data after the "?"). This is in addition to any other query string parameters also in the URI:

   .../resource/?format=json
   .../resource/?format=xml

   Note – "format" is one of the few query string parameters you can use with HTTP methods like POST, which typically require all data to be in the body of the request.

3. Using the file-extension dot-notation in the URI when making a request.

   Very similar to the example above, you can also request the format using dot notation like you would when giving a file the extension "*.xml" or "*.json":

   .../resource/*.json
   .../resource.xml (optional trailing slash)

   This can also be combined with a query string:
   .../resource/*.xml?key1=value1&key2=value2

Response Data Encoding

When a response body is returned, the raw JSON or XML data will always be encoded using UTF-8. There is no way to configure or specify the response body’s encoding. This is done to ensure that the response content can always be correctly rendered. A request body using a different encoding is allowed because the requester is able to control the contents being sent to Lgfapi. However, the output data may contain characters outside of the encoding used for the request, if for example a consistent character set has not been used throughout the application. UTF-8 covers the full change of characters supported by OCWMS and is therefore the default, and generally preferred, encoding.

Response Data Formats

In general, the HTTP response body can take on any number of different formats and styles. For Lgfapi, several dedicated conventions have been adopted to give uniformity and consistency to the handling of both successful and erroneous requests.
**Error Response**

A standardized error format is returned in the body of the response whenever there is an error while fulfilling the request. This is accompanied by the response status code, which provides additional insight.

The standard error response is comprised of 4 components:

- **Reference** – A unique string used as reference for the request and error. This should be provided in support requests to help more quickly identify the information pertaining to the request in question.
- **Code** – A generic classification pertaining to the error message.
- **Message** – An error message related to the code.
- **Details** – Optional. Either a list or key-value map (dictionary) of more detailed information pertaining to the error(s). For example, this may give a more detailed list of error messages or could be a map of field name(s) to error(s).

Example JSON Error Response Body:
```json
{
  "reference": "25b414f0-7a1d-4f35-ac3c-0ec9886cf37a",
  "code": "VALIDATION_ERROR",
  "message": "Invalid input.",
  "details": {
    "reason_code": "Invalid Reason code"
  }
}
```

Example XML Error Response Body:
```xml
<?xml version="1.0" encoding="utf-8"?>
<error>
  <reference>25b414f0-7a1d-4f35-ac3c-0ec9886cf37a</reference>
  <code>VALIDATION_ERROR</code>
  <message>Invalid input.</message>
  <details>
    <reason_code>Invalid Reason code</reason_code>
  </details>
</error>
```

**Unhandled Errors**

It is possible that the application is unable to convey the nature of the problem back to the requester. In these scenarios, the server will respond with a 500 ("Server Error") status code and an accompanying message.

**Resource Representations**

Representations are by default paginated unless a specific resource is being requested. Pagination allows the response data to be served in chunks (pages) to keep payload sizes manageable.

**Pagination**

A paginated result set is returned when multiple representations may exist in the result set that exceed a preset size. This breaks the result set into chunks (pages), each with its own page number. The page size is determined by the requesting user’s configuration of the field "Rows per Page". This is the same field used to set the number of results per UI page returned. It has an allowed range of 10 to 125 results per page.
Pagination Mode

Two modes of pagination are supported that offer different advantages and disadvantages depending on the user requirements. The default mode is “paged”, but users may specify the type of pagination by using the “page_mode” query string parameter in the URI. The two types are “paged” and “sequenced”.

Mode: Paged

This is the default mode for result sets (…/resource/?page_mode=paged). This will break the data into chunks (pages) and return one page per request. This will additionally return metadata such as the total count of results and the total number of pages.

Each page of the result set is given a pagination header:
- result_count – The total number of results across all pages.
- page_count – The total number of pages.
- page_nbr – The current page number.
- next_page – Hyperlink to the next page (if available).
- previous_page – Hyperlink to the previous page (if available).
- results – The result set list for the page.

A specific page number for a paginated result set is requested in the URI’s query string using the parameter “page”. For example, to request the data for page 3 of a result set, one would add …/resource/?page=3. You will also see these automatically added in the hyperlinks generated for “next_page” and “previous_page”.

An example of a paginated JSON response:

```json
{
  "result_count": 1,
  "page_count": 1,
  "page_nbr": 1,
  "next_page": null,
  "previous_page": null,
  "results": [
    {
      "id": 0,
      ...
    }
  ]
}
```

An example of a paginated XML response:

```xml
<?xml version="1.0" encoding="utf-8"?>
<entity_name>
  <result_count>1</result_count>
  <page_count>1</page_count>
  <page_nbr>1</page_nbr>
  <next_page></next_page>
  <previous_page></previous_page>
  <results>
    <list-item>
      <id>0</id>
    </list-item>
  </results>
</entity_name>
```
Mode: Sequenced
The sequenced mode \( \text{/resource/?page_mode=sequenced} \) is similar to the Paged mode, except for a few important details. This mode is recommended for system to system integration where superfluous information and intuitive/human-readable values are not necessary.

Each page of the result set is given a header that conveys extra information to the user and makes it easier to navigate between pages:
- next_page – Hyperlink to the next page (if available).
- previous_page – Hyperlink to the previous page (if available).
- results – The result set list for the page.

First, you’ll notice that the pagination header does not have the total result count or total page count. This is because sequenced pagination doesn't know either of these values, and doesn’t want to. Instead, each page is generated on the fly in an effort to improve performance, which means less work than paged mode where the total counts are fetched up front. Determining total count can be expensive when you have a large result set.

With sequenced, you also sacrifice some human readability and functionality as the “page” query string parameter is replaced by a system-generated “cursor” as well as the hyperlinks will not be as intuitive to understand. Since in this mode the total result set is not known, only what’s rendered per page, there is no way to report the total number of pages or label each with a specific page number. A cursor identifier is generated for each page instead of a page number:

\( \text{.../resource/?cursor=cD0xNDAw&page_mode=sequenced} \)

Non-paginated Responses
There are a few scenarios where a request will return data in the body of the response for a specific object, so pagination is not needed.

The first is for a GET retrieve style request where the “id” value of the resource is known and is requested in the URI \( \text{.../resource/{id}} \).

The second is when creating a single resource using a POST request. The response will be a non-paginated representation for only the new resource.
3. Entity Module

The lgfapi entity module is used to access and modify OCWMS application data. It exposes specific methodologies for identifying subsets of data and obtaining their representations as well as allowing for the creation of certain resources. The entities supported and corresponding functionality will continue to be expanded through subsequent releases.

Supported Entities

The entity module has a documenting feature that can be accessed via a GET request to the top-level (root) URL (.../lgfapi/v10/entity/). This will return a sorted list of supported entities for the given lgfapi version and an accompanying base URL.

Each entity represents an object or combination of objects within OCWMS that is accessible via lgfapi. However, not all entities support all HTTP methods. Furthermore, these entities may share characteristics with their respective counterparts in other areas of the OCWMS application, but as a whole should be considered independent of other application functionality.

Entity Metadata

It is possible to obtain additional information for each entity by making a GET request to the “describe” entity operation (.../lgfapi/v10/entity/{entity_name}/describe/). This will return metadata that can be used to further your understanding of the entity. See “Entity Operations” section for more details.

Input Data Types

Lgfapi supports user input depending on the HTTP method:

- **GET/HEAD**
  - Query string parameters
- **POST**
  - Request body data
    - The format must be JSON or XML
  - The “format” query string parameter alone is supported to specify the desired format for the response.

Although the input formats may be type ambiguous, the input value is cast to the appropriate type as defined in the entity’s field metadata. Some fields have naming conventions that are outlined below. The following types are supported for user input:

**String/Text**

Query String: .../?field=abc123
JSON: {"field": "abc123"}
XML: <field>abc123</field>

**Integer**

Query String: .../?field=123
JSON: {"field": 123}
XML: <field>123</field>

**Numeric/Decimal**

Query String: .../?field=1.234
JSON: {"field": 1.234}
XML: <field>1.234</field>

**Boolean**

Except for a few specific cases, all True/False Boolean field names end with “_flg”.

---

Copyright © 2019, 2020, Oracle and/or its affiliates.
The input value for all formats should be either "true" or "false".

Query String: .../?field_flg=true
JSON: {"field_flg": true}
XML: <field_flg>true</field_flg>

**Temporal (Date/Time)**

All date, time, and date-time fields require the iso-8601 format: YYYY-mm-ddTHH:MM:SS.ffffff
Note that the microsecond component "f" is optional. Using January 30th, 2018 at 6:30pm as an example:

**Date**
Field names for date-only fields typically end with "_date".

Query String: .../?field_date=2018-01-30
JSON: {"field_date": "2018-01-30"}
XML: <field_date>2018-01-30</field_date>

**Time**
Field names for time-only fields typically end with "_time".

Query String: .../?field_time=18:30:00
JSON: {"field_time": "18:30:00"}
XML: <field_time>18:30:00</field_time>

**Date-time**
Field names for date-time fields typically end with "_ts".
All Date-time objects are assumed to be in the time zone of the user's facility context.
In other words, it should be the date/time you would expect to see if viewed by the user in the UI.

Query String: .../?field_ts=2018-01-30T18:30:00
JSON: {"field_ts": "2018-01-30T18:30:00"}
XML: <field_ts>2018-01-30T18:30:00</field_ts>

**Relational**
Relational fields are when one resource has a link to another resource. These fields always end in "_id" and by default, are integer values. They are unique when filtering, in that you can use the double-underscore ("__") notation to reference a related resource's fields, or even nested related resources. This is covered in more detail in the Resource Result Set Filtering section.

Query String: .../?field_id=1
JSON: {"field_id": 1}
XML: <field_id>1</field_id>
Resource Result Set Filtering

Lgfapi offers the ability to apply filters to GET and HEAD requests in order to narrow down the final result set. This is done by adding query string filter parameters to the URI. Furthermore, Lgfapi supports several built-in lookup functions to assist in common filtering tasks.

It is important to note that all entity data is automatically filtered by the user’s eligible facilities and companies. This prevents users from being able to access and/or change data outside of their assigned scope that same way that data is isolated in the UI or RF features. The difference with Lgfapi is that users may access data from multiple eligible facilities and companies in a single request. In the UI and RF, this typically requires manually changing the user’s context.

The most basic format for a filter uses simply the exact operator ("="): .../?field=value
This can be chained to apply multiple filters: .../?field1=value1&field2=value2

Lgfapi uses double underscore ("__") notation in order to join multiple fields or functions in the query string filters. The double underscore is used to distinguish the field names when filtering on a related resource’s attributes or when applying a lookup function.

Applying a lookup function: .../?field__lookup=value
Filtering on a related resource: .../?relation_id__related_field=value
Applying a lookup function on a related resource: .../?relation_id__related_field__lookup=value

These are discussed in detail in the following sections.

Supported Lookup Functions

The following lookup functions are provided by Lgfapi. Note that any match function with a corresponding "i" function means that function is case-insensitive. For example, "exact" is used to match exactly on a value, as does "iexact" except that the latter ignores upper/lower case.

**Arithmetic Lookups**

- **gt** – Greater than

  Example: Filtering sales order detail(s) for only those with an ordered quantity.
  .../order_dtl/?ord_qty__gt=0

- **gte** – Greater than or equal to

  Example: Filtering sales order detail(s) for only those with an ordered quantity.
  .../order_dtl/?ord_qty__gte=1

- **lt** – Less than

  Example: Filtering sales order detail(s) for only those with ordered quantity below 10.
  .../order_dtl/?ord_qty__lt=10

- **lte** – Less than or equal to

  Example: Filtering sales order detail(s) for those with ordered quantity at or below 10.
  .../order_dtl/?ord_qty__lte=10

**Text Match Lookups**

- **contains/icontains** – Text contains substring

  Example: Filtering sales order(s) for orders with “FOO” in the order_nbr field.
  .../order_hdr/?order_nbr__contains=FOO
Example: Same as previous example, but ignore case.
.../order_hdr/?order_nbr__icontains=FOO

- `exact/iexact` – Text exactly matches

Example: Match sales order(s) exactly on the order number.
.../order_hdr/?order_nbr__exact=ORDER001

Note: “Exact” is not typically needed. The above filter condition does not require the exact lookup since this is automatically implied by the exact operator (“=”).

The query string can be simplified to:
.../order_hdr/?order_nbr=ORDER001

“iexact”, on the other hand, is a useful tool when you need to do an exact match, but ignore letter casing:
.../order_hdr/?order_nbr__iexact=OrDeR001

- `startswith/istartswith` – Text starts with

Example: Filtering sales order(s) for only those whose order_nbr starts with “ORD”:
.../order_hdr/?order_nbr__startswith=ORD

- `endswith/iendswith` – Text ends with

Example: Filtering sales order(s) for only those whose order_nbr ends with “001”:
.../order_hdr/?order_nbr__endswith=001

**Temporal (Date/Time) Lookups**

The following temporal functions may only be used on date, time, and/or date-time data. Consider the “order_hdr” entity’s “order_shipped_ts” date-time field with a value “2018-09-17T20:30:59”:

- `year` – Match on a date’s year (date or date-time).
  ...
  .../order_hdr/?order_shipped_ts__year=2018

- `month` – Match on a date’s month (date or date-time).
  ...
  .../order_hdr/?order_shipped_ts__month=09

- `week_day` – Match on a date’s day of the week (date or date-time).
  Takes an integer value representing the day of week from 1 (Sunday) to 7 (Saturday).
  ...
  .../order_hdr/?order_shipped_ts__week_day=2

- `day` – Match on a date’s day (date or date-time).
  ...
  .../order_hdr/?order_shipped_ts__day=17

- `hour` – Match on a date’s hour (time or date-time).
  Assumes a 24-hour clock.
  ...
  .../order_hdr/?order_shipped_ts__hour=20

- `minute` – Match on the time’s minutes (time or date-time).
  ...
  .../order_hdr/?order_shipped_ts__minute=30
You can also apply other lookup and arithmetic functions to temporal fields:

- **Date Range**
  
  For example, if we have a date-time field where we want to search for resources that have a value within a range, it is possible to chain two temporal filters together to search within a set date range:

  ```
  .../order_hdr/?order_shipped_ts__gte=2018-09-01T00:00:00&order_shipped_ts__lt=2018-10-01T00:00:00
  ```

  Or, it is possible to use the “range” lookup function:

  ```
  .../order_hdr/?order_shipped_ts__range=2018-09-01T00:00:00,2018-10-01T00:00:00
  ```

  However, since in this example we don’t have any specific time data, this could have also been accomplished more easily using the “month” lookup:

  ```
  .../order_hdr/?order_shipped_ts__month=09
  ```

  There may be multiple different ways to arrive at the same result when filtering. It is always desirable to be as specific as possible to minimize the result set and improve efficiency.

