

NolaPro API Documentation

The NolaPro API has been created to allow 3rd party applications to more easily put data into and get data out of NolaPro and to ensure that data coming in adheres to the core business rules of NolaPro. The API is based on a REST-style architecture and uses HTTP/HTTPS to manage communication between systems.

To access a resource in NolaPro you will need to construct an appropriate URL and use GET or POST to send data/parameters. The response will come back in XML format.

Example: Reduce the inventory count for NolaPro item 33454

URL = https://api.nolapro.com/rest/item/item_decrease

POST Fields:

data[itemid]=33454

data[locationid]=2

data[qty]=5

data[notes]='Waste'

auth[companyid]=1

auth[userid]=3

auth[url]=/rest/item/item_decrease

auth[time]=1197917516

auth[hash]=BKQ065Xg9H92vcdGMxW3ohKGUI8=

The URL format is <http://domain.com/rest/controller/action> where controller is the NolaPro entity or database table name and action is the process or function that you want to perform on that entity. In the example above the entity is item and we want to perform the action item_decrease. You will get a response back that says if there was an error during processing or if the request was successful.

We are continually building additional controllers and actions for new processing or data retrieval that users would like to do from 3rd-party applications.

Any programming language that can send post information via HTTP and can parse XML can be used to interact with the NolaPro API. We have constructed a PHP wrapper class that provides methods to simplify talking to the API.

This document provides details on the current capabilities of the API and also goes over how to construct requests in order to interact properly with the system.

Setting Up the API

The NolaPro API is written in PHP and requires PHP 5 running on Apache in order to function properly. We do not plan to test the application on any other web server programs (IIS, etc.).

Database Connection

To link the API to your NolaPro database open up in a text editor the file *api/app/config/database.php*. Adjust the values listed below to point to your database. Make these changes for both the \$default and \$test sections.

```
'host' => 'localhost',
'login' => 'root',
'password' => '',
'database' => 'nolapro',
```

Server Configuration

Turn on the short open tags option in your PHP configuration. The line in the php.ini file should read:

```
short_open_tag = On
```

Restart Apache after making any php.ini configuration changes.

You can access the API as a subdirectory off of your NolaPro site (<https://mysite.com/nolapro/api>), or you can set up a dedicated virtual host entry for it. We strongly encourage adding SSL and only creating a virtual host entry for port 443 (and not one for unencrypted port 80) if you make a dedicated API virtual host. The document root of the API should be *api/app/webroot*. Below is an example virtual host entry if you choose to use this method. If you are fine accessing the API off of your NolaPro directory, then you do not need to worry about adding the below section to your Apache configuration.

```
NameVirtualHost *:443
<VirtualHost *:443>
    DocumentRoot /var/www/html/api/app/webroot
    ServerName api.mydomain.com
    ErrorLog /var/log/apache2/api.mydomain.com-error_log
    SSLEngine on
    SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
    SSLCertificateFile /etc/apache2/ssl.crt/_mydomain.com.crt
    SSLCertificateKeyFile /etc/apache2/ssl.key/mydomain.key
    SSLCertificateChainFile /etc/apache2/ssl.crt/sf_issuing.crt
    <Files ~ "\.(cgi|shtml|phtml|php3?)"$">
        SSLOptions +StdEnvVars
    </Files>
    SetEnvIf User-Agent ".*MSIE.*" \
        nokeepalive ssl-unclean-shutdown \
        downgrade-1.0 force-response-1.0
    CustomLog /var/log/apache2/api.mydomain.com-access_log common
    <Directory />
        AllowOverride all
        Options -Indexes FollowSymLinks
    </Directory>
</VirtualHost>
```

```
Order Deny,Allow
Deny from all
Allow from 192.168.1.25
</Location>
</VirtualHost>
```

In the example above modify the <Location> section as needed to allow access only from certain trusted IPs. The API comes with its own authentication mechanism to prevent unauthorized access, but this additional measure offers even more protection.

NolaPro Setup

In order to use the NolaPro API you will need to activate the API add-on by first purchasing an activation code and then applying the code on the *Admin -> Setup -> Add-ons & Plug-ins* page.

After turning the API feature on you will need to generate an API key for one of your NolaPro users. All data entered via the API will be tagged with this user's ID. Go to *Admin -> Setup -> User Add/Update*. Choose the option *Generate New API Key* for the *API Key Management* select box. Click save. You should now see an API key for this user on the page next to *API Key*.

The next section will go into detail on how to create requests to the API.

Constructing a Proper API Request

Authentication

Every NolaPro request requires that 5 POST or GET fields be set. These fields are used to authenticate the requester and validate that the posted data is from the authenticated user and has not been modified.

