Entity API

Author Kodmyran AB

Introduction

The entity API provides access to all objects and functionality within Kodmyran Commerce. This part of the API is also used by the graphical administration tools but can be very hard to work with without a very good understanding of entities, their relationships etc. This API is based upon a concept of entities where you can create, update and delete objects (CRUD). Unless you really need the full power of this API we recommend using the integration API instead.

The API can also be browsed from your favorite web browser as most links are hyperlinks. E.g. using Chrome and installing one of the many JSON/REST plugins (JSONview) will give you a JSON interface with clickable links where you can move forward, backwards etc. through search answers.

The entity API is available to all customers with a premium/complete or higher subscription free of charge for up to 20 calls/minute. To increase the limit beyond 20 calls/minute a separate agreement/license is required.

Transactional rules

A single call to the entity API is fully transactional, a failure during processing will result in a database rollback of all data changed during that call. Creating or updating entities may result in one or more audit logs being written.

Security

The user must first authenticate as described in the overview using an API key.

Once the key has been validated, and the domain name checked the user associated with the API key is checked for the proper permissions. Initially the user must possess the “Remote call: Read” and/or the “Remote call: Write” permissions (depending upon the HTTP request verb).

Once the user passes this check the role that they possess must also contain permissions to access the requested entity type. The permissions granted to that role for that object type dictates the users access. The available permissions are:

- bSelect, the ability to query data from this type of object. Without this permission, all read/GET requests are denied.
- bInsert, the ability to create new entries. Without this permission, all HTTP POST requests are denied.
- bUpdate, the ability to update an existing object. Without this permission, all HTTP PUT requests are denied.
- bDelete, the ability to delete an object. Without this permission, all HTTP DELETE requests are denied.

Callbacks/Webhooks

To avoid having to poll the API for changes the system supports callbacks (webhooks). A webhook is a call from Kodmyran Commerce to an external party whenever a pre-determined condition occurs within Kodmyran Commerce. They are typically triggered as the result of a new object or changes to an existing object.

Whenever an object is declared *dirty* the entity type is compared to the pre-registered webhooks, if a match is found a call is dispatched to the webhook URL. The webhook itself is a HTTP POST operation containing a header block and a block optionally containing the entire business object (determined during webhook registration).

Please observe that the outbound webhook is subject to a timeout of roughly 5 seconds after connecting. That means the amount of work performed directly in the webhook receiver should be limited. Typically, the receiver should add the work to an internal queue for later processing and respond with a HTTP 200 message, or in the case of quick updates complete the work and respond with a 200 message. Responding with a non 2xx message will indicate to Kodmyran Commerce that the message could not be processed and the system will attempt re-delivery at a later time. If the message has been re-delivered too many times the webhook will be put into a paused state and no more webhooks will be sent until manually re-enabled. Depending upon the length of time between the initial send and the pause, messages may be lost in which case a full resynchronization may be required.

The programmer building an integration should never ever trust the delivery of a webhook, it is provided as a means to quickly synchronize two systems - but the webhook may be lost or delayed. The webhook is considered a hint and not a synchronization primitive.

To ensure that webhooks do not create loops they are only sent to webhooks registered by users other than the one that caused the initial triggering. Hence it is very important that different integrations do not re-use the same API key.

The system will never make multiple simultaneous calls for the same webhook, they will be executed in a single-threaded fashion one by one. For the integration API a single webhook call may contain multiple objects however, in that case they should always be executed in the order received.

Base endpoint

The full API is available at the endpoint:

```
https://testaccount.shop4sale.se/admin/api/entity
```

Accessing the URL above directly with your browser will give you a list of entities the system supports, however it does not mean that you can always access that type of entity. The user permissions have not been applied at this stage.

Best practice

To simplify troubleshooting we recommend that you set the User-Agent HTTP header in your request to something that uniquely describes your integration. Keep the text short and to the point and use english language. Do not include special characters like umlaut and avoid very long descriptions.

Do not include fields unless they are required or have been modified, a fields value will be retained within Kodmyran Commerce. The only exception to this rule applies to sub-entitites as described further down in this document.

Never include the object ID in a top-level object upon update or creation (e.g. omit the 'id' field at top-level)

Many fields within the builtin objects are read-only, read-only upon update or virtual (derived) - these fields will not allow write operations and the call will fail.