### Additional Lookups

- **isnull** – Boolean; Is the field’s value null?  
  This lookup is used to test if a field is null. This is a useful lookup as it can be used on any type of field to test for null.

  Example: Filtering sales order(s) for only those where the shipped timestamp is null:

  ```
  .../order_hdr/?order_shipped_ts__isnull=true
  ```

  This is important because it allows you to make this test for any field type. If, for example, you tried to filter on the field’s value directly (```.../order_hdr/?order_shipped_ts=null```), you would receive an error that “null” is not a valid date. Since the field is of type date-time, it is expecting a temporal value and is interpreting “null” as the input.

- **in** – Filter by values in a list
  
  This lookup function allows for filtering by a group of values. These values may be a mix of different types, but the type(s) should be consistent with the type of the field being filtered. The input is a comma-delimited list with no spaces between entries in the list.

  Example: Filter order_hdr by specific status id values:

  ```
  .../order_hdr/?status_id__in=10,30,90
  ```

  Or, it can be applied for filtering on a specific set of sales order numbers:

  ```
  .../order_hdr/?order_nbr__in=ORDER001,ORDER002,ORDER003
  ```

  It is also possible to use an “in” lookup with a single value to effectively function the same as an exact operator (“=”). The two following examples are equivalent in that they will return the same result set:

  ```
  .../order_hdr/?order_nbr=ORDER001
  .../order_hdr/?order_nbr__in=ORDER001
  ```

  The difference is that an “in” lookup in inherently slower because of the way the filter is built and applied when filtering the data. If you have a single value to match on, it is recommended to use “=” instead of “in”.

---

Copyright © 2019, 2020, Oracle and/or its affiliates.
- **range** – Filter for resources with value within an inclusive range.

  **Numeric range**
  .../order_hdr/?status_id__range=10,90

  **Date range**
  .../order_hdr/?order_shipped_ts__range=2018-09-01T00:00:00,2018-10-01T00:00:00

**Relational Resource Filtering**
It is possible to filter on any related field for the given entity. All related field names end with “_id” and are integers by default.

For example, the simplest and fastest performing related resource filter is to search directly on the resource’s id. An "id" is the unique value assigned to every resource. Using the "order_hdr" field, "facility_id", we could filter specifically for order belong to the facility with id "1":

.../order_hdr/?facility_id=1

Adding the "company_id" field is a very common thing to do, in order to filter resources by facility and company (assuming the company's id is also "1"):

.../order_hdr/?facility_id=1&company_id=1

But what if we wanted to filter by the value of a field belonging to the related resource. For example, what if we knew the facility and company codes, but didn’t yet know their respective "id" values. It is possible to filter on the related resource’s fields using double-underscore ("__") notation.

Assuming facility with id=1 has a code “FAC1” and company with id=1 has a code “COM1”:

.../order_hdr/?facility_id__code=FAC1&company_id__code=COM1

This is not as efficient as using just the "id" of the related resources since Gfapi will need to do an additional lookup for each related resource to filter on their respective "code" fields. It is recommended to cache client-side the "id" values of commonly used, static entities (like facility and company) in order to improve performance in high-throughput systems.

It is also possible to filter multiple levels deep with related resources. For example, in order to filter on the order's facility's parent company, we could further chain the facility field, "parent_company_id", as it is a related resource of "facility_id" and of entity type “company”:

.../order_hdr/?facility_id__parent_company_id=1

Again, you can also search on a related field:

.../order_hdr/?facility_id__parent_company_id__code=COM1

This is a handy and powerful tool for looking up resource sets based on related data. However, it is important to remember that as the relational filter depth increases, the performance may decrease as well since there is more work to be done to lookup related resource(s). Client-side caching and other performance methodologies are discussed in their own section.

**Chaining Multiple Filters**
It is possible to chain multiple filters on the same field. Each condition is just another key-value pair where the field is consistent. For example, if we wanted to filter the order_hdr entity to return those whose order_nbr starts with “ABC” and additionally contains the word “TEST”, we would write it as:

.../order_hdr/?order_nbr__startswith=ABC&order_nbr__contains=TEST
It is possible to chain together any number of different field and lookup combinations to arrive at your desired result set. However, it is important to note that the more filters applied, the more the performance may degrade. Therefore, it is always preferred to be as specific as possible when using filtering.

Resource Representations (GET)

Within the lgfapi entity module, JSON or XML resource representation(s) of entity(s) may be obtained through a GET request. A GET request is made for a specific entity in the format:

```plaintext
.../lgfapi/v10/entity/{entity_name}/
```

By default, each request is filtered by the requesting user’s eligible facility(s) and company(s). It is possible to add additional filter conditions in the URI query string in order to arrive at the data required. If, after filtering, no data is found, a 404 – Not Found error will be returned in the standard lgfapi response.

Furthermore, there are two conventions for how to request resource representation(s) – "list" and "retrieve". For the following examples, the "company" entity will be used.

**List**

A list request is used to fetch one or more object representations of an entity. The result set is based on the default facility/company context filters and any optional filter parameters provided in the URI. The default results set is comprised of all resources for the given entity that are eligible to the requesting user. Since the result set may be of an arbitrarily size, a paginated data set is always returned.

The representation for all eligible objects can be requested by not providing the query string portion of the URI:

```plaintext
.../lgfapi/v10/company/
```

Query string filter parameters may optionally be used to further narrow down the data set. For example, to filter additionally by company code "ABC", we would add the following:

```plaintext
.../lgfapi/v10/company/?code=ABC
```

**Retrieve**

A retrieve request is used to fetch a single resource by its integer "id" value. This is the most performant way to get a representation for a single resource where the "id" is known. The result set is not paginated. The "id" value is specified in the URI after the entity name:

```plaintext
.../lgfapi/v10/company/{id}/
```

For example, if we had previously looked up the company with code “ABC” and found its "id" value to be 1, we could retrieve its representation in the future by making a GET request to the URI:

```plaintext
.../lgfapi/v10/company/1/
```

Note that since the lookup is for a specific resource, no filters are allowed in the query string. It is permitted to pass in allowed non-filter reserved parameters like "format" and "fields". However, any pagination related query string parameters like "page_mode" are not supported since the returned representation is not paginated.
Note that .../lgfapi/v10/company/?id=1 is still considered a "list" style request and is paginated.

"Last-Modified" HTTP Header
If the requested resource exists and the data is temporally tracked, the Last-Modified HTTP header will be returned. This is the date-time that the resource was last updated. It is in iso-8601 format in the requesting user’s time zone. This can be cached client-side and used in conjunction with HEAD requests as an efficient way to check for resource modification.

Resource Representation Data Conventions
For both list and retrieve GET requests, the “format” query string parameter can be passed in order to convey the desired response format as “json” (default) or “xml”.

Hyperlink-Related Resource Representations
All resources use hyperlinked representations for related resource fields. These are the fields whose name ends with "_id". They represent another entity resource that can generate its own representation using the hyperlink provided. Lgfapi uses hyperlinked relationships to allow for users to crawl to the intended data sets. This allows for the preservation of RESTful principals as well as to keep the data interchange sizes manageable.

All related field representations contain three pieces of information:

1. "id" – The integer id value of the related resource
2. "key" – A string identifier for the related resource
3. "url" – A crawlable retrieve style hyperlink to the related resource
   • Both "id" and "key" are always provided. However, the value for "url" may be blank if the related resource it not one of the supported entities. In this case, it is not possible to build a hyperlink to the resource as it does not support generating its own representations.

For example, when getting a representation for the "company" entity where the company is of type Regular, the related field "company_type_id" would be represented like the following JSON string:

```json
{
  ...
  "company_type_id": {
    "id": 1,
    "key": "R",
    "url": "https://.../wms/lgfapi/v10/entity/company_type/1"
  },
  ...
}
```

Or, if the desired format is XML:

```xml
<company>
  ...
  <company_type_id>
    <id>1</id>
    <key>R</key>
    <url>https://.../wms/lgfapi/v10/entity/company_type/1</url>
  </company_type_id>
  ...
</company>
```

The only exception for the related field representation format is for status_id related fields. These fields are always represented as only the related resource’s integer "id" value. It is possible to get a
representation for any status-based entity by making a retrieve request. The only difference is that due to the volume of status fields on various entities, the integer value is used to reduce payload size.

For example, the “order_hdr” entity has the related field “status_id” for the entity “order_status”. It is represented on the “order_hdr” as just the “id” value:

```json
{
  ...
  "status_id": 10,
  ...
}
```

However, it is possible to get a representation of the status by making the request:

```
GET https://.../wms/lgfapi/v10/order_status/10
```

**IMPORTANT**
There are many related resource fields that are optional. If there is no linked resource, the field’s value will be “null” if using JSON or an empty tag if using XML. For more information, reference the entity’s field metadata for the “required” attribute.

### Related Data Sets

The related resources previously discussed all link to a single resource. However, it is possible that the current resource has a list many other linked resources of the same type. A good example is a sales order header that has one or more child details. As a convenience and additionally for guidance/performance reasons, many entity representations have additional hyperlinked relations to these data sets. These field names always end in “_set”.

Continuing the sales order header example, the order details set could be represented as the following in an order_hdr retrieve representation. Assume there are two detail line items and the “id” value of the order_hdr entity is “123”.

```
GET https://.../wms/lgfapi/v10/entity/order_hdr/123

{
  "id": 123,
  ...
  "order_dtl_set": {
    "result_count": 2,
    "url": "https://.../wms/lgfapi/v10/entity/order_dtl/?order_id=123"
  },
  ...
}
```

It’s important to note that unlike the “_id” related resources which have a retrieve style hyperlink to the specific resource, “_set” related representations use list style with query string filters in order to return a paginated list of 1 to n resource representations. Also, instead of giving the “id” and “key”, the related count is returned.

If no related resources are found for the set, the value will be “null” for JSON representations and an empty tag for XML.

### Field Selection

GET requests for the lgfapi entities support the “fields” query string parameters. It takes a comma-delimited list of field names for the entity and returns only those fields in the representation.
For example, to return only the "id" and "code" for all eligible companies using a list style request with no filters:

GET https://.../wms/lgfapi/v10/entity/company?fields=id,code

The "fields" parameter can be combined with filter parameters and other parameters with special meaning, like "format". Here is a more complex example if one wanted to search for all eligible companies of type regular and return only the "id" and "company" for each company entity found, in XML format:

GET https://.../wms/lgfapi/v10/entity/company?fields=id,code&format=xml&company_type_id=1

This can also be applied to retrieve style request for a specific resource:

GET https://.../wms/lgfapi/v10/entity/company/1?fields=id,code

This is an important tool when performance is of concern. If it is known ahead of time that only specific field values are required, narrowing the returned data set using the "fields" parameter can greatly reduce the overall payload size and remove the need for unnecessary field and/or relation lookups.

**Ordering**

By default, no ordering is applied to list style GET requests that can return 0 or more representations. This is done for performance considerations as applying ordering to any request may degrade performance, especially in the case of larger data sets.

It is possible to specify an order-by clause for list style requests using the "ordering" query string parameter. It accepts a comma-delimited list of field names by ordering priority.

For example, one could request all eligible companies and order by the type and then the code:

GET https://.../wms/lgfapi/v10/entity/company/?ordering=company_type_id,code

By default, fields are ordering ascending. To order by descending value, add a dash ("-"), before the field name in the ordering list. This can be applied to order first by company type ascending and then company code descending:

GET https://.../wms/lgfapi/v10/entity/company/?ordering=company_type_id,-code

Just like any other query string parameter, it may be chained with other parameters and filters.
Resource Existence and Modification (HEAD)

HTTP requests for lgfapi entities using the HEAD method are an efficient way to determine if a resource or list of resource(s) exists. Additionally, it is possible to determine if a specific resource has been modified since a target date-time. The HEAD method does not return any data in the body of the response. The only data returned is the response status code and any HTTP headers. Because HEAD requests do not have to know specifics about each resource and build a representation (like in a GET request), minimum data is transmitted and the server-side determinations can be optimized.

HEAD requests accept both retrieve and list style URI that same as a GET request. This can be used to check for the existence of a specific resource or filter for the existence of potentially many resources in a list.

“If-Modified-Since” HTTP Request Header

Entity HEAD requests allow for the requester to optionally pass the “If-Modified-Since” HTTP header in the request. This is only permitted for retrieve style requests when querying for a specific resource by id in the URL. The header’s value is the target date-time in iso-8601 format in the appropriate time zone. When provided, the value will be compared to the resource’s last modification time to determine if it has been modified since the header’s date-time. If the resource exists, and it has been modified, a 200 - Ok status code is returned. If it exists but has not been modified, a 304 – Not Modified status code is returned.

Not that if the entity does not support mod time tracking, the header is ignored and a 200 – Ok response code is returned meaning only that the resource exists.

The “If-Modified-Since” request header is typically used in conjunction with the “Last-Modified” response header that is returned with every retrieve style GET request for those entities that track mod timestamps. For example, a common scenario might start with a retrieve style GET request being made for a resource. The value of the “Last-Modified” response header is saved client-side for that resource. Sometime later, the client wants to check if the resource has been updated. A HEAD request can be made to determine if the resource has been modified since the original GET request by passing the last mod timestamp in the “If-Modified-Since” request header.

In scenarios where the updated resource representation is not needed, a HEAD request is much more efficient than a GET request. Or, it may be used to determine if a more expensive GET request is subsequently called to fetch the updated resource representation. It is also common to use HEAD request modification checks as a trigger mechanism for downsteam operations.

Response Statuses

The HTTP response status will be one of the following and vary depending on the outcome and if checking for existence or existence and modification of one or more resources. Note that this is not the full list of all possible response statuses. Rather, the following statuses are directly tied to this HTTP method’s functionality within lgfapi. For example, one can still receive a 401 status code if not providing valid user authentication credentials.

- 200 - Ok
  When checking for only existence, a 200 status code response means that the resource(s) exist. When additionally checking for modification, this status code confirms that the specific resource exists and has been modified.

- 304 – Not Modified
Only applicable when checking for modification of a specific resource using the 'If-Modified-Since' header. This status means that the resource exists but has not been modified since the input target date-time.

- **400 - Bad Request**
  For HEAD requests, it is possible to receive this status when using the 'If-Modified-Since' header with an invalid date-time value or format. This may also be returned if other invalid data is found, such as invalid query string filters.

- **404 - Not Found**
  No resource(s) were found based on the input provided. This may mean that either the resource(s) do not exist, or they do exist but the requesting user is not eligible for any of the resources.