- 1) **auth[url]** – This field should contain the API URL being requested minus the domain name. So for a request to `https://apidemo.nolapro.com/rest/gltransvoucher/save` the value of `auth[url]` would be `/rest/gltransvoucher/save`. For a request to `https://demo.nolapro.com/api/rest/gltransvoucher/save` `auth[url]` would also be `/rest/gltransvoucher/save`
- 2) **auth[companyid]** – This should be the ID from the `gencompany` table for the company that the action will be performed on. If you only have one company in NolaPro this will likely be ID 1.
- 3) **auth[userid]** – This is the ID of the NolaPro user that API requests will be made under. You can get this ID by going to Admin -> Setup -> User Add/Update and bringing up your API user. Look for the label *API User ID*. It can also be found in the `genuser` table in the database.
- 4) **auth[time]** – This should be the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT) (aka a Unix timestamp).
- 5) **auth[hash]** – The hash value helps ensure that the data being sent is authentic and from a source with access to the NolaPro user's API key. To get an API key for your user account go to Admin -> Setup -> User Add/Update. Look for the label *API Key Management*. Choose *Generate New API Key* and then save. After saving look for the label *API Key*. This is the secret code you'll need to construct a hash.

The hash should be built as follows:

Data String = Concatenation of all post fields except `auth[hash]` in the form `field1value1field2value2...(name=Jack&id=5&color=red` would be `nameJackid5colorred`)

Hash = `base64_encode(HMAC-SHA1 Signature(Data String, API Key))`

You can find an example of calculating the HMAC-SHA1 signature with PHP in the `npapi.php` file under the `hmacsha1()` method.

Data

Data that you'd like to send to the API needs to use an array-style naming convention similar to the authentication information (`auth[[]]`). All values need to be sent in fields using the name `data[[]]`.

Information to be saved for a specific model or table name should be in a multi-dimensional format like the following:

```
data[gltransvoucher][voucher]=TSTVCHR
data[gltransvoucher][description]=A Test Voucher from the API Test Page
data[gltransvoucher][wherefrom]=2
```

The field names to pass in are the same as the field names in the database table related to the model. You may want to use MySQL Query Browser or phpMyAdmin as a reference tool to get the field names you'll need when saving data.

You do not need to ever send in data[] fields for companyid, entrydate, createdate, entryuserid or createuserid as these will be filled in for you automatically based on values in auth[].

Info that needs to be sent, but that does not relate to a specific model can be sent without a model prefix simply as:

```
data[page]=1
data[limit]=25
data[order]=entrydate
data[conditions]=(id > 10) and entrydate < '2008-03-05'
data[fields][1]=entrydate
data[fields][2]=id
data[fields][3]=description
data[fields][4]=voucher
```

In some cases in order to properly perform an action via the API one or more additional related model elements may need to be submitted at the same time. An example of this is when saving a general ledger voucher (gltransvoucher). The voucher contains generic information about the transaction (description, when it was entered, when it was posted to the GL), but needs related gltransaction records to be submitted at the same time. The gltransaction records contain information about the amount of money going to each general ledger account in the voucher. The data fields for this type of submission would look like:

```
data[gltransvoucher][voucher]=TSTVCHR
data[gltransvoucher][description]=A Test Voucher from the API Test Page
data[gltransvoucher][wherefrom]=2
data[gltransaction][0][glaccountid]=1937
data[gltransaction][0][amount]=78.35
data[gltransaction][1][glaccountid]=1985
data[gltransaction][1][amount]=-78.35
```

The npapi PHP class contains some helper methods for creating these field names a little more easily.

Using the npapi class allows you to do something like the following to construct the field names automatically:

```
$api = new npapi();
$api->set_model('gltransvoucher');
$data = array(
    "voucher" => "TSTVCHR",
    "description" => "A Test Voucher",
    "wherefrom" => "2");
$api->set_data($data);

$api->set_model('gltransaction');
$data = array(
    array('glaccountid' => 1937, 'amount' => 78.35),
    array('glaccountid' => 1985, 'amount' => -1*78.35)
);
$api->set_data($data);

$api->action = "gltransvoucher/save";
$api->call_api();...
```

Sending the Request to the API

Most high-level programming languages have ways of sending an HTTP request to a URL and retrieving the contents. In PHP this can easily be accomplished using the curl library. You can see an example of this type of use in npapi.php under the call_api() method. The response back from the API will be an XML document.

