Integration API

Author Kodmyran AB

Introduction

The *integration API* provides a subset of the entity API. Unlike the entity API the concept here is for a consumer to read what is essentially a transaction log, apply the changes to their local system and when required push changes back into Kodmyran Commerce.

This is the recommended approach for most integrations towards webshops and similar as it provides more consistency checks and will automatically resolve various things for you.

The term *consumer* is used to signify the endpoint of an integration that reads and consumes the events generated by Kodmyran Commerce.

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.

When setting up the initial synchronization view you specifiy a webhook URL that will be called as an HTTP POST operation when events are available. In your webhook handler you need to call the API to retrieve the actual journal entries. When retrieving the journal you specify a starting point (the last event received). Kodmyran Commerce will automatically and regularly flush old events from the journal.

The system will never make multiple simultaneous calls for the same webhook, they will be executed in a single-threaded fashion one by one.

Synchronization views

Kodmyran Commerce has support for a wide variety of synchronization patterns, but for the integration API a synchronization view is the only one to use.

A synchronization view is a special type of view used by Kodmyran Commerce to know which objects and changes that belong to which integration. If an entry exists in the view matching the newly updated object, information is added to the synchronization log. When requested, the synchronization log (journal) is sent in batches to the consumer which applies them in order. The consumer maintains a variable that identifies the last event applied, this value is presented in each request for new events and Kodmyran Commerce retrieves events newer than this in the response.

The starting point is to be treated as an opaque textual value, do not treat is an integer as the format is defined as a string less than 20 characters. Currently it is a monothonic ever-increasing integer but this is not guaranteed for future versions.

Endpoint

The integration API is available at the endpoint:

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

The integration endpoint is the main method for integrating downwards in the stack, towards external systems such as webshops.

It is designed to synchronize specific types of objects and in general simplify the integration as much as possible by providing a stream of events that the remote system can execute to stay in sync with Kodmyran Commerce. The events received must be executed in the order received.

The downstream system may also submit new orders, users and products into Kodmyran Commerce. Products and users can be updated after the initial submission but orders are blocked from updating once submitted. Some fields on existing products are also blocked from updating but this is done silently without producing an error message.

The integration API can be considerad a simplified subset of the entity API, streamlined for simple integration.

Synchronization pattern

The synchronization pattern for the integration API is essentially a matter of processing a list of events, in order. Each event has a unique ID that uniquely identified the specific event.

Each integration has its own dedicated (virtual) synchronization view and can be at different positions in the log at any given time. The very first call you typically need to make is to create your dedicated synchronization view. During the view creation you will be provided with a view-ID/hash that all other calls in the integration API require, in addition to the API key. The value is provided for all other calls except view creation as a HTTP header.

Kodmyran Commerce will push a webhook to the consumer shortly after a event has occured, the consumer upon receipt of the callback can request the journal and start applying the changes in order.

After applying each individual event the consumer will update a persisted variable that indicates the last event applied. The event ID is to be considered a string of 20 characters or less, it needs to be stored to preserve the full value.

Each journal request will return a list of no more than 250 entries, the size of each batch is set in the initial request to register the view and can not be changed without removing and re-creating a syncview. For each requested journal segment a field is provided, *moredata*, that is a boolean to indicate whether more data is available or not. We recommend continuing requesting events as long as the moredata parameter is true or a certain number of maximum loops has been reached.

Please note that if multiple synchronization views are in use, or for special cases where the same object is updated several times in rapid succession, the integer event ID may not be contigous. The consumer should be prepared for an event ID with gaps.

Full resynchronization

Sometimes there can be a need for a full resynchronization, either because the systems have lost synchronization (e.g. after a restore) or because the downstream system has not been initialized yet (brand new).

The merchant can request a full resynchronization which will empty the current synchronization view, and then copy a reference for every user and product into the view (that are mapped to that store). The synchronization will then proceed normally and slowly get the two systems back in sync. Note that order data is never synchronized back to the remote system. This operation can be triggered from within the admin GUI.

Recommended method

First time setup: Create a new synchronization view and setup the callback URL. Do this only once or provide a button/trigger to allow the user to re-create/reinitialize the view if required. Note that the default settings in Kodmyran Commerce do not allow updating a synchronization view after initial creation.

The provided callback will be requested as soon as there are events available. The callback should request and process all events in order and once completed terminate with a HTTP 200 OK message. The request will always be a HTTP POST operation.

Sending new orders, users and products to Kodmyran Commerce is typically done through a job run from cron or a similar setup on the remote system. It can also be done by utilizing webhooks from the webshop to call a script, however this must usually be complemented by a cron style job anyway as webhooks are not usually 100% reliable.