For example, use a retrieve style request to check for the existence of a company entity with id=1:

```
HEAD https://.../wms/lgfapi/v10/entity/company/1
```

Or, it can be applied to a list style request with filters:

```
HEAD https://.../wms/lgfapi/v10/entity/company?code=ABC
```
Creating a Resource (POST)

Lgfapi allows for the creating and linking of a limited number of entity resources using an HTTP POST request. The new resource’s initial data set is passed in the body of the request, in the structure and formats outlined below. The requesting user must have the “lgfapi_create_access” permission. Also, the requesting user must be eligible for the facility/company context of the data being created.

Example request to create an IBLPN:

```
POST .../wms/lgfapi/v10/entity/iblpn/
```

Input Data

Data passed in the body of any POST request to the entity module requires the follow structure and data conventions.

Data Structure

Data is input in the request body in one of two sections:

- Fields – Initial field data. The “fields” section is used to pass in the initial field data required by the entity. Optional fields have a default and should be omitted from the “fields” data if you with the default to be applied. Lgfapi will attempt to use any data passed in the request body over the field default.

- Options – Additional/miscellaneous data. The “options” section is used to pass in extraneous data not directly required by the entity. A common example is the need to pass in a reason code when creating certain entities for the purposes of tracking against writing inventory history records.

**JSON Example**

```json
{
  "fields": {
    "string_field": "ABC",
    "decimal_field": 1.234
  },
  "options": {
    "reason_code": "RC"
  }
}
```

**XML Example**

```xml
<request>
  <fields>
    <string_field>ABC</string_field>
    <decimal_field>1.234</decimal_field>
  </fields>
  <options>
    <reason_code>RC</reason_code>
  </options>
</request>
```

Dates/Times

Temporal data must be iso-8601 format.
Related Resources

Relational fields (denoted by a field name ending in "_id") require the integer "id" value of the target resource. This can be obtained by making a GET request to the corresponding entity with appropriate filters.

Assuming that you already know the corresponding fields each have an "id" value of 1; when creating a new resource with the required related fields “facility_id” and “company_id”, the JSON POST request body is modeled as:

```json
{
   "fields": {
      "facility_id": 1,
      "company_id": 1
   }
}
```

If a related field is optional and not required as part of the initial resource creation, the field should be omitted to apply the default value.

Response Statuses

A non-paginated representation of the new resource will be returned in the body of the HTTP response in the desired format.

- **200 – Ok**
  A lookup was done and it was determined that the resource already exists. No new resource was created. Instead, the body of the response contains a representation of the existing resource. This is only applicable to certain entities.

- **201 - Created**
  The resource was successfully created.

- **400 - Bad Request**
  The request was invalid. This could be due to data validation failures, permission errors, or other missing requirements of the operation.

Validations

Field and object-level validations are applied before the new resource is created. Any errors will be returned the response body in the standard format. All related resources must be within the facility/company context of the resource being created. Meaning, users cannot link the new resource to any resources outside of its facility and/or company. For example, it is not possible to link an IBLPN to a pallet where the pallet is for a different facility or company than the IBLPN.

Nested Related Objects

Some entities, such as “inventory”, allow for the creation and association of some related objects within the request to create the inventory object. This allows for the creation of multiple related objects using a single API call instead of multiple requests.

The currently supported related objects are “batch_number” and “inventory_attribute”. Instead of passing in the "id" value of the related objects as the field definition’s value, you may alternatively insert a nested object representation. If the nested object does not exist, it will be created. If it does exist, no creation for that object takes place but in both cases it will be associated to the inventory object being created.

For example, when making a POST request to create an inventory object, it is valid to associate an existing batch using its "id" value:
It is also possible to send a nested representation of the batch object which will functionally act as “get or create”. The nested object must still pass all of the same validations as if it were being created independently and its “id” value passed in:

```
{
    "batch_number_id": { 
    "batch_nbr": "BATCH001",
    "item_id": 1,
    "expiry_date": "2019-01-01"
    },
    ...
}
```

**Supported Entities**

- **inventory_attribute**
  - Functions as get-or-create based on the provided attributes for the given facility and company combination.
- **batch_number**
  - Function as get-or-create based on the batch number for the given facility and company combination.
- **iblpn**
  - Creates an inbound container with no inventory.
- **inventory**
  - Creates inventory in either an iblpn or an active location.
  - Requires “reason_code” option for inventory history tracking.
  - Success results in inventory history adjustment(s) being generated.
  - Supports nested “batch_number” and “inventory_attribute” object creation.
- **inventory_lock**

Create an inventory lock that can applied to containers and locations.

**Updating a Resource (PATCH)**

Lgfapi allows you to update specific fields on a limited number of entity resources using an HTTP PATCH request. Only the desired changes are to be passed in the body of the request using the “fields” section (very similar to a create resource (POST) request). The requesting user must have the “lgfapi_update_access” permission and must be eligible for the facility/company context of the data being modified. Successful modification will additionally update the object’s “mod_ts” and “mod_user” fields.

The entities and fields that may be modified are limited at this time, with a few exceptions, to custom (“cust”) fields, where supported. These fields are for “pass through” data that generally has no functional significance.

Updates are restricted to a single object per request and the “id” of the target object is required as part of the resource URL.

The following is an example URL to update a sales order:
PATCH .../wms/lgfapi/v10/entity/order_hdr/123/

Input Data

Data passed in the body of any PATCH request to the entity module requires the following structure and data conventions.

- Fields - Field data with target value for update. The “fields” section is used to pass in the fields to update and the desired value. Any omitted fields will be unchanged.

  JSON example of updating the values of multiple “cust” fields:

  ```json
  {
    "fields": {
      "cust_field_1": "A",
      "cust_decimal_2": 1.234
    }
  }
  ```

Response Statuses

A non-paginated representation of the updated resource will be returned in the body of the HTTP response in the desired format.

- 200 – Ok
  The resource was successfully updated.

- 400 - Bad Request
  The request was invalid. This could be due to data validation failures, permission errors, or other missing or incomplete requirements.
## Entities and Fields for PATCH

### ib_shipment

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_date_1</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_2</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_3</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_4</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_5</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_1</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_2</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_3</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_4</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_5</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_number_1</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_2</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_3</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_4</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_5</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_6</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_7</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_8</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_9</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_10</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_11</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_12</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

### ib_shipment_dtl

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_date_1</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_2</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_3</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>cust_date_4</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_5</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_1</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_2</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_3</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_4</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_5</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_number_1</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_2</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_3</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_4</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_5</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_6</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_7</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_8</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_9</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_10</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_11</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_12</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**item_characteristics**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_attr_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_attr_2</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**load**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_6</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_7</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_8</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_9</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_10</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_date_1</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_2</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_3</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_4</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_5</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_1</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_2</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_3</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_4</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_5</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_number_1</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_2</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_3</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_4</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_5</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_2</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
### order

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_short_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_6</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_7</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_8</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_9</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_10</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_11</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_12</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>externally_planned_load_flg</td>
<td>Boolean</td>
<td>• Only valid if the order is less than Shipped status.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When updating the flag to false, any externally_planned_load_nbr values set on the corresponding order details will be removed.</td>
</tr>
<tr>
<td>stop_ship_flag</td>
<td>Boolean</td>
<td>• Update the stop_ship_flag on the order header if API call made is successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allowed order statuses for setting the stop_ship_flag to true: Created, Partially Allocated, Allocated, In-Picking, Picked, In-Packing, Packed, Loaded. If order status is shipped or cancelled, then respond with error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If order status is shipped or cancelled, then respond with error, other statuses should be ok.</td>
</tr>
</tbody>
</table>

### order_dtl

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_date_1</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_2</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_3</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_4</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_date_5</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_1</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_2</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_3</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_4</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_decimal_5</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_long_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>cust_number_1</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_2</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_3</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_4</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_number_5</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_6</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_7</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_8</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_9</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_10</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_11</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_short_text_12</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**purchase_order_hdr**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**purchase_order_dtl**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>stop_recv_flg</td>
<td>boolean</td>
<td></td>
</tr>
</tbody>
</table>

**work_order_hdr**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_field_1</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_2</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_3</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_4</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>cust_field_5</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
Entity Operations (GET/POST)

Many entities offer specialized operations in order to assist users in more complicated, or performance intensive operations. These operations can act on one or more resources and may affect entities beyond the one(s) targeted in the request. The URLs may follow a "list" or "retrieve" styles:

Format for an entity operation URL evocable for a specific resource by “id”:

`.../wms/lgfapi/v10/entity/{entity_name}/{id}/{operation_name}/`

Format for a "bulk" entity operation URL evocable for potentially multiple resources:

`.../wms/lgfapi/v10/entity/{entity_name}/{operation_name}/`

Entity operations are invoked in the same manner as previously discussed for GET and POST requests. Each operation has its own URL tied to the entity. Entity operations that use a GET request are still for obtaining a representation in the response body and do not modify data. Entity operations that use POST requests trigger an action or series of actions on the entity that can change resource state.

Response Status

Entity operations follow the response statuses previously discussed for GET and POST request, with one addition:

- **204 – No Content**
  This HTTP response status is returned when the request was successfully fulfilled, but there is no additional content to return to the requester. Users should interpret this as success and expect the response body to be empty.

Bulk Operations

Entities may also support “bulk” operation that allow the same operation to be run on one or more resources within a single request. There are several key differences and additional options that apply to bulk operations.

Parameter Data Filtering

Since bulk operations are capable of acting on one or more objects in a single request, the request body’s “parameter” data is required. This data is a series of one or more filter conditions that will be applied to identify the target list of objects. Each operation may have its own allowed set of filter conditions that can be applied. This may include allowing users to filter on related objects and using complex lookups such as “in” by the same double underscore (“__”) notation as in a GET request’s filters.
Note that all data is still automatically filtered by the user’s eligible facilities and companies and that the user is not permitted to run bulk operations on objects outside their allowed scope.

In general, all bulk operations allow for the filtering of objects by “id”. For example, a JSON request body’s parameters section for filtering on multiple object id’s would be:

```json
{
    "parameters": {
        "id__in": [1, 2, 3]
    }
}
```

Filtering on facility code and company code could be achieved by doing the following (assuming the entity and operation allow it):

```json
{
    "parameters": {
        "facility_id__code": "FAC1",
        "company_id__code": "COM1"
    }
}
```

The maximum number of objects that may be acted upon in a single request is dictated by the requesting user’s "Rows per Page" attribute. This is configurable per user but also applies in other areas of the application such as how many objects are returned per page in an Igifapi GET request, or in the UI when refreshing a page’s data grid.

**Commit Frequency**

All bulk operations are provided this additional “options” integer input parameter (default = 0). This parameter allows the requester to dictate at what frequency the changes are applied to each resource or group of resources being processed.

The default value of 0 specifies that no updates are committed unless all resources are processed successfully (all or nothing). All changes are rolled back on the first error, and only the first error is reported back to the user using the standard response.

A value of 1 indicates that the changes should be committed per resource successfully processed. Any error will only cause a failure and roll back of changes for the specific resource that failed. All errors will be accumulated and returned in the standardized bulk response format (see below).

Although a value > 1 is permitted, it is not advised that customers use this unless instructed to do so by support. This is typically only used for more advanced or larger data processing scenarios and for certain performance considerations.

**Response Status and Content**

When the commit frequency is 0, the bulk operations will give the standard error response format as previously documented, if any error is found. However, a different response status and standardized format is provided on total success or when the commit frequency value is > 0:

A 200 – OK response is returned for bulk operations along with a standardized bulk response having the following attributes:

- record_count – Total number of resources processed in the request.
- success_count – Number of successfully processed resources.
- failure_count – Number of unsuccessfully processed resources.
- **details** – A nested dictionary (key/value map) that provides additional details for any resources that failed during the processing of the operation.
  - The key to identify each resource and its failure is by default the resource's unique "id" value. However, a different identifying key may be returned per operation, as documented.
  - If no details are provided, the value will be null.

The following is a JSON example where 2 objects were processed, but one (having id=123) failed:

```json
{
    "record_count": 2,
    "success_count": 1,
    "failure_count": 1,
    "details": {
        123: "Invalid status."
    }
}
```
4. Supported Entity Operations

Describe Entity

GET .../wms/lgapi/v10/entity/{entity_name}/describe/

The describe operation is unique in that it is common and can be used on any entity. It returns a formatted representation of the entity’s metadata including any filterable “parameters” and all field definitions. This is the primary tool for obtaining details about a specific entity.

Response components:
- parameters – A list of fields that can be used for filtering the entity.
- fields – Field definitions and metadata for the entity.
  - type – The field data type
  - allow_blank – String fields only. Is an empty string value permitted?
  - max_length – String fields only. Max string length permitted.
  - required – Does the field require data.
  - default – If the fields is not required, the default value when no value is provided.
Inventory

Link Serial Numbers

POST .../wms/lgfapi/v10/entity/inventory/{id}/link_serial_nbrs/

This operation is used to link one or more serial numbers to a single inventory record. The “id” value of the target inventory record is required in the URI.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>serial_nbr_list</td>
<td>Array of Strings</td>
<td>X</td>
<td></td>
<td>A list of serial number strings to be linked to the target inventory record.</td>
</tr>
</tbody>
</table>

Bulk Update Inventory Attributes

POST .../wms/lgfapi/v10/entity/inventory/bulk_update_inventory_attributes/

This operation is used to update the inventory attributes of one or more inventory objects. Inventory in a Received or Located IBLPN and inventory in an active location may be modified. Inventory history adjustment records will be written for each inventory record successfully modified.

The attributes individually are not necessarily required, but in total at least one attribute must be provided to indicate a change. Additionally, an attribute value may or may not be required as dictated by other configuration such as the corresponding item’s attribute requirements or the location allowing mixing of attributes. Furthermore, the inventory cannot be or have been allocated.

An empty string is a valid value to indicate removing the value from the corresponding attribute. Any attribute that is omitted from the request data will retain its current value.

The “parameters” section of the request body is required in addition to the “options” section outlined below. Only the “id” parameter filter is valid. It may be used as “id__in” with an array of values.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>invn_attr_a</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_b</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_c</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_d</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_e</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_f</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_g</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_h</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_i</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_j</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_k</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_l</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_m</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_n</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_o</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Target attribute value.</td>
</tr>
<tr>
<td>options</td>
<td>commit_frequency</td>
<td>Integer</td>
<td></td>
<td>1</td>
<td>0 = Roll back on first error.</td>
</tr>
</tbody>
</table>
Location

update_active_inventory

The update_active_inventory API allows you to adjust the inventory quantity in an active location for a specific item. Only a single location and item may be updated per request.

Note: This is a new API meant to replace the existing legacy `update_active_inventory` API. The legacy API will eventually be retired so no further enhancements will be made to it. New functionality will instead be added to this API as part of the Iglfapi suite.

