PHP EvE API / EvE-Central API Library (Ale - API Library for Eve)

Table of Contents

Requirements

For this library to be useful, you will need:

- A web server that runs PHP5. Some functions require PHP 5.1.3 or greater; the recommended version would be a recent version of PHP 5.2.x. This library uses SimpleXML.
- A web server that does not block the fsockopen() function. Unfortunately, many free PHP5 webhosts do block that function.
- A good deal of patience :)

Overview

eveapi-php grabs XML data through the EvE API or EvE-Central API, caches it locally in a text file, and parses the XML into an array for further processing by you.

The caching is a bit naive, but on the upside, very simple to maintain - no DB administration needed. There is no "cleanup" of the cache directory in this code at present. If you are going to use this API to grab data for many different users, that may become a small nuisance.

Installation

Copy the contents of the eveapi directory into a directory on your server. For purposes of this document, the examples will assume that you copied everything into './classes/eveapi'.

The EvE API library defaults to caching its fetches, in accordance with CCP's (and EvE-Central's) wishes. The default caching directory is './xmlcache/'.

Basic operation

You will need to use at least two includes in your code: 'class.api.php', which has the core class and does all the actual fetching of data, caching, and so on; and one or more of the parsing class files. In order to choose a character, for example, you'd need 'class.charselect.php'.

```
require_once('./classes/eveapi/class.api.php');
require_once('./classes/eveapi/class.charselect.php');
```

You'll also want the API User ID, API Key, and the character name. Those could be hardcoded or entered on the fly, stored in a DB, whatever. I'll hardcode them here. You get them from http://myeve.eve-online.com/api/, of course.

```
$apiuser = '12345'; // User ID
$apipass = '67890'; // API Key - limited or full, depending on what you intend to fetch
$mychar = 'Win Sauce'; // Character name
```

Time to initialize a new Api object. You'll want to decide on whether to debug; and you'll set the credentials. Note we just use User ID and API Key to start. You can only fetch rather limited things with that combo, but that's how we need to start out.

```
$api = new Api();
$api->setDebug(true); // When testing and debugging your code – comment out once everything's working
$api->setCredentials($apiuser,$apipass);
```

Alright, now we'll go fetch us the character ID for that character name, and we'll grab the corporation ID while we're at it. Fetching anything in the PHP API Library is done in two steps: Get the raw XML, then parse it using a parser class. The function name in Api and in the parsing class is identical, by convention in this library.

```
$apicharsxml = $api->getCharacters();
$apichars = Characters::getCharacters($apicharsxml);
```

That was simple enough. Now step through the $apichars array to find the character we'll use.

```
foreach($apichars as $index => $thischar)
{
        if($thischar['charname']==$mychar)
        {
                $apichar=$thischar['charid'];
                $apicorp=$thischar['corpid'];
        }
}
```

Really what you'd do is present the existing characters to a user to choose from, but this works for a demonstration example.

At this point, you can continue by grabbing more data, using the $apichar variable. For example, to get a list of the members of your corporation (which assumes an include of class.membertracking.php - remember, each API function has a separate parsing class):

```
// The following fetch will need our character ID - so set it, now that we have it
$api->setCredentials($apiuser,$apipass,$apichar);
$membersxml = $api->getMemberTracking();
$members = MemberTracking::getMemberTracking($membersxml);
```

I trust you get the idea. In a nutshell, you use a one-two-punch of grabbing the XML, and then parsing the XML. The names of the parsing and XML grabbing functions are identical, and the name of the parsing class matches the function minus the “get”.
Example: api->getWalletJournal and WalletJournal::getWalletJournal.

Lastly, if you had debug on and you want to output the debug messages, use this:

```
$api->printErrors();
```

There's an example api-testing.html file included in this release, which shows some of the functions of the API. It fetches hard-coded user ID and API key from config.php, so edit config.php before you upload the code to your server for testing.

Error Reporting

Error reporting is rudimentary in this version of the library. The individual functions will return “null” if an API error was encountered. If $api->setDebug(true); has been set, you can print out any encountered errors with $api->printErrors();. Currently, this also outputs informational debug information, such as cache file creation, cache loading, &c.

The library offers getApiError() to get the API result code of the last fetch, and getApiErrorText() to get the text accompanying the error code - see the documentation for getApiError() and getApiErrorText(), below.

If setDebug(false) (the default), PHP warning messages will be suppressed when an API server goes offline. If setDebug(true), then PHP warning messages and the exception created by SimpleXML choking on the 404 page will be shown.

The library will throw an exception if it detects incorrectly passed parameters. This is a change from previous versions, where the library would silently discard incorrectly passed parameters.

Control functions for the EvE API PHP library

The examples in this section assume that you created a variable $api for the Api class, like so:

```
$api = new Api();
```

setCredentials(userid, apikey, charid)

Set the user ID, API key (full or limited) and character ID to be used in subsequent API operations.