It is assumed that the webshop can handle inventory levels on its own as it will not receive a callback for stock levels it itself is the reason for. The exception is structural products and composite warehouses which will incur an inventory callback regardless.

The consuming system must have logic in place that prevents an incoming synchronization from Kodmyran Commerce to occur at the same time as sending new/changed users, products and orders to Kodmyran Commerce. Without a mutually exclusive lock it is possible for the two systems to get out of sync with regards to stock levels.

Kodmyran Commerce requires HTTPS (encryption) for both inbound and outbound communication

Processing the event change list

The change list contains a meta header describing how many events there are in the log in total and the amount of entries in the current callback block. The event block itself is a list of entries with four fields each:

- The event log ID, this is the unique number of the event. This number may not be consecutive as events that do not pertain to this integration are not sent there (leaving a gap).
- The timestamp when the original log entry was made. This can be used for statistics about e.g. how long delay there is between change and apply. It does not serve any function otherwise.
- The type of log entry.
- The object itself.

There are currently four types of log entries:

- Product, this is a product object containing information about pricing, names, etc.

- User, this is a user object with address information about a customer
- Order, contains the status of an order. Fields included are the current status, the order rows, delivery information and payment references etc.
- Inventory, changes to inventory are reported separately from the product object as this can change much more frequently than the above fields. On some platforms, this avoids a costly update and cache invalidation cycle for products.

The format of each structure is identical to the format used when submitting new data as described in the next section.

If the remote system processing logic encounters a change type it does not recognize it should just skip that entry and continue with the next one. This ensures future compatibility when additional types are added to the protocol.

Example journal request from Kodmyran Commerce:

Request:

```
GET https://testaccount.shop4sale.se/admin/api/integrate/journal/
Accept: application/json
Content-Type: application/json
X-Api-Key: acebd1311088d194312de6220fdb9f66
X-SyncView: 5389273db0b0cbe40febfe706f4e316fad59fbe3
```

```
{
    "lastjournalid":180
}
```

Response:

```
{
    "callStatus":"OK",
    "message":"No error",
    "moredata":true,
    "journal": [
        {
            "meta": {
                "journalid":"189",
                "entity":"product",
                "entityid":"13572",
                "occurred":"2020-10-13 16:16:20",
                "mode":"update",
                "externalreference":"8304"
            },
            "data": {
```

```
"prodno":"13572",
"producttype":"physical",
"sku":"8304",
"gtin":"",
"taric":"0",
"origincountry":"",
"assoctype1":"0",
"assoctype2":"0",
"assoctype3":"0",
"assoctext1":"",
"assoctext2":"",
"assoctext3":"",
"price":376,
"orgprice":392,
"currency":"SEK",
"vatrate":25,
"weight":1000,
"note":"",
"name":[
    {
        "name":"Multiholk Funkis",
        "language":"sv"
    },
    {
        "name":"Tysk titel",
        "language":"de"
    },
    {
        "name":"Norsk titel",
        "language":"no"
    }
],
"description": [
    {
        "language":"sv",
        "long":"<p>En snygg multiholk<\/p>",
        "short":""
    },
    {
        "language":"de",
        "long":"<p>Ein multiholk<\/p>",
        "short":"Ein multiholk"
    },
    {
        "language":"no",
        "long":"",
                    "short":""
                }
            ],
            "specification":[],
            "stock": [
                {
                    "warehouseid":1,
                    "available":30,
                    "instock":95,
                    "threshold":0,
                    "location":""
                },
                {
                    "warehouseid":2,
                    "available":0,
                    "instock":0,
                    "threshold":0,
                    "location":""
                }
            ],
        "suppliers": [
            {
                "name":"Birdienamnam",
                "supplierid":45,
                "supplierpn":"BIRDIE2020",
                "purchaseprice":"10.00",
                "primarysupplier":true,
                "currency":"GBP"
            }
        ],
        "prices": [
            {
                "currency":"SEK",
                "price":376,
                "orgprice":392,
                "agreementid":0,
                "minamount":0,
                "validfrom":"1970-01-01 01:00:00",
                "validto":"1970-01-01 01:00:00"
            }
        ],
        "images":[]
    }
}
```

Sending new orders, users and products

To update or create new objects in Kodmyran Commerce you make a HTTP POST operation to one of the below endpoints. Operations other than POST are denied except for deleting (de-activating) a product which uses the HTTP DELETE operation.

We recommend synchronizing using the order: `product, user, order` as orders are generally dependent upon a user. Orders are also commonly dependent upon products making orders the last type of object to sync.

HTTPS is required for all operations.

Delete product

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.

You need to specify one or more selection keys, prodno takes precendence, followed by sku, and lastly gtin. The service will respond with a HTTP 200 message if the deletion was successful and a standard callStatus field.