Regardless of the method used to identify the location, the following input is valid:

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>item_barcode</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Item identifier.</td>
</tr>
<tr>
<td>options</td>
<td>item_code</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Item identifier.</td>
</tr>
<tr>
<td>options</td>
<td>item_alternate_code</td>
<td>String</td>
<td>C</td>
<td></td>
<td>Item identifier.</td>
</tr>
<tr>
<td>options</td>
<td>adjustment_qty</td>
<td>Numeric</td>
<td>C</td>
<td></td>
<td>Non-zero adjustment quantity.</td>
</tr>
<tr>
<td>options</td>
<td>actual_qty</td>
<td>Numeric</td>
<td>C</td>
<td></td>
<td>Non-negative final quantity.</td>
</tr>
<tr>
<td>options</td>
<td>batch_nbr</td>
<td>String</td>
<td>N</td>
<td></td>
<td>Batch tied to target inventory.</td>
</tr>
<tr>
<td>options</td>
<td>expiry_date</td>
<td>Date</td>
<td>N</td>
<td></td>
<td>Expiration date tied to target inventory.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_X</td>
<td>String</td>
<td>N</td>
<td></td>
<td>Attributes A-O tied to the inventory.</td>
</tr>
<tr>
<td>options</td>
<td>reason_code</td>
<td>String</td>
<td>Y</td>
<td></td>
<td>Recorded on inventory history.</td>
</tr>
<tr>
<td>options</td>
<td>transaction_ref_nbr</td>
<td>String</td>
<td>N</td>
<td></td>
<td>Recorded on inventory history.</td>
</tr>
<tr>
<td>options</td>
<td>locn_capacity_check_flg</td>
<td>Boolean</td>
<td>N</td>
<td></td>
<td>Validate locations max units and volume?</td>
</tr>
<tr>
<td>options</td>
<td>company_id</td>
<td>Integer</td>
<td>N</td>
<td></td>
<td>Item’s company.</td>
</tr>
<tr>
<td>options</td>
<td>company_code</td>
<td>String</td>
<td>N</td>
<td></td>
<td>Item’s company’s code.</td>
</tr>
</tbody>
</table>

- Only one of `item_barcode`, `item_code`, or `item_alternate_code` is allowed.
- Only one of `actual_qty` or `adjustment_qty` is allowed.
- If positive change in quantity:
  - The provided `batch_nbr` will be created if it does not exist.
- Only one of `company_id` or `company_code` is allowed.
  - Although not required by the API, the company context may be necessary if there is ambiguity when identifying the item to adjust. This is common in 3PL scenarios where the same identifying information may be present for different items across companies for which the user is eligible.

Location Lookup by ID

`POST .../entity/location/{id}/update_active_inventory/`

The caller knows the unique `id` value of the active location, which is added to the request URL. No additional `parameters` data is required from the request body.
Location Lookup by Filters

POST .../entity/location/update_active_inventory/

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>barcode</td>
<td>String</td>
<td>Y</td>
<td>Location's barcode.</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id</td>
<td>Integer</td>
<td>N</td>
<td>Location's facility.</td>
</tr>
</tbody>
</table>

- Only a single location may be updated per request.
  - The `__in` lookup is not supported for `barcode`.
- `facility_id` supports string lookup by `code` using the double-underscore notation:
  - `facility_id__code`

Example Request Body:

```
{
  "parameters": {
    "facility_id__code": "FAC-1",
    "barcode": "LOCN1"
  },
  "options": {
    "item_barcode": "ITEM1234",
    "adjustment_qty": -10,
    "batch_nbr": "BATCH1234",
    "expiry_date": "2020-01-02",
    "invn_attr_a": "A",
    "invn_attr_b": "B",
    "reason_code": "RC",
    "transaction_ref_nbr": "TX123457890",
    "company_code": "COM-1"
  }
}
```
**Item**

**Image Upload**

The image_upload API allows you to update an image either by Item ID or Item by Filter.

**Assumptions**

- Only one item may be updated per request.
- An error will be returned if no items are found.
- An error will be returned if more than one item is found.

**Item by id**

```
POST .../entity/item/{id}/image_upload/
```

**Item by Filters**

```
POST .../entity/item/image_upload/
```

**Supported Item Filter Attributes**

The "parameters" section of the request body supports item filters when using this URL style.

- company_id (Required)
  - This additionally allows filtering on company code: "company_id__code"
- barcode
- part_a
- part_b
- part_c
- part_d
- part_e
- part_f
- item_alternate_code

**Example request body parameters:**

```
{
    "parameters": {
        "company_id__code": "COM1",
        "barcode": "ABC123"
    }
}
```
Request Image Data

Regardless of which URL is used, the image data is passed in the request body's "options" section in the "image_data" key. Data is required to be base64 encoded.

Example request body options:

```json
{
   "options": {
      "image_data": "ABC123"
   }
}
```

Trailer

First Available

The first_available API allows you to identify yard locations with available capacity. After fetching this API, you will get the first yard location with capacity based on the yard location putaway sequence. If the putaway sequence is not configured, the fetch will display according to the yard location pick sequence. After you get the location, you can use the locate to yard API to update the trailer location to the yard.

Identify yard location by capacity:

GET .../entity/location/yard/first_available

Request

The following are the Query String Filters for this API:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>String</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Facility context by code.</td>
</tr>
</tbody>
</table>

- Only one of "facility_id" or "facility_id__code" is allowed per request.
- If no additional context is provided, the user's default facility/company will be used.

Example Requests

GET .../entity/location/yard/first_available?facility_id=1
The following is an example GET request for facility ID:

GET .../entity/location/yard/first_available?facility_id=1

The following is an example GET request for facility ID code:

GET .../entity/location/yard/first_available?facility_id_code=STRAJB01

`locate_to_yard`

The `locate_to_yard` API allows the caller to update a trailer’s location to or within the yard. Regardless of the method used to identify the trailer, the following input is valid:

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>location_barcode</td>
<td>String</td>
<td>Y</td>
<td>Barcode of yard location.</td>
</tr>
</tbody>
</table>

`Trailer Lookup by ID`

`POST .../entity/trailer/{id}/locate_to_yard/`

The caller knows the unique `id` value of the trailer, which is added to the request URL. No additional `parameters` data is required from the request body.

**Example Request Body:**

```json
{
   "options": {
      "location_barcode": "LOCN-1"
   }
}
```

`Trailer Lookup by Filters`

`POST .../entity/trailer/locate_to_yard/`

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>trailer_nbr</td>
<td>String</td>
<td>Y</td>
<td>Trailer number to be moved.</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id</td>
<td>Integer</td>
<td>N</td>
<td>Trailer’s company.</td>
</tr>
</tbody>
</table>

- Only a single trailer may be moved per request.
  - The `__in` lookup is not supported for `trailer_nbr`.
- `company_id` additionally supports string lookup by `code` using the double-underscore notation:
  - `company_id__code`
Example Request Body:

```json
{
    "options": {
        "location_barcode": "LOCN-1"
    },
    "parameters": {
        "facility_id": 1,
        "company_id__code": "COM-1",
        "trailer_nbr": "TRLR-1"
    }
}
```

**remove_from_yard**

The **remove_from_yard** API allows the caller to release a trailer from its current yard location.

**Trailer Lookup by ID**

**POST** `.../entity/trailer/{id}/remove_from_yard/`

The caller knows the unique `id` value of the trailer, which is added to the request URL. No additional `parameters` data is required from the request body.

**Trailer Lookup by Filters**

**POST** `.../entity/trailer/remove_from_yard/`

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>trailer_nbr</td>
<td>String</td>
<td>Y</td>
<td>Trailer number to be removed.</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id</td>
<td>Integer</td>
<td>N</td>
<td>Trailer’s facility.</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id</td>
<td>Integer</td>
<td>N</td>
<td>Trailer’s company.</td>
</tr>
</tbody>
</table>

- Only a single trailer may be moved per request.
  - The `__in` lookup is not supported for `trailer_nbr`.
- `facility_id` and `company_id` both additionally support string lookup by `code` using the double-underscore notation:
  - `facility_id__code`
  - `company_id__code`
Example Request Body:

```json
{
    "parameters": {
        "facility_id": 1,
        "company_id__code": "COM-1",
        "trailer_nbr": "TRLR-1"
    }
}
```

Load check_in

The check_in API allows the caller to check-in an inbound or outbound load to a dock door. Regardless of the method used to identify the load, the following input is valid:

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>dock_nbr</td>
<td>String</td>
<td>Y</td>
<td>Dock door for check-in.</td>
</tr>
</tbody>
</table>

Load Lookup by ID

POST `/entity/load/{id}/check_in/`

The caller knows the unique `id` value of the trailer, which is added to the request URL. No additional `parameters` data is required from the request body.

Example Request Body:

```json
{
    "options": {
        "dock_nbr": "DOCK-1"
    }
}
```

Load Lookup by Filters

POST `/entity/load/check_in/`
<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>load_nbr</td>
<td>String</td>
<td>Y</td>
<td>Load for check-in.</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id</td>
<td>Integer</td>
<td>N</td>
<td>Load's facility.</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id</td>
<td>Integer</td>
<td>N</td>
<td>Load's company.</td>
</tr>
</tbody>
</table>

- Only a single load may be moved per request.
  - The `__in` lookup is not supported for `load_nbr`.
- `facility_id` and `company_id` both additionally support string lookup by `code` using the double-underscore notation:
  - facility_id__code
  - company_id__code

Example Request Body:

```json
{
    "parameters": {
        "facility_id__code": "FAC-1",
        "company_id__code": "COM-1",
        "load_nbr": "LOAD-1"
    },
    "options": {
        "dock_nbr": "DOCK-1"
    }
}
```
check_out

The check_out API allows the caller to check-out an inbound or outbound load from a dock door.

Load Lookup by ID

POST .../entity/load/{id}/check_out/

The caller knows the unique `id` value of the trailer, which is added to the request URL. No additional `parameters` data is required from the request body.

Load Lookup by Filters

POST .../entity/load/check_out/

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>load_nbr</td>
<td>String</td>
<td>Y</td>
<td>Load for check-in.</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id</td>
<td>Integer</td>
<td>N</td>
<td>Load’s facility.</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id</td>
<td>Integer</td>
<td>N</td>
<td>Load’s company.</td>
</tr>
</tbody>
</table>

- Only a single load may be moved per request.
  - The `__in` lookup is not supported for `load_nbr`.
- `facility_id` and `company_id` both additionally support string lookup by `code` using the double-underscore notation:
  - facility_id__code
  - company_id__code

Example Request Body:

```json
{
    "parameters": {
        "facility_id__code": "FAC-1",
        "company_id__code": "COM-1",
        "load_nbr": "LOAD-1"
    }
}
```
Ship Load

The Ship Load API allows you to ship a load by uploading the load via ID or filter.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>load_nbr</td>
<td>X</td>
<td>string</td>
<td>Load for shipping</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id</td>
<td></td>
<td>Integer</td>
<td>Facility context by id</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id__code</td>
<td></td>
<td>string</td>
<td>Facility context by code</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id</td>
<td></td>
<td>Integer</td>
<td>Company context by id</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id__code</td>
<td></td>
<td>string</td>
<td>Company context by code</td>
</tr>
</tbody>
</table>

Load Lookup by ID

POST .../entity/load/{id}/ship/

- No additional `parameters` data in the request body is required.

Load Lookup by Filters

POST .../entity/load/ship/

Example Request Body:

```json
{
  "parameters": {
    "facility_id__code": "FAC-1",
    "company_id__code": "COM-1",
    "load_nbr": "LOAD-1"
  }
}
```

This API includes the following features:

- Supports the ship load transaction for a load that is in the Loaded/ Loading Started /Checked Out status.
- An error is displayed if the load is in a "Cancelled", "Ship Load In Progress", or "Shipped" status.
- The shipload transaction can be performed either by providing the id or code for the company/facility along with the load number.
- A Ship Load confirmation file is generated after the load is shipped.
Once a load is shipped via the Ship Load API, the following applies to Inventory History Transaction (IHT) records:

- Inventory history IHT-3 '3 - Container Shipped' is written with respect to each container present on the load.
- For shipped loads with OBLPNs associated with asset inventory history, IHT-58 '58 - Asset Shipped' is written with respect to each OBLPN associated with an asset.
- Inventory history IHT-60 '60 - Load Shipped File' is written for the outbound Load shipped.

The Ship Load API supports the following validations:

**Ship Load API supports Order type with "Single Order on multiple Loads":**

- If "Single Order on multiple Loads" is set to "Do not Allow" in the order type, the system displays the error message: "Load has Order/s marked to Prevent one order on different loads with Error."
- When an order is in Packed status but only some of the packed OBLPNs are loaded.
- When an order is in Packed status but some OBLPNs are loaded to different loads.
- For OBLPNs with pending audit if the Company parameter "ALLOW_LOAD_SHIP_WITH_AUDIT_PENDING" is set to no.

Company Parameter REQD_FIELDS_FOR_SHIPPING is defined:

- When the required fields configured for the parameter 'REQD_FIELDS_FOR_SHIPPING' are not defined for the targeted load.
- When one of the container item on the load is serial number tracked and the number of serial numbers allocated do not match with the count of serial numbers present in the container.

**Serial Number Validations**

- If company parameter ALLOW_LOAD_SHIP_WITH_AUDIT_PENDING = False and company parameter SERIAL_NUMBER_TRACKING_LEVEL is 1 or 2
- If company parameter SERIAL_NUMBER_TRACKING_LEVEL is 0 or Non serial tracked items exist and company parameter ALLOW_LOAD_SHIP_WITH_AUDIT_PENDING = False
- This API will not show you any warning message like the UI or RF, and it will proceed with the Ship Load transaction.
- The Ship Load API does not generate multiple outbound files.
Container

The “iblpn” and “oblpn” entities are derived from the “container” entity and have access to all of the following entity operations, in addition to their own.

Get Sales Orders

GET .../wms/lgfapi/v10/entity/container/{id}/orders/

Returns a paginated representation of “order_hdr” entities for all sales order(s) allocated against the inbound or outbound container.

Lock Container

POST .../wms/lgfapi/v10/entity/container/{id}/lock/

Apply one or more inventory locks to the target inbound or outbound container.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>lock_code_list</td>
<td>Array of Strings</td>
<td>X</td>
<td></td>
<td>Inventory lock code(s) to be applied.</td>
</tr>
</tbody>
</table>

Bulk Lock Container

POST .../wms/lgfapi/v10/entity/container/bulk_lock/

Apply one or more inventory locks to one or more inbound or outbound container(s).