Reading the Response

A very useful tool for monitoring the inputs and outputs of the API is NolaPro's API Log. You can access this via the file adminapilog.php in NolaPro or by going to Admin -> Setup -> API Log Viewer. Each transaction that goes through the API will be recorded here (failures and successes). You can click the View link to see the returned XML document and the input parameters. This can be very helpful when debugging. API logging is turned off by default. You can turn it on by going to Admin -> Setup -> System Settings and changing the NolaPro API Logging box to Yes.

Each response contains an **<rsp>** tag as the root element. **<rsp>** contains an attribute called **stat** that is set to either *ok* or *fail*. If **stat** is set to *fail* then there will be a child element called **<err>** with attributes of **code** and **msg** that contain information on the reason for the failure.

Example of a failure return message

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="fail">
  <err code="90010" msg="Invalid parent record ID. A record does not exist
for the given ID: Table = glaccount, Field = id, Value = 193766, companyid = 1 (See
[gltransaction][glaccountid] in data field)" />
</rsp>
```

Example of a success return message

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  <gltransvoucher id="582" href="/rest/gltransvoucher/view/582">
    </gltransvoucher>
</rsp>
```

PHP 5 offers a very useful tool for parsing XML documents called SimpleXML. You can see an example of parsing the returned document with SimpleXML in the npapi call_api() method.

API Controllers & Actions

To facilitate communication with the NolaPro Business Management System we have structured the API into a series of controllers, which in most cases represent a database table in NolaPro, and actions that can be performed on the controllers. For example, to insert data into the `gltransvoucher` table (used to store general ledger voucher details) you'll need to call the `save` action on the `gltransvoucher` controller. The `save` action performs several data validation and consistency checks and then adds the info to the database. A success or fail message is sent back in XML format.

All controllers have an action called ***search*** that you can perform to locate records. The ***search*** action has a quick URL format you can use as well as a post version which allows further refinement.

The ***search*** URL format is as follows:

`controller/search/field/value/operator`

Valid operators are: `=, <, <=, >, >=, <>, like` (the default operator is `=` so this can be left off requests using this)

Example:

`https://apidemo.nolapro.com/rest/gltransvoucher/search/post2date/2008-01-01`

This will return all general ledger vouchers posted to the date Jan 1, 2008.

For the ***search*** POST format you can send additional parameters to narrow the query.

data[recursive] = Whether to return related records to the model being searched. Set to 0 to perform no recursion. (ex. searching `gltransvoucher` with recursive set to 1 will return `gltransaction` records associated to `gltransvoucher` via `gltransaction.voucherid`)

data[page] = Represents the page number for results if you are using a limit (ex. get the 3rd page of results where 25 results are listed per page; `data[page]` would be 3 and `data[limit]` would be 25)

data[limit] = Reduce the number of records returned to this number (ex. to show only the first 10 results `data[limit]` would be 10)

data[order] = The order in which to sort the results. This should be a list of field names comma separated in the preferred sorting order. (ex. add `desc` to the field name to do a reverse sort as in `data[order] = last_name, first_name, orderid desc`)

data[conditions] = Allows you to add conditions on fields to further filter the results. The conditions can be built similarly to an SQL where clause (ex. `data[conditions] = (id > 10) and entrydate < '2008-03-05'`)

data[fields][n] = Where *n* is a positive integer this selector lets you choose which fields you would like to have returned. (ex.

`data[fields][1]=entrydate, data[fields][2]=id, data[fields][3]=description, data[fields][4]=voucher)`

The URL to perform a POST search would look like:

`https://apidemo.nolapro.com/rest/gltransvoucher/search`

All controllers also support the ***view*** action which allows you to easily retrieve one record by its ID. Simply construct a URL in the form `controller/view/id` (ex. `gltransvoucher/view/5678`)

Most controllers also support a *save* action which allows you to insert or update a record for a model. Construct the data[] field names by using the model name and table fields in the form data[model][field1]=value1 data[model][field2]=value2. If the field data[model][id] is not submitted, then an insert will be attempted. If data[model][id] is supplied, an update will run in most cases. As mentioned in the *Constructing a Proper API Request* section you can use the npapi class to simplify setting data fields for submission to the API.

Current Available Controllers & Actions

We are continually adding new controllers and actions to the API. Our current list is below.

accounttype

search

view

arcompany

search

view

save

arorderdetail

search

view

arordernotes

search

view

save

arordertax

search

view

arorder

search

view

save

arorder_deposit

search

view

save

arstatusoptions

search

view

save

carrierservice

search

view

save

carrier

search

view

save

ccaccountoption

search

view

save

ccaccountsetting

search

view

save

ccaccount

search

view

save

ccprocessor

search

view

save

checkacct

search

view

save

compositeitemid

search

view

save

customersalestax

search

view

save

customer

search

view

save

flexfield

search

view

save

flexvalue

search

view

save

gencompany

search

view

save

glaccount

search

view

save

get_balance

get_balance returns the current balance for the given general ledger account id.