Request:

```
DELETE https://testaccount.shop4sale.se/admin/api/integrate/product HTTP/1.0
Content-Type: application/json
X-API-Key: 6c38acfe12cc8192b
X-Syncview: 6c38acfe12cc8192b6c38acfe12cc8192b6c38acfe12cc8192b

{
    prodno: 1092,
    sku: "XYZ123",
    gtin: "7856321289371"
}
```

Response:

```
{
    callStatus: "OK",
    message: "No error"
}
```

Create/Update product

Endpoint:

```
https://testaccount.shop4sale.se/admin/api/integrate/product
```

When sending a new product to Kodmyran Commerce the initial warehouse levels will be respected and added to the inventory tables. When sending warehouse levels for existing products the stock level will be silently ignored unless explictly configured in Kodmyran Commerce to respect the level.

Example object:

```
{
    prodno: 2020,
    producttype: "item",
    displayid: "MX-1892",
    gtin: "714346319138",
    name: [
        { en: "Kermit the frog" },
        { sv: "Grodan Kermit" }
    ],
    association: 0,
    assoctext1: "Green",
    assoctext2: "",
    assoctext3: "",
    price: 145.00,
    orgprice: 145.00,
    currency: "SEK",
    vatrate: 25,
    weight: 800,
    stock: [
        {
            warehouseid: 2,
            available: 8,
            instock: 10,
            threshold: 2,
            location: "X2-04"
        },
        {
            warehouseid: 3,
            available: 0,
            instock: 0,
            threshold: 2,
            location: "B36"
        }
    ],
    suppliers: [
        {
            name: "SoftToys Inc",
            supplierid: 8,
            supplierpn: "KMIT34",
            purchaseprice: 4,
            primarysupplier: true,
            currency: "USD"
        }
    ],
    prices: [
        {
            currency: "SEK",
            price: 145.00,
            orgprice: 145.00,
            agreementid: 0,
            validfrom: "",
            validto: ""
        },
        {
            currency: "EUR",
            price: 13.00,
            orgprice: 13.00,
            agreementid: 0,
            validfrom: "",
            validto: "2017-10-28T23:59:59+01:00",
        }
    ]
    note: "",
    customfields: {}
}
```

Create/Update user

Endpoint:

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

Example object:

```
{
    userid: 1831,
    username: "kalle.anka@kodmyran.se",
    email: "kalle.anka@kodmyran.se",
    orgno: "420903-8596",
    language: "sv",
    newsletter: true,
    smsnotification: true,
    invoiceaddress: {
        company: "",
        firstname: "Kalle",
        lastname: "Anka",
        address1: "Storgatan 3",
        address2: "",
        zipcode: "12345",
        city: "Ankeborg",
        country: "SE"
    },
    deliveryaddress: {
        company: "Joakim von Anka AB",
        firstname: "Joakim",
        lastname: "von Anka",
        address1: "Penningkullen 1",
        address2: "",
        zipcode: "12345",
        city: "Ankeborg",
        country: "SE"
    },
    note: ""
}
```

Create order

Endpoint:

```
https://testaccount.shop4sale.se/admin/api/integrate/order
```

Example object:

```
{
    userid: 1831,
    orderid: 902,
    ordertimestamp: "2017-05-31T09:18:36+02:00".
    responsibleadminuserid: 1,
    storeid: 1,
    warehouseid: 1,
    customerreference: "CCX-3601",
    paymentmethod: "Klarna",
    paymentreference: "81561452961561",
    paymentdata: 365,
    freightmethod: "Express",
    currency: "SEK",
    language: "SV",
    invoiceaddress: {
        company: "",
        firstname: "Kalle",
        lastname: "Anka",
        address1: "Storgatan 3",
        address2: "",
        zipcode: "12345",
        city: "Ankeborg",
        country: "SE"
    },
    deliveryaddress: {
        company: "Joakim von Anka AB",
        firstname: "Joakim",
        lastname: "von Anka",
        address1: "Penningkullen 1",
        address2: "",
        zipcode: "12345",
        city: "Ankeborg",
        country: "SE"
    },
    comment: "",
    note: "",
    orderrows: [
        {
            sku: "MX-1892",
            name: "Kermit the frog",
            amount: 2,
            vatrate: 25,
            price: 140,
            discount: 0,
            discountmode: "abs"
        }
    ]
}
```

When sending new orders, it is important that the request includes a payment reference and the payment provider. This enables Kodmyran Commerce to be able to activate the payment at delivery time. For payment providers that may exist in multiple instances, such as Klarna, you should include the EID used in the request as well (the paymentdata field).

For Klarna specifically the payment reference is the Klarna reservation number, the older format with a passive invoice ID is not supported.

If you attempt to create a new order with the same orderid as a previous order the request will either be denied or responded to as a new object but referencing the older object (configuration setting).