The “parameters” section of the request body is required in addition to the “options” section outlined below. Only the “id” parameter filter is valid. It may be used as “id__in” with an array of values.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>lock_code_list</td>
<td>Array of Strings</td>
<td>X</td>
<td></td>
<td>Inventory lock(s) to be applied.</td>
</tr>
<tr>
<td>options</td>
<td>commit_frequency</td>
<td>Integer</td>
<td></td>
<td>1</td>
<td>0 = Roll back on first error. 1 = Commit per object.</td>
</tr>
</tbody>
</table>
Unlock Container

*POST* .../*wms/lgfapi/v10/entity/container/[id]/unlock/

Remove one or more inventory locks to the target inbound or outbound container.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>lock_code_list</td>
<td>Array of Strings</td>
<td>X</td>
<td>Inventory lock code(s) to be removed.</td>
</tr>
</tbody>
</table>

Bulk Unlock Container

*POST* .../*wms/lgfapi/v10/entity/container/bulk_unlock/

Remove one or more inventory locks from one or more inbound or outbound container(s).

The “parameters” section of the request body is required in addition to the “options” section outlined below. Only the “id” parameter filter is valid. It may be used as “id__in” with an array of values.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>lock_code_list</td>
<td>Array of Strings</td>
<td>X</td>
<td></td>
<td>Inventory lock(s) to be removed.</td>
</tr>
<tr>
<td>options</td>
<td>commit_frequency</td>
<td>Integer</td>
<td>1</td>
<td>1</td>
<td>0 = Roll back on first error. 1 = Commit per object.</td>
</tr>
</tbody>
</table>

Palletize Container

*POST* .../*entity/container/[id]/palletize/

Allows you to palletize an Inbound or Outbound LPN.

The “parameters” section of the request body is required in addition to the “options” section outlined below. Only the “id” parameter filter is valid. It may be used as “id__in” with an array of values.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>Container_nbr</td>
<td>X</td>
<td>String</td>
<td>IB or OB LPN to be linked. &quot;_in&quot; lookup is not supported.</td>
</tr>
<tr>
<td>parameters</td>
<td>Facility_id</td>
<td></td>
<td>Integer</td>
<td>Container’s facility</td>
</tr>
<tr>
<td>Parameters</td>
<td>Company_id</td>
<td></td>
<td>Integer</td>
<td>Container’s company</td>
</tr>
</tbody>
</table>

Example

```json
{
    "parameters": {
        "facility_id": 1,
        "company_id": 1,
```
Both `facility_id` and `company_id` also support filtering on `code` as well.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>Facility_id</td>
<td></td>
<td>string</td>
<td></td>
<td>Container’s facility</td>
</tr>
<tr>
<td>Parameters</td>
<td>Company_id</td>
<td></td>
<td>string</td>
<td></td>
<td>Container’s company</td>
</tr>
</tbody>
</table>

**Functional Request Data**

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>pallet_nbr</td>
<td>X</td>
<td>string</td>
<td></td>
<td>Pallet number to be used for palletizing IB or OBLPN's.</td>
</tr>
<tr>
<td>options</td>
<td>pallet_position</td>
<td></td>
<td>string</td>
<td></td>
<td>Position of Inbound or OBLPN during palletization.</td>
</tr>
<tr>
<td>options</td>
<td>allow_mix_pa_types_Flg</td>
<td>boolean</td>
<td>False</td>
<td></td>
<td>whether to allow mixing of LPN's with different PA types on a single pallet.</td>
</tr>
<tr>
<td>options</td>
<td>allow_mix_dest_shipto</td>
<td></td>
<td>string</td>
<td></td>
<td>valid values to be passed are</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Validate Ship To</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Validate Destination</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Validate Ship To and Destination</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ignore Ship To and Destination</td>
</tr>
</tbody>
</table>
"options": {
    "pallet_nbr": "PLT001",
    "pallet_position": "01",
    "allow_mix_pa_types_flg": false
    "allow_mix_dest_ship_to": Ignore Ship To and Destination
}

**Depalletize Inbound/Outbound LPN**

Allows you to depalletize an Inbound or Outbound LPN so you do not have to use RF guns for performing depalletization in automated guided facilities.

**Identify container by ID:**

**POST .../entity/container/[id]/depalletize/**

- The specific inbound or outbound LPN's id value is known and is provided in the URL.
- No additional `parameters` data in the request body is required.

**Identify container by Filters**

**POST .../entity/container/depalletize/**

- Igfapi provides mechanism to determine the container entity to be dissociated with pallet.
- The `parameters` section of the request body will allow for the users to identify the specific OBLPN

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>Container_nbr</td>
<td>X</td>
<td>String</td>
<td>IB or OB LPN to be linked. &quot;_in&quot; lookup is not supported.</td>
</tr>
<tr>
<td>parameters</td>
<td>Facility_id</td>
<td></td>
<td>Integer</td>
<td>Container’s facility</td>
</tr>
<tr>
<td>Parameters</td>
<td>Company_id</td>
<td></td>
<td>Integer</td>
<td>Container’s company</td>
</tr>
<tr>
<td>parameters</td>
<td>type</td>
<td></td>
<td>String</td>
<td>Container’s type “I” or “O”.</td>
</tr>
</tbody>
</table>

```json
{
    "parameters": {
        "facility_id": 1,
        "company_id": 1,
        "container_nbr": "LPN001",
        "type": "O"
    }
}
```
Both `facility_id` and `company_id` also support filtering on `code` as well.

```json
{
  "parameters": {
    "facility_id__code": "FAC-1",
    "company_id__code": "COM-1",
    "container_nbr": "LPN001",
    "type": "O"
  }
}
```

### Pick-Pack

#### Pick Confirm

The Pick Confirm API allows you to perform cubed or non cubed picking.

*Note: This is a new API meant to replace the existing legacy `pick_confirm` API. The legacy API will eventually be retired so no further enhancements will be made to it. New functionality will instead be added to this API as part of the lgfapi suite.*

This API supports features of the legacy API including the following new parameters:

-  o  **mhe_mode_flg** - true/false; default true
-  o  **async_flg** - true/false; default true
-  o  **short_flg** - true/false; default false
   -  ▪  Replaces using the legacy "action_code" = "SHORT".

Pick Confirm API can be called using the following POST request:

```
POST ..lgfapi/v10/pick_pack/pick_confirm/
```
## Request Parameters

### Pick List

These represent the parameters required for a single pick/short:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Facility context by code.</td>
</tr>
<tr>
<td>company_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Company context by id.</td>
</tr>
<tr>
<td>company_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Company context by code.</td>
</tr>
<tr>
<td>wave_nbr</td>
<td>X</td>
<td>String</td>
<td></td>
<td>Associated wave.</td>
</tr>
<tr>
<td>order_nbr</td>
<td>X</td>
<td>String</td>
<td></td>
<td>Associated sales order.</td>
</tr>
<tr>
<td>item_alternate_code</td>
<td>C</td>
<td>String</td>
<td></td>
<td>Item identifier.</td>
</tr>
<tr>
<td>item_barcode</td>
<td>C</td>
<td>String</td>
<td></td>
<td>Item identifier.</td>
</tr>
<tr>
<td>qty</td>
<td>X</td>
<td>Number</td>
<td>0</td>
<td>Quantity to be acted upon.</td>
</tr>
<tr>
<td>batch_nbr</td>
<td></td>
<td>String</td>
<td></td>
<td>Inventory batch/lot.</td>
</tr>
<tr>
<td>uom_qty</td>
<td></td>
<td>Number</td>
<td></td>
<td>Filter on Case or Pack quantity when searching for allocations.</td>
</tr>
<tr>
<td>allocation_uom</td>
<td></td>
<td>String</td>
<td></td>
<td>&quot;UNITS&quot;, &quot;PACKS&quot;, or &quot;CASES&quot;.</td>
</tr>
<tr>
<td>reason_code</td>
<td></td>
<td>String</td>
<td></td>
<td>Reason for short.</td>
</tr>
<tr>
<td>pick_location</td>
<td>C</td>
<td>String</td>
<td></td>
<td>From location.</td>
</tr>
<tr>
<td>from_container_nbr</td>
<td>C</td>
<td>String</td>
<td></td>
<td>From container.</td>
</tr>
<tr>
<td>to_container_nbr</td>
<td>C</td>
<td>String</td>
<td></td>
<td>LPN inventory is packed into. Not required for short.</td>
</tr>
<tr>
<td>update_inventory_on_short_flg</td>
<td></td>
<td>Boolean</td>
<td>False</td>
<td>Also short source inventory on pick short?</td>
</tr>
<tr>
<td>close_container_status</td>
<td></td>
<td>String</td>
<td>&quot;packed&quot;</td>
<td>Final OBLPN status: &quot;picked&quot; or &quot;packed&quot;.</td>
</tr>
<tr>
<td>short_on_close_flg</td>
<td></td>
<td>Boolean</td>
<td>False</td>
<td>Should any remaining unpacked quantity shorted?</td>
</tr>
<tr>
<td>mhe_system_code</td>
<td>C</td>
<td>String</td>
<td></td>
<td>MHE system.</td>
</tr>
<tr>
<td>short_flg</td>
<td></td>
<td>Boolean</td>
<td>False</td>
<td>Is this a short?</td>
</tr>
</tbody>
</table>
Validations

- Facility must be in user's eligible facilities and not be ambiguous.
- Possible if there is a Store and a DC with the same code.
- Company must be in user's eligible companies.
- If facility or company context is not included in the input parameters, user defaults are used.
- User cannot pass both "facility_id" and "facility_id__code" in the same request.
- User cannot pass both "company_id" and "company_id__code" in the same request.
- "mhe_system_code" is required if "mhe_mode_flg" is True.
- Only one of "item_alternate_code" or "item_barcode" is allowed.
- Only one of "pick_location" or "from_container_nbr" is allowed.
- "to_container_nbr" is required for "pick" operation, but is not required for "short".

Request-Level Flags

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mhe_mode_flg</td>
<td>Boolean</td>
<td>True</td>
<td></td>
<td>When true, enforce that &quot;mhe_system_code&quot; is provided.</td>
</tr>
<tr>
<td>async_flg</td>
<td>Boolean</td>
<td>True</td>
<td></td>
<td>Run API asynchronously?</td>
</tr>
</tbody>
</table>

The following is an example JSON request:

```json
{
    "mhe_mode_flg": false,
    "async_flg": false,
    "pick_list": [
        {
            "facility_id__code": "FAC",
            "company_id": 1,
            "wave_nbr": "WAVE001",
            "order_nbr": "ORDER001",
            "item_barcode": "ITEM1234",
            "qty": 10,
            "from_container_nbr": "IBLPN0001",
            "to_container_nbr": "OBLPN0001",
            "short_flg": false
        }
    ]
}
```
The following is an example XML request:

```
<request>
  <mhe_mode_flg>false</mhe_mode_flg>
  <async_flg>false</async_flg>
  <pick_list>
    <list-item>
      <facility_id__code>FAC</facility_id__code>
      <company_id>1</company_id>
      <wave_nbr>WAVE001</wave_nbr>
      <order_nbr>ORDER001</order_nbr>
      <item_barcode>ITEM1234</item_barcode>
      <qty>10</qty>
      <from_container_nbr>IBLPN0001</from_container_nbr>
      <to_container_nbr>OBLPN0001</to_container_nbr>
      <short_flg>false</short_flg>
    </list-item>
  </pick_list>
</request>
```

Close LPN

The close_lpn API allows you to close an LPN during picking/packing. This API replaces the legacy pick confirm API when the action code is closed. While performing pick and pack operations (either non cubed active picking or cubed picking), the Close action code indicates to WMS that the Outbound LPN being picked needs to be closed.

Note: This is a new API meant to replace the existing legacy `close_lpn` API. The legacy API will eventually be retired so no further enhancements will be made to it. New functionality will instead be added to this API as part of the lgfapi suite.

This API supports features of the legacy API including the following new parameter:

- async_flg - true/false; default true

Close LPN API can be called using the following POST request:

```
POST ..lgfapi/v10/pick_pack/close_lpn/
```
Request Parameters

The following table provides details about the query string parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Facility context by code.</td>
</tr>
<tr>
<td>company_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Company context by id.</td>
</tr>
<tr>
<td>company_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Company context by code.</td>
</tr>
<tr>
<td>to_container_nbr</td>
<td>X</td>
<td>String</td>
<td></td>
<td>To OBLPN.</td>
</tr>
<tr>
<td>close_container_status</td>
<td></td>
<td>String</td>
<td>&quot;packed&quot;</td>
<td>Final OBLPN status: &quot;picked&quot; or &quot;packed&quot;.</td>
</tr>
<tr>
<td>short_on_close_flg</td>
<td></td>
<td>Boolean</td>
<td>False</td>
<td>Should any remaining unpacked quantity shorted?</td>
</tr>
<tr>
<td>update_inventory_on_short_flg</td>
<td></td>
<td>Boolean</td>
<td>False</td>
<td>Also short source inventory on pick short?</td>
</tr>
<tr>
<td>reason_code</td>
<td></td>
<td>String</td>
<td></td>
<td>Reason for short.</td>
</tr>
<tr>
<td>async_flg</td>
<td></td>
<td>Boolean</td>
<td>True</td>
<td>Run API asynchronously?</td>
</tr>
</tbody>
</table>

The following is an example JSON request:

```
{
    "facility_id__code": "FAC",
    "company_id": 1,
    "to_container_nbr": "OBLPN001",
    "close_container_status": "picked",
    "short_on_close_flg": true,
    "async_flg": true
}
```

Wave Complete

The Wave Complete API replaces the legacy API when the action code is Complete. This is an indicator to inform WMS that all picks are completed for that wave, and there are no more picks outstanding.

*Note: This is a new API meant to replace the existing legacy `close_lpn` API. The legacy API will eventually be retired so no further enhancements will be made to it. New functionality will instead be added to this API as part of the lgfapi suite.*

This API supports features of the legacy API including the following new parameter:

- **async_flg** - true/false; default true
- When false:
- Instead of submitting a celery task at the end for later processing, it should be immediately processed and a response returned.
- On success, return a 204 - "No Content" HTTP response status with no response body.
- When true: Return HTTP response status 202 - "Accepted" with no response body.
- Signals that we received the request and it was successfully submitted for processing.