Any return code other than HTTP code 200-299 is an invalid request.

Searching

Each type of entity can then be appended to the URL to make a search within that space, e.g. to get a list of all users you can access:

`https://testaccount.shop4sale.se/admin/api/entity/user`

The response is divided into two sections, a meta section and an objects section. The meta section will contain the call status (should be “OK”), the number of objects in total, the current offset and the count of objects on this page.

The objects section will contain a list of all the requested objects, up to the query limit. If not set the query limit defaults to a maximum of 20 entries.

You can pass parameters to this URL to control the response you receive, parameters starting with an underscore are special and are treated as meta parameters. The meta parameters are identical regardless of the type of entity. Parameters without an initial underscore are query parameters and can, optionally, contain a leading command for larger than, less than and similar operations. The actual parameters depend upon the entity type and can be seen in the Swagger interface definition.

Field	Description
_limit	Controls the number of entries returned per page. Defaults to 20, maximum 100.
_order	Controls the ordering of the results. Specify the field name as the value. By prefixing the value wit
_offset	The starting offset of the search. Defaults to 0.
_fields	A list of fields to include in the search response in addition to the entity ID and URL which are alw
_tbldef	A special use for the GUI tools that can invoke a pre-setup table definition. This is not intended fo

The supported prefixes are listed below (for query parameters only, not valid for meta parameters). If no prefix has been specified the comparison is assumed to be that of equality.

Prefix	Description
>	Larger than, only makes sense for numerical and date/timestamp fields.
>=	Larger than or equal, only makes sense for numerical and date/timestamp fields.
<	Less than, only makes sense for numerical and date/timestamp fields.
<=	Less than or equal, only makes sense for numerical and date/timestamp fields.
-	Not
~	Like, apply an SQL style LIKE search. Do not include wildcard characters in the string. This makes

Please note that the same query parameter can occur multiple times with different prefixes, e.g. to search for entities within a specific time range.

Once you have made a search and isolated an entity for deeper inspection you can obtain every detail of that entity by appending the ID to the endpoint, the example below fetches the user with ID 123:

```
https://testaccount.shop4sale.se/admin/api/entity/user/123
```

The precise design of such a response depends upon the entity type, but these fields are always present:

Id	The entity ID
Generation	The entity generation, initially 1. Incremented by 1 for each commit that changes the object.
Created	The timestamp for when the entity was created in ISO 8601 format
Changed	The timestamp for when the entity was last changed in ISO 8601 format

A GET to this endpoint will return the object in JSON form.

A POST to this endpoint will return an error, as it contains an entity ID. To make a POST request you will have to use the end-point level directly above (e.g. `/admin/api/entity/user` in this example).

A PUT to this endpoint will update the object and return a 200 response on success.

Search and retrieve more than the default fields

A common operation is to search for all entities matching some criteria and then extracting a certain number of fields, in order to avoid first having to make a search request and then fetch each individual item separately the API provides a _fields parameter. Using the fields parameter the client can request specific fields straight away in the search response.

Say for example we want to search for all users with a firstname='Erik' and retrieve their last name. Instead of first finding all entity IDs matching the search and retrieving their last name one by one, we can do it all in one operation:

```
GET https://testaccount.shop4sale.se/admin/api/entity/user?firstname=Erik&_fields=lastname
```

```
{
    "meta": {
        "status":"OK",
        "count":20,
        "offset":0,
        "totalcount":189,
        "nextpage":"https://testaccount.shop4sale.se/admin/api/entity/user/?firstname=Erik&_
    },
    "objects": [
        {
            "id":680,
            "url":"https://testaccount.shop4sale.se/admin/api/entity/user/680",
            "lastname": "Nordgren"
        },
        {
            "id":707,
            "url":"https://testaccount.shop4sale.se/admin/api/entity/user/707",
            "lastname": "Andersson"
        },
        {
            "id":746,
            "url":"https://testaccount.shop4sale.se/admin/api/entity/user/746",
            "lastname": "Svensson"}
        },
        ...
    ]
}
```

This avoids a performance bottleneck refered to as row-at-a-time, and also reduces the number of calls you need to make increasing performance significantly.

