Oracle® Warehouse Management Cloud

Rest API Guide

Update 18C

Part No. F10011-01

September 2018

ORACLE

Copyright Notice

Oracle® Warehouse Management Cloud Rest API Guide, Update 18C

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Send Us Your Comments

Oracle® Warehouse Management Cloud Rest API Guide, Update 18C

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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

Date	Document Revision	Summary of Changes
07/2018	-01	Initial release.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

1. Overview

Oracle Warehouse Management (WMS) Cloud version 9.0.0 includes 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 version 9.0.0, 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 have been 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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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.

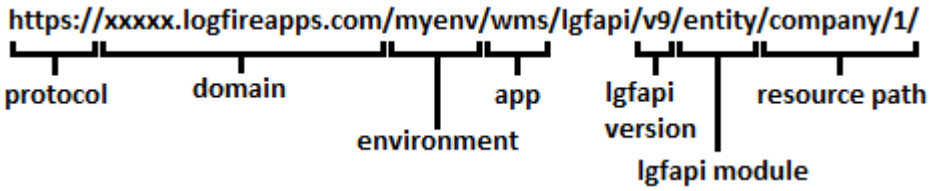
PATCH
Modify existing resource(s).

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:



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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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.

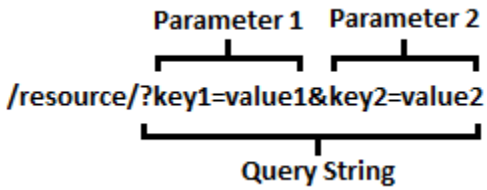
Data Input Methodology

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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



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-ACII 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 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. Lgfapi 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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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:

```
{
 "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 needing 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:

```
{
 "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.

```
<serial_nbr_list>
        <list-item>SN1</list-item>
```


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

<list-item>SN2</list-item>
</serial_nbr_list>

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

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:

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

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Status Code	Status Message	HTTP Method	Description
			and provide any returned error information.

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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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:

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

Example XML Error Response Body:

```
<?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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Crawl-able hyperlinks are provided to navigate between the pages.

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:

```
{
  "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 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>
```

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

…
</list-item>
</results>
</entity_name>

Mode: Sequenced
The sequenced mode (*…/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:

…/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 *(…/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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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/v9/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/v9/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 sting 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>abc1234</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>

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Boolean
Except for a few specific cases, all True/False Boolean field names end with "_flg".
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": "2018-01-3030T18:30:00"}}
XML: <field_ts>2018-01-3030T18:30:00</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>

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

- startwith/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

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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 © 2018, Oracle and/or its affiliates. All rights reserved.

- 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 lgfapi 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

 Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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:

…/lgfapi/v9/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:

…/lgfapi/v9/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:

…/lgfapi/v9/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:

…/lgfapi/v9/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:

…/lgfapi/v9/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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Note that *…/lgfapi/v9/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 crawl-able 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:

```
{
  …
  "company_type_id": {
    "id": 1,
    "key": "R",
    "url": "https://…/wms/lgfapi/v9/entity/company_type/1"
  },
  …
}
```

Or, if the desired format is XML:

```
<company>
  …
  <company_type_id>
    <id>1</id>
    <key>R</key>
    <url>https://.../wms/lgfapi/v9/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

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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:

```
{
  …
  "status_id": 10,
  …
}
```

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

GET https://.../wms/lgfapi/v9/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/v9/entity/order_hdr/123

```
{
  "id": 123,
  …
  "order_dtl_set": {
    "result_count": 2,
    "url": "https://.../wms/lgfapi/v9/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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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/v9/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/v9/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/v9/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/v9/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/v9/entity/company/?ordering=company_type_id,-code

Just like any other query string parameter, it may be chained with other parameters and filters.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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 down-steam 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

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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 sting 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/v9/entity/company/1

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

HEAD https://.../wms/lgfapi/v9/entity/company?code=ABC

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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" 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/v9/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

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

XML Example

```
<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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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:

```
{
  "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.

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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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/v9/entity/{entity_name}/{id}/{operation_name}/

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

…/wms/lgfapi/v9/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.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

4. Supported Entity Operations

Describe

GET …/wms/lgfapi/v9/entity/{entity_name}/describe/

The describe operation is unique in that it can 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.

Inventory

Link Serial Numbers
POST …/wms/lgfapi/v9/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.

Category	Parameter Name	Required	Description
options	serial_nbr_list	X	A list of serial number strings to be linked to the target inventory record.

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/v9/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/v9/entity/container/{id}/lock/
Apply one or more inventory locks to the target inbound or outbound container.

Category	Parameter Name	Required	Description
options	lock_code_list	X	A list of inventory lock codes to be applied to the target container.

Unlock Container
POST …/wms/lgfapi/v9/entity/container/{id}/unlock/
Remove one or more inventory locks to the target inbound or outbound container.

Category	Parameter Name	Required	Description
options	lock_code_list	X	A list of inventory lock codes to be removed from the target container.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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/v9/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.

Category	Parameter Name	Required	Description
options	reason_code	X	Reason code to be used for inventory history tracking.

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/v9/entity/oblpn/{id}/mark_delivered/
Updates a Shipped OBLPN to Delivered status and writes container delivered inventory history.

Order Header

Get IBLPN(s)
GET …/wms/lgfapi/v9/entity/order_hdr/{id}/iblpns/
Returns a paginated representation of all IBLPN(s) allocated to the sales order.

GET OBLPN(s)
GET …/wms/lgfapi/v9/entity/order_hdr/{id}/oblpns/
Returns a paginated representation of all OBLPN(s) allocated to the sales order.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.