The Wave Complete API can be called using the following POST request:

```
POST ..lgfapi/v10/pick_pack/wave_complete/
```

The following table provides details about the query string parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Facility context by code.</td>
</tr>
<tr>
<td>company_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Company context by id.</td>
</tr>
<tr>
<td>company_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Company context by id.</td>
</tr>
<tr>
<td>wave_nbr</td>
<td>X</td>
<td>String</td>
<td></td>
<td>Associated wave.</td>
</tr>
<tr>
<td>update_inventory_on_short_flg</td>
<td></td>
<td>Boolean</td>
<td>False</td>
<td>Also short source inventory on pick short?</td>
</tr>
<tr>
<td>close_container_status</td>
<td></td>
<td>String</td>
<td>&quot;packed&quot;</td>
<td>Final OBLPN status: &quot;picked&quot; or &quot;packed&quot;.</td>
</tr>
<tr>
<td>reason_code</td>
<td></td>
<td>String</td>
<td></td>
<td>Reason for short.</td>
</tr>
<tr>
<td>mhe_system_code</td>
<td></td>
<td>String</td>
<td></td>
<td>MHE system.</td>
</tr>
<tr>
<td>async_flg</td>
<td></td>
<td>Boolean</td>
<td>True</td>
<td>Run API asynchronously?</td>
</tr>
</tbody>
</table>

The following is an example JSON request JSON:

```json
{
    "facility_id__code": "FAC",
    "company_id": 1,
    "wave_nbr": "WAVE001",
    "update_inventory_on_short_flg": true,
    "async_flg": true
}
```

**Task**

**Next Task**

The next_task API allows you to determine the next task via an API operation.

You can search for the next task using the following GET request:
GET .../entity/task/next_task

The following table provides details about the query string parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>Integer</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>String</td>
<td></td>
<td>Facility context by code.</td>
</tr>
<tr>
<td>location_barcode</td>
<td></td>
<td>String</td>
<td></td>
<td>User's current location.</td>
</tr>
<tr>
<td>task_type</td>
<td></td>
<td>String</td>
<td></td>
<td>Required task type.</td>
</tr>
<tr>
<td>ordering_rule</td>
<td></td>
<td>String</td>
<td></td>
<td>Order tasks by rule name.</td>
</tr>
</tbody>
</table>

**Facility ID/Facility Code**

- If a value isn't provided, the user's default facility context will be used.
- Task look up is done relative to the user's facility and eligible company contexts.

**Location Barcode**

- If provided, search for task within the same location area (if available) and/or pick sequence (if available).

**Task Type**

- If provided, search for task only of the given type.

**Ordering Rule**

- If provided, order the found tasks by the corresponding field(s) and return the top result.
- The value accepted by the API is that of the Task Ordering Rule's description.

The following is an example GET request using location barcode:

GET ...
/entity/task/next_task?location_barcode=MY_LOCN_BRC&D&task_type=MY_TASK_TYPE&ordering_rule=MY_RULE
**IBLPN**

The "iblpn" entity is derived from the "container" entity and therefore also has access to all of its entity operations, in addition to the following.

**Direct Consume**

POST .../wms/lgfapi/v10/entity/iblpn/[id]/direct_consume/

Consume a Received or Located IBLPN and update its inventory to zero. This will write IBLPN consumed inventory history records.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>reason_code</td>
<td>String</td>
<td>X</td>
<td></td>
<td>Used for inventory history tracking.</td>
</tr>
</tbody>
</table>

**direct_consume**

The `options` parameters, `transaction_ref_nbr`, may now be passed in the request body. This parameters will be added to any CNTR_CONSUMED inventory history records created as part of the API’s execution. The inventory history field `ref_code_3` will now be set as “TRN”. The value of `ref_value_3` will be that of `transaction_ref_nbr` or an empty string.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>transaction_ref_nbr</td>
<td>String</td>
<td>N</td>
<td></td>
<td>Max length of 250 characters.</td>
</tr>
</tbody>
</table>

**Example Request Body:**

```json
{
    "options": {
        "reason_code": "IT",
        "transaction_ref_nbr": "TX12345"
    }
}
```

**Composite Create**

POST .../wms/lgfapi/v10/entity/iblpn/composite_create/

This operation allows for the creation of a Received or Located IBLPN along with one or more inventory records in a single request. Furthermore, it allows for the creating and/or association of the inventory’s corresponding batch and inventory attribute, where applicable. This API follows all of the same validations and extended actions, such as writing inventory history, as the standalone create (POST) APIs for each entity, but brings them together in a single API.

Furthermore, this API takes advantage of allowing for the input of nested data, such as batch and inventory attribute, which will allow for those objects to be created or retrieved if they already exist. The use of the related objects “id” value is still permitted as well. All objects must have the same facility and company context as the IBLPN being created, and must still pass all standard user eligibility validations.
This API also has unique data structure requirements that mimic those of the individual entity’s create (POST) field inputs. It also allows for the definition of a global context where “facility_id” and “company_id” may be defined at the top level of the data and inherited by each object, if not defined on the object.

The following is an example of JSON request data where the facility/company context is defined at the top level and using the “id” values of “batch_number_id” and “invn_attr_id” to associate those objects that already exist. The defined top-level facility and company will be applied to the iblpn and inventory objects being created. The existing batch and inventory attribute objects being associated to the inventory must be of the same context.

Note that even though “inventory” does not have a “company_id” field, the company is determined from the associated item’s company and must also pass validations.

```json
{
  "fields": {
    "facility_id": 1,
    "company_id": 1,
    "iblpn": {
      "container_nbr": "IBLPN000001",
      "status_id": 30,
      "curr_location_id": 28536
    },
    "inventory": [
      {
        "item_id": 1,
        "curr_qty": 1.2345,
        "batch_number_id": 1,
        "invn_attr_id": 1
      }
    ]
  },
  "options": {
    "reason_code": "IT"
  }
}
```
The following is an example of JSON request data where the facility/company context is defined per object and using the "id" values of "batch_number_id" and "invn_attr_id" to associate those objects that already exist. Also demonstrates creating multiple inventory records for different item/batch/attribute combinations in a single IBLPN:

```
{
  "fields": {
    "iblpn": {
      "facility_id": 1,
      "company_id": 1,
      "container_nbr": "IBLPN000002",
      "status_id": 10
    },
    "inventory": [
      {
        "facility_id": 1,
        "item_id": 1,
        "curr_qty": 1.2345,
        "batch_number_id": 1,
        "invn_attr_id": 1
      },
      {
        "facility_id": 1,
        "item_id": 2,
        "curr_qty": 10,
        "batch_number_id": 2,
        "invn_attr_id": 2
      }
    ]
  },
  "options": {
    "reason_code": "IT"
  }
}
```
The following is an example of JSON request data where the facility/company context is defined at the top level and the "id" values of "batch_number_id" and "invn_attr_id" have been replaced with nested objects to create and associate those objects, which may or may not already exist:

```json
{
  "fields": {
    "facility_id": 1,
    "company_id": 1,
    "iblpn": {
      "container_nbr": "IBLPN000003",
      "status_id": 10
    },
    "inventory": [
      {
        "item_id": 3,
        "curr_qty": 1,
        "batch_number_id": {
          "batch_nbr": "BATCH001",
          "item_id": 3,
          "expiry_data": "2019-01-01"
        },
        "invn_attr_id": {
          "invn_attr_a": "A",
          "invn_attr_b": "B",
          "invn_attr_c": "C"
        }
      }
    ],
    "options": {
      "reason_code": "IT"
    }
  }
}
```
**modify_item_qty**

The IBLPN **modify_item_qty** API allows the caller to adjust item inventory in a “Received” or “Located” IBLPN. You can only update a single IBLPN and item per request.

Regardless of the method used to identify the IBLPN, the following input is valid:

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>item_barcode</td>
<td>String</td>
<td>C</td>
<td>Item identifier.</td>
</tr>
<tr>
<td>options</td>
<td>item_alternate_code</td>
<td>String</td>
<td>C</td>
<td>Item identifier.</td>
</tr>
<tr>
<td>options</td>
<td>adjustment_qty</td>
<td>Numeric</td>
<td>Y</td>
<td>Non-zero adjustment quantity.</td>
</tr>
<tr>
<td>options</td>
<td>batch_nbr</td>
<td>String</td>
<td>N</td>
<td>Batch tied to target inventory.</td>
</tr>
<tr>
<td>options</td>
<td>expiry_date</td>
<td>Date</td>
<td>N</td>
<td>Expiration date tied to target inventory.</td>
</tr>
<tr>
<td>options</td>
<td>invn_attr_X</td>
<td>String</td>
<td>N</td>
<td>Attributes A-O tied to the inventory.</td>
</tr>
<tr>
<td>options</td>
<td>reason_code</td>
<td>String</td>
<td>Y</td>
<td>Recorded on inventory history.</td>
</tr>
<tr>
<td>options</td>
<td>transaction_ref_nbr</td>
<td>String</td>
<td>N</td>
<td>Recorded on inventory history.</td>
</tr>
</tbody>
</table>

- Only one of `item_barcode` or `item_alternate_code` is allowed.
- IBLPN inventory matching is restrictive and does not support wildcard searches:
  - If no `batch_nbr` is provided, only match IBLPN inventory without a batch.
  - If no `expiry_date` is provided, only match IBLPN inventory without expiration.
  - If no `invn_attr_X` value is provided for A-O, it will be treated as blank.

**IBLPN Lookup by ID**

POST .../entity/iblpn/{id}/modify_item_qty/

Caller knows the unique `id` value of the IBLPN, which is added to the request URL. No additional `parameters` data is required from the request body.

**IBLPN Lookup by Filters**

POST .../entity/iblpn/modify_item_qty/

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>container_nbr</td>
<td>String</td>
<td>Y</td>
<td>IBLPN to be adjusted.</td>
</tr>
<tr>
<td>parameters</td>
<td>facility_id</td>
<td>Integer</td>
<td>N</td>
<td>IBLPN’s facility.</td>
</tr>
<tr>
<td>parameters</td>
<td>company_id</td>
<td>Integer</td>
<td>N</td>
<td>IBLPN’s company.</td>
</tr>
</tbody>
</table>

- Only a single IBLPN may be moved per request.
  - The `__in` lookup is not supported for `container_nbr`.
- `facility_id` and `company_id` both additionally support string lookup by `code` using the double-underscore notation:
  - `facility_id__code`
  - `company_id__code`
Example Request Body:

{
    "parameters": {
        "facility_id__code": "FAC-1",
        "company_id": 2,
        "container_nbr": "IBLPN1234"
    },
    "options": {
        "item_barcode": "ITEM1234",
        "adjustment_qty": -10,
        "batch_nbr": "BATCH1234",
        "expiry_date": "2020-01-02",
        "invn_attr_a": "A",
        "invn_attr_b": "B",
        "reason_code": "RC",
        "transaction_ref_nbr": "TX123457890"
    }
}
OBLPN

The "oblpn" entity is derived from the "container" entity and therefore also has access to all of its entity operations, in addition to the following.

Mark Delivered

POST .../wms/lgfapi/v10/entity/oblpn/{id}/mark_delivered/
Updates a Shipped OBLPN to Delivered status and writes container delivered inventory history.

create_from_iblpn

The OBLPN `create_from_iblpn` API allows you to create an OBLPN in Outbound Ready status and allocate inventory from a designated IBLPN in a single request. Additionally allows the caller to trigger packing of the OBLPN.

POST .../entity/oblpn/create_from_iblpn/

Assumptions

1. All allocation data must have the same facility and company context as the OBLPN.
   a. Allocations may be for multiple sales orders from multiple IBLPNs for different items as long as the facility/company context is consistent with the created OBLPN.
2. Sales order status will be recalculated on success.
3. IBLPN status will be recalculated on success.
4. Inventory history is only written if the OBLPN is packed.

Request Body Data

The request body data utilizes the 3 categories in the following ways:

1. `fields` – The initial data required to create the OBLPN.
2. `parameters` – List of data defining allocations.
3. `options` – Additional functional data.

OBLPN Fields Data

The OBLPN’s initial data is defined in the `fields` section of the request under the `oblpn` key. This is similar to the request body data requirements when creating an LPN directly through the entity’s create mechanism.

Supported fields:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td>Integer</td>
<td>Y</td>
<td>OBLPN’s facility.</td>
</tr>
<tr>
<td>company_id</td>
<td>Integer</td>
<td>Y</td>
<td>OBLPN’s company.</td>
</tr>
<tr>
<td>container_nbr</td>
<td>String</td>
<td>Y</td>
<td>OBLPN’s container number.</td>
</tr>
<tr>
<td>curr_location_id</td>
<td>Integer</td>
<td>N</td>
<td>OBLPN’s location.</td>
</tr>
<tr>
<td>lpn_type_id</td>
<td>Integer</td>
<td>N</td>
<td>Associated LPN Type.</td>
</tr>
<tr>
<td>length</td>
<td>Numeric</td>
<td>N</td>
<td>OBLPN’s length dimension.</td>
</tr>
</tbody>
</table>
width | Numeric | N | OBLPN’s width dimension.
height | Numeric | N | OBLPN’s height dimension.

- If providing `lpn_type_id` - `length`, `width`, and `height` are not valid.

Example Request Body:

```
"fields": {

  "oblpn": {

    "facility_id": 1,
    "company_id": 1,
    "container_nbr": "OBLPN-1",
    "lpn_type_id": 5
  }

}
```

Allocation Parameters Data

Allocation data is defined in the `parameters` section of the request in the `allocations` key. The data is a list of objects, each linking one sales order detail to one IBLPN for the given inventory and quantity. An order detail or IBLPN may be referenced across multiple allocation definitions within the same request. Each of the following allocation scenarios is supported:

- Single order detail from single IBLPN.
- Single order detail from multiple IBLPNs.
- Multiple order details from single IBLPN.
- Multiple order details from multiple IBLPNs.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocations</td>
<td>order_nbr</td>
<td>String</td>
<td>Y</td>
<td>Sales order identifier.</td>
</tr>
<tr>
<td>allocations</td>
<td>iblpn_nbr</td>
<td>String</td>
<td>Y</td>
<td>IBLPN identifier.</td>
</tr>
<tr>
<td>allocations</td>
<td>qty</td>
<td>Numeric</td>
<td>Y</td>
<td>Non-zero quantity to allocate.</td>
</tr>
<tr>
<td>allocations</td>
<td>order_dtl</td>
<td>Object</td>
<td>Y</td>
<td>Nested object identifying the sales order detail.</td>
</tr>
</tbody>
</table>

- Sales order status must be less than “Packed”.
- IBLPN status must be “Received”, “Located”, or “Partially Allocated” and have the necessary available unallocated quantity.

The nested `order_dtl` object requires one of two definitions in order to identify the sales order detail.

Identify Sales Order Detail by Sequence Number

If the order detail’s unique sequence number is known to the user, this may be provided in the request and is the only piece of data necessary to identify the correct detail for the given sales order number.
### Identify Sales Order Detail by Attributes

The sales order detail may also be identified by its attributes. At least one of the following pieces of information is required. If more than one order detail is identified, an error will be returned. Additionally, this is a restrictive search in that any omitted data will not be treated as a wildcard.