If you need to include more than one additional field in the responses you can provide a comma separated list to the _fields input parameter.

Sub-entities

Sub-entities are normal entities that are linked to another (parent) entity. Typical examples are e.g. orderrows for an order. Many types of entities in Kodmyran Commerce contain these linked sub-entities in their response, when they do they will follow a similar structure to regular entities and will include their own created, changed, id and generation fields.

When you submit an existing entity for change/update and include a list of rows these will update/replace the current rows. A row ID not included in the update will be removed. Hence if you e.g. want to update an order and add a new orderrow you must also include all existing orderrows in the call or the old rows will be removed.

Note that e.g. orderrows can be accessed either as a sub-entity to an order, or directly as an orderrow through the API.

When accessed as a sub-entity the permissions required are those of the parent entity. Hence in the order/orderrow example only the order entity needs to have write permissions granted even when changing an orderrow.

Create or update an object

To create an object you must use the HTTP VERB 'POST', to modify an existing object you instead use the HTTP verb 'PUT'. Unlike some other REST APIs we do not use the PATCH verb (PUT provides identical functionality). For PUT and POST operations the system requires that you include a meta header block describing base information about the request, such as the language in use, or if it is intended for a particular multishop. You then provide the actual object wrapped within a request block.

When creating an object you issue your request to the base level for that entity, e.g.:

```
https://testaccount.shop4sale.se/admin/api/entity/user
```

When updating an object you issue your request to the URL for that entity, e.g. for user 123:

```
https://testaccount.shop4sale.se/admin/api/entity/user/123
```

An example request to create a new user may look similar to this:

```
POST https://testaccount.shop4sale.se/admin/api/entity/user
X-API-Key: 12345667890ABCDEF
Content-Type: application/json
User-Agent: The Fred Flintstone Connector

{
    "meta": {
        "storeid": 1,
        "language": "sv"
    },
    "request": {
        "usertype": "user",
        "company": "Stora Bolaget AB",
        "pno": "556767-4444",
        "vatno": "SE556767444401",
        "firstname": "Big",
        "lastname": "Boss",
        "address": "Storgatan 1",
        "extaddress": "Box 123",
        "countryid": 1,
        "zipcode": "12345",
        "city": "Lillby",
        "email": "big.boss@storabolaget.se",
        "telephone": "070-4112233",
        "languageid": 1
    }
}
```

The system will respond according to the example at the end of this document. A request to update the same user will have the same type of blocks as described above, but the URL to PUT to is different.

Deletions

When you call the DELETE method of an entity that entity will undergo a reference check. If references are found the deletion will be denied. This matches the behavior of the low-level framework.

Responses

You can find example responses in this section. If an error occurs the system will respond with a 3xx, 4xx or 5xx message. A 4xx message indicates that the request was denied, either because you lack permissions or that the data does not conform with the entity model. A 5xx message indicates a technical error and generally means there is a more severe error.

Creating a new entity

The API will respond with a HTTP 201 message and a JSON object.

```
{
    "callStatus": "OK",
    "entityid": 123,
    "url": "https://testaccount.shop4sale.se/admin/api/entity/user/123",
    "created": "2017-05-16T20:56:32+01:00",
    "changed": "2017-05-16T20:56:32+01:00",
    "generation": 1
}
```

Update an existing entity

The API will respond with a HTTP 200 OK message and a JSON object

```
{
    "callStatus": "OK",
    "changed": "2017-05-16T20:57:38+01:00",
    "generation": 2
}
```

Getting a full response

For PUT and POST operations you will by default get a response with various metadata such as timestamps, the generation etc but not the complete object. In most cases this is more than enough, however if your code requires the entire object you can request a full response by setting the fullresponse meta parameter to the value true.

The entire object will then be returned under the object key in the response.

Inbound request:

```
{
    "meta": {
        "fullresponse": true
    },
    "request": {
        ...
    }
}
```

Response from API:

```
{
    "callStatus": "OK",
    "entityid": 123,
    "url": "https://testaccount.shop4sale.se/admin/api/entity/user/123",
    "created": "2017-05-16T20:56:32+01:00",
    "changed": "2017-05-16T20:56:32+01:00",
    "generation": 1
    "object": {
        ...
    }
}
```