Parameters can be passed into the *get_balance* action via the short URL form

glaccount/get_balance/id/date/status where **id** is the ID of the glaccount you want the balance for, **date** is the as-of date in YYYY-MM-DD form (ex. get the balance as of 2008-01-01) and **status** refers to posted vs unposted general ledger entries (0=unposted and 1=posted).

The POST form of the inputs should be data[id], data[date] and data[status]

get_activity

get_activity returns detailed transaction info for the given account and time period.

Parameters can be passed into the *get_activity* action via the short URL form *glaccount/get_activity/id/begindate/enddate/status* where **id** is the ID of the glaccount you want to get gltransvouchers for, **begindate** is the start of activity date in YYYY-MM-DD form, **enddate** is the final day of activity and **status** refers to posted vs unposted general ledger entries (0=unposted and 1=posted).

The POST form of the inputs should be data[id], data[begindate], data[enddate] and data[status]

gltransaction

search

gltransvoucher

search

view

save

gl_costcenter

search

view

save

igroup

search

view

save

invcompany

search

view

save

inventorylocation

search

view

save

invoiceterms

search

view

save

itemcategory

search

view

save

itemlocation

search

view

save

itemoptiongroup

search

view

save

itemtransaction

search

view

save

itemvendor

search

view

save

item

search

view

save

get_item_cost

get_item_cost returns the cost of the given item ID based on the inventory location ID.

The URL short form parameters for this action are *item/get_item_cost/itemid/locationid* where *itemid* is the ID of the item you want a cost value for and *locationid* is the inventory location that should be used in determining the cost. (NolaPro records the price paid to vendors when receiving inventory from POs into a specific inventory location in your company; this is used to determine the item cost. This is different from the item's price which is the amount you charge your customers for the product.)

POST form fields should be **data[itemid]* and **data[locationid]*.

get_item_stats

get_item_stats returns the on-hand, committed, available and on-PO quantities for the given item ID based on the inventory location ID.

The URL short form parameters for this action are *item/get_item_stats/itemid/locationid* where *itemid* is the ID of the item you want stock levels for and *locationid* is the inventory location that should be used .

POST form fields should be **data[itemid]* and **data[locationid]*.

item_increase

item_increase increases the on-hand inventory count for the given item ID by the quantity supplied for an inventory location. A cost is also required so that a GL entry can be made to account for the increase in the value of the inventory. An itemtransaction record will also be created to record the details of this increase.

The URL short form parameters for this action are

item/item_increase/itemid/locationid/qty/cost/offsetglaccountid/notes where *itemid* is the ID of the item you want to increase the count for, *locationid* is the inventory location that should be used, *qty* is the amount of the increase, *cost* is the per unit monetary value of the increase, *offsetglaccountid* is the GL account to use for the credit side of the transaction (the debit will be to the inventory account) and *notes* can contain additional information about the increase.

POST form fields should be **data[itemid]*, **data[locationid]*, **data[qty]*, **data[cost]*, *data[offsetglaccountid]* and *data[notes]*. (Starred fields are required.)

item_decrease

item_decrease decreases the on-hand inventory count for the given item ID by the quantity supplied for an inventory location. A GL entry will be made to account for the decrease in the value of the inventory. An itemtransaction record will also be created to record the details of this decrease.

The URL short form parameters for this action are

item/item_decrease/itemid/locationid/qty/offsetglaccountid/notes where *itemid* is the ID of the item you want to decrease the count for, *locationid* is the inventory location that should be used, *qty* is the amount of the decrease, *offsetglaccountid* is the GL account to use for the debit side of the transaction (the credit will be to the inventory account) and *notes* can contain additional information about the decrease.

POST form fields should be **data[itemid]*, **data[locationid]*, **data[qty]*, *data[offsetglaccountid]* and *data[notes]*. (Starred fields are required.)

item_group

search

view

save

markupset

search

view

save

pricelevel

search

view

save

priceperpriceunit

search

view

save

quotecomment

search

view

save

salesman

search

view

save

salestax

search

view

save

sales_categories

search

view

save

search_cat

search

view

save

search_tree

search

view

save

service_workclass

search

view

save

shiptosalestax

search

view

save

shipto

search

view

save

taxexempt

search

view

save

taxgroup

search

view

save

unitname

search

view

save

vendorsalestax

search

view

save

vendor

search

view

save