- If no `batch_nbr` is provided, only match order detail(s) without a batch.
- If no `invn_attr_X` value is provided for A-O, it will be treated as blank.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_dtl</td>
<td>item_barcode</td>
<td>String</td>
<td>C</td>
<td>Item identifier.</td>
</tr>
<tr>
<td>order_dtl</td>
<td>item_alternate_code</td>
<td>String</td>
<td>C</td>
<td>Item identifier.</td>
</tr>
<tr>
<td>order_dtl</td>
<td>batch_nbr</td>
<td>String</td>
<td>N</td>
<td>Batch identifier.</td>
</tr>
<tr>
<td>order_dtl</td>
<td>invn_attr_X</td>
<td>String</td>
<td>N</td>
<td>Attributes A-O tied to the order detail.</td>
</tr>
</tbody>
</table>

### Example Request Body:

```
"parameters": {

    "allocations": [

    {
        "order_nbr": "ORDER-1",
        "order_dtl": {
            "seq_nbr": 1,
        },
        "iblpn_nbr": "IBLPN-1",
        "qty": 1
    }
    ]
}
```
"order_nbr": "ORDER-2",

"order_dtl": {
    "item_barcode": "ITEM2",
    "batch_nbr": "BATCH-1",
    "invn_attr_a": "A",
    "invn_attr_b": "B"
  },

  "iblpn_nbr": "IBLPN-2",

  "qty": 2
}

Additional Options Data

Functional request data in the `options` section:

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>pack_flg</td>
<td>Boolean</td>
<td>N</td>
<td>Pack the OBLPN? (Default = False)</td>
</tr>
</tbody>
</table>

- OBLPN will be routed regardless of the `pack_flg` value.
- If `pack_flg` = True:
  - OBLPN will be updated to “Packed” status.
  - The created allocations will be completed.
  - The sales order detail(s) will be updated.
  - OBLPN’s final weight and volume will be calculated.
  - Inventory history will be written.

Example Request Body:

"options": {
    "pack_flg": true
}

Full Request Body Example:

The following example would create a packed OBLPN allocated from two different IBLPNs for the same order.
{
    "fields": {
        "oblpn": {
            "facility_id": 1,
            "company_id": 1,
            "container_nbr": "OBLPN-1"
        }
    },
    "parameters": {
        "allocations": [
            {
                "order_nbr": "ORDER-1",
                "order_dtl": {
                    "seq_nbr": 1
                },
                "iblpn_nbr": "IBLPN-1",
                "qty": 2
            },
            {
                "order_nbr": "ORDER-1",
                "order_dtl": {
                    "item_barcode": "ITEM-1",
                    "batch_nbr": "BATCH-1",
                    "invn_attr_a": "A",
                    "invn_attr_o": "O"
                },
                "iblpn_nbr": "IBLPN-2",
                "qty": 5.52
            }
        ]
    }
}
Link oblpn with asset

POST .../wms/lgfapi/v10/entity/oblpn/{id}/link_asset

links asset (reusable tote) to oblpn.

Assumptions

- Only one OBLPN may be linked to one asset per request.
- OBLPN must be within user's eligible facilities/companies.

Request Body Data

The request body data utilizes the 3 categories in the following ways:

1. `parameters` – allows user to identify the specific oblpn
2. `options` – Additional functional data.

Parameters

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>container_nbr</td>
<td>String</td>
<td>Y</td>
<td>OBLPN to be linked. &quot;__in&quot; lookup is not supported</td>
</tr>
<tr>
<td>Parameters</td>
<td>facility_id</td>
<td>Integer</td>
<td></td>
<td>Container's facility.</td>
</tr>
<tr>
<td>Parameters</td>
<td>company_id</td>
<td>Integer</td>
<td></td>
<td>Container's company.</td>
</tr>
</tbody>
</table>

Example Request Body:

```json
{
    "parameters": {
        "facility_id": 1,
        "company_id": 1,
        "container_nbr": "OBLPN-1"
    }
}
```
Note: Both facility id and company id also support filtering on "code".

**Additional Options Data**

Functional request data in the `options` section:

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>asset_nbr</td>
<td>String</td>
<td>Y</td>
<td>Asset to be linked. May be created as part of this API.</td>
</tr>
<tr>
<td>options</td>
<td>asset_seal_nbr</td>
<td>String</td>
<td></td>
<td>Optionally tracked seal number.</td>
</tr>
<tr>
<td>Options</td>
<td>replace_container_nbr_with_asset_flg</td>
<td>boolean</td>
<td></td>
<td>Rename OBLPN to match asset upon linking?</td>
</tr>
<tr>
<td>options</td>
<td>validate_lpn_type_flg</td>
<td>boolean</td>
<td></td>
<td>Validate the LPN type of the OBLPN with the LPN type of the asset</td>
</tr>
</tbody>
</table>

```json
{
  "options": {
    "asset_nbr": "ASSET-01",
    "asset_seal_nbr": "SEAL-001",
    "replace_container_nbr_with_asset_flg": true,
    "validate_lpn_type_flg": true
  }
}
```
• If the Asset already exists in the system, then it will be made "In Use" status and update the Asset OBLPN field with the corresponding OBLPN, Destination field with the OBLPN destination of the linked OBLPN and Seal Nbr field with corresponding seal nbr passed in the API
• If Original OBLPN is renamed while interfacing (i.e. when "replace_container_nbr_with_asset" = true), system will update the following:
  o Populate OBLPN field with the Asset Nbr,
  o Destination field with the OBLPN destination
  o Seal Nbr field with corresponding seal nbr passed in the API
• OBLPN type in the Asset table will not get updated with the OBLPN type of the OBLPN
• If the Asset interfaced is new, then a new record is created in the Asset UI with the status "In Use" with corresponding OBLPN, Seal and destination.
• If the Original OBLPN is renamed with Asset nbr while interfacing (i.e. when "replace_container_nbr_with_asset" = true), system updates the OBLPN field with the Asset Nbr, Destination field with the Original OBLPN’s destination and Seal Nbr field with corresponding seal nbr passed in the interface
• If the OBLPN is already linked to an asset and another Asset Nbr is passed in the interface for linking with OBLPN, the original asset number needs to be updated back to status "In-warehouse" while the new asset number is updated back to status "In-use".
• In case if the OBLPN is already linked to an asset/seal and another seal nbr is passed in the API, then update the seal nbr field with the corresponding seal.
• If the Asset interfaced in the API is new to the system, then a new record is created in the Asset table
• The fields "asset_nbr" and "asset_seal_nbr" is updated with corresponding data in the oblpn.
• If Original OBLPN is replaced with Asset Nbr while interfacing (i.e. when "replace_container_nbr_with_asset" = true), system should update the Container table as mentioned below:
  o LPN Nbr is updated with the Asset Nbr
  o Asset Nbr and Asset Seal Nbr is updated with the corresponding value passed in the API
  o OBLPN Type field is not updated with the OBLPN type of the Asset
  o "Ref OBLPN Nbr" field is updated with original OBLPN Nbr
• The following Inventory History records are created:
  o IHT 57 - Asset Received – This record is not written if the Asset interfaced in the API is new to the system
  o IHT 31- OB Container Modified is written if the OBLPN is renamed with Asset Nbr while linking.

Pallet

Sort LPN

The Sort LPN API allows you to sort an LPN to a Pallet in a sort location mimicking what the RF Inbound Sorting process does. The RF modules include: RF Sort LPN, and RF Inbound Sort Location.

You can sort an LPN to a pallet in a sort location with the following POST request:

POST .../entity/pallet/sort_lpn/
The following table provides details about the Input Parameters/Filters used to identify the target pallet:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>integer</td>
<td>Facility context by id.</td>
<td></td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>string</td>
<td>Facility context by code.</td>
<td></td>
</tr>
<tr>
<td>company_id</td>
<td></td>
<td>integer</td>
<td>Company context by id.</td>
<td></td>
</tr>
<tr>
<td>company_id__code</td>
<td></td>
<td>string</td>
<td>Company context by code.</td>
<td></td>
</tr>
<tr>
<td>pallet_nbr</td>
<td>X</td>
<td>string</td>
<td>Target sort pallet.</td>
<td></td>
</tr>
</tbody>
</table>

- The pallet will be created if it doesn’t exist.
- The requesting user’s default facility/company context will be assumed if overrides are not provided.

### Functional Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_nbr</td>
<td>X</td>
<td>string</td>
<td></td>
<td>LPN being sorted to pallet.</td>
</tr>
<tr>
<td>sort_zone</td>
<td>X</td>
<td>string</td>
<td></td>
<td>Destination sort zone.</td>
</tr>
<tr>
<td>sort_locationBarcode</td>
<td>X</td>
<td>string</td>
<td></td>
<td>Destination sort location.</td>
</tr>
<tr>
<td>sort_to_inventory</td>
<td></td>
<td>string</td>
<td>&quot;pallet-call-directed-putaway&quot;</td>
<td>Sort method.</td>
</tr>
<tr>
<td>allow_received_status_flg</td>
<td></td>
<td>boolean</td>
<td>False</td>
<td>Allow sorting of IBLPN in Received status.</td>
</tr>
<tr>
<td>allow_picked_status_flg</td>
<td></td>
<td>boolean</td>
<td>False</td>
<td>Allow sorting of OBLPNs in Picked status.</td>
</tr>
</tbody>
</table>

- Default valid LPN statuses:
  - Located
  - Allocated
  - Packed

The following is an example body for Sort LPN to Pallet:

```json
{
  "parameters": {
    "facility_id": 1,
    "company_id__code": "FOO",
    "pallet_nbr": "PALLET001"
  }
}
```
"options": {
    "container_nbr": "LPN001",
    "sort_zone": "ZONE01",
    "sort_location_barcode": "BRCD001",
    "sort_to_inventory": "pallet-call-directed-putaway"
}

Response Status

- 204 - No content
  - Operation successfully completed.
- 400 - Validation error
- 500 - Internal server error

Sort LPN/Close Pallet

The Sort LPN/Close Pallet API is used as part of the inbound sorting process which groups LPNs to pallets in sort locations. This API mimics the RF IB Sort LPN module which calls the Sort LPN Close IB Pallet back end entry point with parameters.

You can Sort LPNs and Close Pallet with the following POST requests:

POST .../entity/pallet/close_inbound_sorting/
POST .../entity/pallet/{id}/close_inbound_sorting/

The following table provides details about the Input Parameters/Filters used to identify the target pallet:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td>X</td>
<td>integer</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>string</td>
<td></td>
<td>Facility context by code.</td>
</tr>
<tr>
<td>company_id</td>
<td></td>
<td>integer</td>
<td></td>
<td>Company context by id.</td>
</tr>
<tr>
<td>company_id__code</td>
<td></td>
<td>string</td>
<td></td>
<td>Company context by code.</td>
</tr>
<tr>
<td>pallet_nbr</td>
<td>X</td>
<td>string</td>
<td></td>
<td>Target sort pallet.</td>
</tr>
</tbody>
</table>

- The pallet will be created if it doesn’t exist.
- The requesting user's default facility/company context will be assumed if overrides are not provided.
The following table details the functional options:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_replen_task_flg</td>
<td>boolean</td>
<td>True</td>
<td></td>
<td>Generate a replenishment task on close?</td>
</tr>
<tr>
<td>task_type_description</td>
<td>string</td>
<td></td>
<td></td>
<td>Required type description for generated replen task. Valid when create_replen_task_flg = True.</td>
</tr>
</tbody>
</table>

Default valid LPN statuses:

- Located
- Allocated
- Packed

The following is an example body for Create Replenishment Task Flag:

```json
{
   "parameters": {
      "facility_id": 1,
      "company_id__code": "FOO",
      "pallet_nbr": "PALLET001"
   },
   "options": {
      "create_replen_task_flg": true,
      "task_type_description": "My Task Type"
   }
}
```

**Response**

**Response Status:**

- 204 - No content
  - Operation successfully completed.
- 400 - Validation error
- 500 - Internal server error
Replenishment

Replenish to Active

The replenish_to_active API allows you to complete an open replenishment task for an active location.

You can replenish to active with the following POST request:

POST .../lgfapi/v10/replenishment/replenish_to_active/

Parameters

The following table provides details about the Input Parameters/Filters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td></td>
<td>integer</td>
<td></td>
<td>Facility context by id.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td></td>
<td>string</td>
<td></td>
<td>Facility context by code.</td>
</tr>
<tr>
<td>company_id</td>
<td></td>
<td>integer</td>
<td></td>
<td>Company context by id.</td>
</tr>
<tr>
<td>company_id__code</td>
<td></td>
<td>string</td>
<td></td>
<td>Company context by code.</td>
</tr>
</tbody>
</table>

- Used if the replenishment is in a context other than the requesting user’s default.
- The requesting user’s default facility/company context will be assumed if values are not provided.
- Either “facility_id” or “facility_id__code” may be used, but not both.
- Either “company_id” or “company_id__code” may be used, but not both.

The following table details the functional options:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>C</td>
<td>integer</td>
<td></td>
<td>&quot;id&quot; of task to be completed.</td>
</tr>
<tr>
<td>task_id__task_nbr</td>
<td></td>
<td>string</td>
<td></td>
<td>Business key for task to be completed.</td>
</tr>
<tr>
<td>replen_location_id</td>
<td>C</td>
<td>integer</td>
<td></td>
<td>&quot;id&quot; of active location to be replenished.</td>
</tr>
<tr>
<td>replen_location_id__barcode</td>
<td></td>
<td>string</td>
<td></td>
<td>Barcode of active location to be replenished.</td>
</tr>
<tr>
<td>qty</td>
<td></td>
<td>decimal</td>
<td>Allocation Qty</td>
<td>Quantity to replenish.</td>
</tr>
</tbody>
</table>

- Either "task_id" or "task_id__task_nbr" is required.
- Either "replen_location_id" or "replen_location_id__barcode" is required.
- If 'qty' is not provided, the full allocation quantity of the associated allocation will be used.
  - If 'qty' is provided, it must be greater than 0.
The following is an example body for Replenish Location ID Barcode:

```json
{
    "facility_id": 1,
    "company_id__code": "COMPANY",
    "task_id": 1,
    "replen_location_id__barcode": "LOCN1"
}
```

### Sales Order Header

#### Get IBLPN(s)

**GET */wms/lgfapi/v10/entity/order_hdr/{id}/iblps/**

Returns a paginated representation of all IBLPN(s) allocated to the sales order.

#### Get OBLPN(s)

**GET */wms/lgfapi/v10/entity/order_hdr/{id}/oblps/**

Returns a paginated representation of all OBLPN(s) allocated to the sales order.

### Bulk Lock

**POST */wms/lgfapi/v10/entity/order_hdr/bulk_lock/**

This operation is used to apply, and optionally create, an order lock to one or more orders.

The number of orders that can be modified by this operation in a single requests is configured by the value of the requesting user’s “Rows per Page” attribute.

The “parameters” section of the request body is required in addition to the “options” section outlined below. One or more parameters are used to determine the order(s) for which the operation will be applied. The allowed filter parameters are:

- 'id'
- 'order_nbr'
- 'facility_id'
- 'company_id'
- 'erp_source_hdr_ref'
- 'erp_source_system_ref'
- 'orderdtl__erp_source_line_ref'
- 'orderdtl__erp_source_shipment_ref'
- 'orderdtl__ship_request_line'
### Options

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>lock_code</td>
<td>String</td>
<td>X</td>
<td></td>
<td>Order lock to be applied.</td>
</tr>
<tr>
<td>options</td>
<td>lock_description</td>
<td>String</td>
<td></td>
<td>Value of lock_code</td>
<td>Description of order lock. Only used when creating a new order lock.</td>
</tr>
<tr>
<td>options</td>
<td>comments</td>
<td>String</td>
<td></td>
<td>&quot;&quot;</td>
<td>Additional info for the order’s applied lock.</td>
</tr>
<tr>
<td>options</td>
<td>allowAllocate_flg</td>
<td>Boolean</td>
<td></td>
<td>False</td>
<td>Order lock attribute. Only used when creating a new order lock.</td>
</tr>
<tr>
<td>options</td>
<td>autocreate_lock_flg</td>
<td>Boolean</td>
<td></td>
<td>False</td>
<td>When true, the order lock will be created in addition to be applied, if it does not already exist.</td>
</tr>
<tr>
<td>options</td>
<td>commit_frequency</td>
<td>Integer</td>
<td></td>
<td>0</td>
<td>0 = Roll back on first error. 1 = Commit per object.</td>
</tr>
</tbody>
</table>

#### Bulk Unlock

**POST .../wms/lgfapi/v10/entity/order_hdr/bulk_unlock/**

This operation is used to remove an order lock from one or more orders.

The “parameters” section of the request body is required in addition to the “options” section outlined below. One or more parameters are used to determine the order(s) for which the operation will be applied. The allowed filter parameters are:

- 'id'
- 'order_nbr'
- 'facility_id'
- 'company_id'
- 'erp_source_hdr_ref'
- 'erp_source_system_ref'
- 'orderdtl__erp_source_line_ref'
- 'orderdtl__erp_source_shipment_ref'
- 'orderdtl__ship_request_line'

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>lock_code</td>
<td>String</td>
<td>X</td>
<td></td>
<td>Order lock to be removed.</td>
</tr>
<tr>
<td>options</td>
<td>commit_frequency</td>
<td>Integer</td>
<td></td>
<td>0</td>
<td>0 = Roll back on first error. 1 = Commit per object.</td>
</tr>
</tbody>
</table>

### Print

#### Print Shipping Label

**GET.../wms/lgfapi/v10/print/label/shipping/?label_designer_code=foo**

Returns the ZPL representation of the label
POST /wms/lgfapi/v10/print/label/shipping

Submits the label for printing

The “parameters” section of the request body is required in addition to the “options” section outlined below. One or more parameters are used to determine the order(s) for which the operation will be applied.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>GET Request</th>
<th>POST Request</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>label_designer_code</td>
<td>string</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Label designer template to be printed</td>
</tr>
<tr>
<td>options</td>
<td>printer_name</td>
<td>string</td>
<td></td>
<td>X</td>
<td></td>
<td>Default's to cwuser.default_label_printer.</td>
</tr>
<tr>
<td>options</td>
<td>label_count</td>
<td>integer</td>
<td></td>
<td>X</td>
<td></td>
<td>Number of labels to print. Must be greater than 0. Default = 1.</td>
</tr>
</tbody>
</table>

API Filters

- Functions like a bulk operation for identifying one or more IBLPN(s) to be printed:
  - id
    - Including "in" lookup
  - facility_id
  - company_id
  - container_nbr
    - Including "in" lookup

Example Query String for GET

GET.../wms/lgfapi/v10/print/label/shipping/?label_designer_code=foo&facility_id__code=FAC1&company_id__code=COM1&container_nbr=IBLPN1

Example Request Body for POST

```json
{
  "parameters": {
    "facility_id__code": "FAC1",
    "company_id__code": "COM1",
    "container_nbr": "IBLPN1"
  },
  "options": {
    "label_designer_code": "label_1",
  }
}
```
"printer_name": "PRINTER1",
"label_count": 1
}

Response Body Data

On success, a 200 - OK status is returned.

The standardized bulk response body is returned. This will have aggregate information for all IBLPN(s) processed as well as the counts and any details.

For a GET request, the ZPL data will be base64 encoded in the "data" section.

{
   "record_count": 2,
   "success_count": 1,
   "failure_count": 1,
   "data": {
      "IBLPN_1": "VGhpcyBpcyBaUEwgY29kZQ=="
   },
   "details": {
      "IBLPN_2": "Some error message."
   }
}

Print LPN Label

GET.../wms/lgfapi/v10/print/label/ib_container/?label_designer_code=foo

Returns the ZPL representation of the label.

POST .../wms/lgfapi/v10/print/label/ib_container

Submits the label for printing. The “parameters” section of the request body is required in addition to the “options” section outlined below. One or more parameters are used to determine the order(s) for which the operation will be applied.
<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>GET Request</th>
<th>POST Request</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>label_designer_code</td>
<td>string</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Label designer template to be printed</td>
</tr>
<tr>
<td>options</td>
<td>printer_name</td>
<td>string</td>
<td></td>
<td></td>
<td>X</td>
<td>Default's to cwuser.default_label_printer.</td>
</tr>
<tr>
<td>options</td>
<td>label_count</td>
<td>integer</td>
<td></td>
<td></td>
<td>X</td>
<td>Number of labels to print. Must be greater than 0. Default = 1.</td>
</tr>
</tbody>
</table>

**API Filters**

- Functions like a bulk operation for identifying one or more IBLPN(s) to be printed:
  - id
    - Including "in" lookup
  - facility_id
  - company_id
  - container_nbr
    - Including "in" lookup

**Example Query String for GET**

```
GET.../wms/lgfapi/v10/print/label/ib_container/?label_designer_code=foo&facility_id__c
ode=FAC1&company_id__code=COM1&container_nbr=LPN1
```

**Example Request Body for POST**

```json
{
    "parameters": {
        "facility_id__code": "FAC1",
        "company_id__code": "COM1",
        "container_nbr": "OBLPN1"
    },
    "options": {
        "label_designer_code": "label_1",
        "printer_name": "PRINTER1"
    }
}
```
Response Body Data

On success, a 200 - OK status is returned

For a GET request, the ZPL data will be base64 encoded in the "data" section.

```
{
    "record_count": 2,
    "success_count": 1,
    "failure_count": 1,
    "data": {
        "IBLPN_1": "VGhpcyBpcyBaUEwgY29kZQ=="
    },
    "details": {
        "IBLPN_2": "Some error message."
    }
}
```

Print Pallet Label

GET.../wms/lgfapi/v10/print/label/pallet/?label_designer_code=foo

Returns the ZPL representation of the label

POST .../wms/lgfapi/v10/print/label/pallet

Submits the label for printing

The "parameters" section of the request body is required in addition to the "options" section outlined below. One or more parameters are used to determine the order(s) for which the operation will be applied.

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>GET Request</th>
<th>POST Request</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>label_designer_code</td>
<td>string</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Label designer template to be printed</td>
</tr>
<tr>
<td>options</td>
<td>printer_name</td>
<td>string</td>
<td></td>
<td>X</td>
<td></td>
<td>Default's to cwuser.default_label_printer.</td>
</tr>
<tr>
<td>options</td>
<td>label_count</td>
<td>integer</td>
<td>X</td>
<td>Number of labels to print. Must be greater than 0. Default = 1.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

API Filters

- Functions like a bulk operation for identifying one or more IBLPN(s) to be printed:
  - id
    - Including "in" lookup
  - facility_id
  - company_id
  - container_nbr
    - Including "in" lookup

Example Query String for GET

```
GET.../wms/lgfapi/v10/print/label/pallet/?label_designer_code=foo&facility_id__code=FAC1&company_id__code=COM1&pallet_nbr=pallet1
```

Example Request Body for POST

```
{
    "parameters": {
        "facility_id__code": "FAC1",
        "company_id__code": "COM1",
        "pallet_nbr": "pallet1"
    },
    "options": {
        "label_designer_code": "label_1",
        "printer_name": "PRINTER1"
    }
}
```

Response Body Data

On success, a 200 - OK status is returned

For a GET request, the ZPL data will be base64 encoded in the "data" section.

```
{
    "success_count": 1,
```
Report

Custom Inventory Summary

Allows you to execute the custom inventory summary report for only a single item per request. This request returns the result set as a file attached to the response.

If output format is pipe-delimited, use the following:

GET.../report/custom_inventory_summary/?facility_id__code=FAC1&company_id__code=COM1&item_code=ITEM1

If output format is XML use the following:

GET.../report/custom_inventory_summary.xml?item_code=<item_code>&company_id=<company_id>&facility_id=<facility_id>

The “parameters” section of the request body is required in

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>facility_id</td>
<td>integer</td>
<td>C</td>
<td></td>
<td>Required facility context.</td>
</tr>
<tr>
<td>facility_id__code</td>
<td>string</td>
<td>C</td>
<td></td>
<td>Required facility context.</td>
</tr>
<tr>
<td>company_id</td>
<td>integer</td>
<td>C</td>
<td></td>
<td>Required company context.</td>
</tr>
<tr>
<td>company_id__code</td>
<td>string</td>
<td>C</td>
<td></td>
<td>Required company context.</td>
</tr>
<tr>
<td>item_code</td>
<td>string</td>
<td>X</td>
<td></td>
<td>Specific item for the report.</td>
</tr>
<tr>
<td>write_header_line_flg</td>
<td>boolean</td>
<td></td>
<td>False</td>
<td>Include the header line with field names?</td>
</tr>
</tbody>
</table>

- Either `facility_id` or `facility_id__code` is required
- Either `company_id` or `company_id__code` is required

Company Parameter

POST .../entity/company_parm

This operation is used to add single or multiple company parameters.

If you have a new facility and you want to copy the same Company Parameters from your current facility, you can first GET the list by querying the company_parm entity, then POST the applicable data to this operation for the target facility.
Example body request

{
    "fields": {
        "company_id": 1,
        "parm_key": "TEST_PARM_001",
        "parm_value": "test"
    }
}

**Facility Parameter**

POST .../entity/facility_parm

This operation is used to add single or multiple facility parameters.

If you have a new facility and you want to copy the same facility parameters from your current facility, you can first GET the list by querying the facility_parm entity, then POST the applicable data to this operation for the target facility.

Example body request

{
    "fields": {
        "facility_id": 1,
        "prog_key": "FACILITY_PARM",
        "parm_key": "TEST_PARM_001",
        "parm_value": "test"
    }
}

**Location Size Type**

POST .../entity/location_size_type

This operation is used to add single or multiple location size types.
If you have a new facility and you want to copy the same location size type from your current facility, you can first GET the list by querying the location_size_type entity, then POST the applicable data to this operation for the target facility.

Example body request

```
{
  "fields": {
    "company_id": 1,
    "size_type": "TEST_SIZE_001",
    "description": "Test Size 001"
  }
}
```

**Putaway Priority**

This operation allows you to determine the order in which Putaway Types are triggered for putaway.

POST .../entity/putaway_priority

If you have a new facility and you want to copy the same Putaway Priority rules from your current facility, you can first GET the list by querying the putaway_priority entity, then POST the applicable data to this operation for the target facility.

Example body request:

```
{
  "fields": {
    "facility_id": 1,
    "priority": 1,
    "putaway_type_id": 256860,
    "putaway_method_id": 1,
    "putaway_search_mode_id": 0,
    "locn_type_id": 3,
    "locn_size_type_id": 0,
    "replenishment_zone_id": 35995,
  }
}
```
"consider_fefo_flg": false,
"radius": 1,
"radial_increment": 1
}
}

Putaway Type

POST .../entity/putaway_type

This operation is used to add single or multiple putaway type.

If you have a new facility and you want to copy the same putaway type from your current facility, you can first GET the list by querying the putaway_type entity, then POST the applicable data to this operation for the target facility.

Example body request:

{
    "fields": {
        "company_id": 1,
        "pa_type": "TEST_PA_001",
        "description": "Test PA 001",
        "pallet_position_required_flg": false,
        "depalletize_on_putaway_flg": false
    }
}

Putaway Type Cal Rule

POST .../entity/putaway_type_calc_rule

This operation is used to add single or multiple putaway type cal rules.

If you have a new facility and you want to copy the same putaway type cal rule from your current facility, you can first GET the list by querying the putaway_type_cal entity, then POST the applicable data to this operation for the target facility.
Example body request:

```json
{
    "fields": {
        "facility_id": 1,
        "company_id": 1,
        "description": "TEST-001",
        "priority": 1,
        "final_putaway_type_id": 256860,
        "sql_selection_id": 76886,
        "enabled_flg": true
    }
}
```

**Replenishment Zone**

POST `.../entity/replenishment_zone`

This operation is used to add one or more replenishment zones.

If you have a new facility and you want to copy the same replenishment zones from your current facility, you can first GET the list by querying the replenishment_zone entity, then POST the applicable data to this operation for the target facility.

Example body request

```json
{
    "fields": {
        "facility_id": 1,
        "code": "TEST_RZ_001",
        "description": "Test RZ 001"
    }
}
```
SQL Selection (Rule Tree)

POST .../entity/sql_selection

This entity is unique in that the API will allow the user to create the entire rule tree in a single request instead of needing the create and link each parent/child object individually (it can still be done this way if the user chooses to do so). This is accomplished using the `children` list field. This is an abstract field that does not exist on the object itself, but rather defines the `parent_id` link, which will be handled by the API automatically.

To illustrate a complex example, the following request body could be used to create this rule structure as seen from the UI:

Example body request

```
{
    "fields": {
        "facility_id": 1,
        "sql_operator_id": 2,
        "children": [
            {
                "column_name_id": 107,
                "sql_operator_id": 5,
                "column_value": "B"
            },
            {
                "sql_operator_id": 1,
                "children": [
                    {
                        "column_name_id": 1379,
                        "sql_operator_id": 7,
                        "column_value": "B"
                    }
                ]
            }
        ]
    }
}
```
Item Image

Currently the full representation of item GET does not include the item image ('image_data') since that can be large. However if a request specifies the `fields` query string parameter and the 'image_data' field is specified, we will return the field and value.

This will return the id and image data for one or more items.

GET .../entity/item/?fields=id,image_data

This will return the id and image data for a specific item.

GET .../entity/item/{id}/?fields=id,image_data

The ‘fields’ parameter may still be combined with other filters per normal functionality:

GET .../entity/item/?fields=id,image_data&barcode=ITEM123&...