Returns “true” if successful, and “false” if not.

```
$api->setCredentials(userid, apikey, charid);
```

“userid” and “apikey” are required parameters, while “charid” is optional.
Note that “charid” is a numeric value. You can grab it using getCharacters – see above example, please – or using getCharacterID(name);.

You may also use this function to “wipe” previously set authentication, by passing “null” for the parameters.

getCredentials()

Returns an array containing ‘userid’, ‘apikey’ and ‘charid’ indices filled with the vales set previously with the setCredentials call.

```
$credentials = $api->getCredentials();
```

getCacheStatus()

Return “true” if the previous API call had been serviced from cache, and “false” if the previous API call had not been serviced from cache.

```
$dataxml = $api->getMemberTracking();
$cached = $api->getCacheStatus(); // Did we just fetch that, or did we get it from our cache?
```

getApiError()

Return '0' on a successful API call, or the numeric API error code. The plain-text error message can be gotten with getApiErrorText().

```
$dataxml = $api->getMemberTracking();
$error = $api->getApiError(); // Did we encounter an API error?
```

getApiErrorText()

Return an empty string on a successful API call, or the plain-text API error message.

```
$dataxml = $api->getMemberTracking();
$error = $api->getApiError(); // Did we encounter an API error?
If ($error) { $errstring = $api->getApiErrorText(); // What does the code mean? }
```

setDebug(bool)

Set debug output to “true” or “false”. It defaults to “false”.

Returns “true” if successful, and “false” if not.

$api->setDebug(true);

getDebug()

Return the value of the debug level, “true” or “false”.

$debug = $api->getDebug();

setUseCache(bool)

Set caching to “true” or “false”. It defaults to “true”.

Returns “true” if successful, and “false” if not.

$api->setUseCache(false);

getUseCache()

Return the value of the caching variable, “true” or “false”.

$caching = $api->getUseCache();

setCacheDir(dir)

Set the caching directory to a string. It defaults to “./xmlcache”.

Returns “true” if successful, and “false” if not.

$api->setCacheDir(“/var/tmp/dont-live-in-my-homedir-you-bum/”);

getCacheDir()

Return the caching directory as a string value.

$cachedir = $api->getCacheDir();

getCacheFile()

Return the name of the cache file, including path, that was created or used during the last API call. Returns “null” if no cache file was in use.

$cachefile = $api->getCacheFile();

setUserAgent(agent)

Set the HTTP User Agent to a string. It defaults to “eve-apiphp-x.x”, where “x.x” is the version of the library.

Returns “true” if successful, and “false” if not.

```
$api->setUserAgent(“FleetManager”);
```

getUserAgent()

Return the HTTP User Agent string.

```
$useragent = $api->getUserAgent();
```

setTimeTolerance(tolerance)

Set the amount of time in minutes that the API functions will wait after CCP’s “cachedUntil” time has expired before attempting to call an API function again instead of delivering from cache. Default is 5 minutes. Used to allow some leeway in your server’s clock being “fast” compared to CCP’s clock.

Returns “true” if successful, and “false” if not.

```
$api->setTimeTolerance(10); // 5 minutes, pshaw, we need 10
```

getTimeTolerance()

Return the value (in minutes) of the timeTolerance variable.

```
$tolerance = $api->getTimeTolerance();
```

getCacheTime([local-bool])

Return the time the cache file used during the last API call was created, as a GMT or local timestamp, based on CCP's “currentTime” hint. Returns null if no “currentTime” hint is present, which is the case for EvE-Central and CharacterPortrait calls.

```
$dataxml = $api->getMemberTracking();
$cachetime = $api->getCacheTime(); // When was this data returned by the API?
```

local-bool is “true” if you wish a local timezone timestamp returned, and “false” if you wish a GMT timestamp returned. It defaults to “false”.

getExpiryTime([local-bool])

Return the time the cache file used during the last API call will expire, as a GMT or local timestamp, based on CCP's “cachedUntil” hint. Returns null if no “cachedUntil” hint is present, which is the case for EvE-Central and CharacterPortrait calls.

```
$dataxml = $api->getMemberTracking();
$cachetime = $api->getExpiryTime(); // When will the API allow me to fetch again?
```

local-bool is “true” if you wish a local timezone timestamp returned, and “false” if you wish a GMT timestamp returned. It defaults to “false”.

setApiSite(site)

Set the value of the URL that the library will use to connect to the EvE API server. Defaults to "api.eve-online.com".

Returns “true” if successful, and “false” if not.

$api->setApiSite(“sisi-api.eve-online.com”);

getApiSite()

Return the apisite URL as a string.

$apisite = $api->getApiSite();

setApiSiteEvEC(site)

Set the value of the URL that the library will use to connect to the EvE-Central API server. Defaults to "eve-central.com".

Returns “true” if successful, and “false” if not.

$api->setApiSiteEvEC(“test.eve-central.com”);

getApiSiteEvEC()

Return the apisiteevec URL as a string.

$apisite = $api->getApiSiteEvEC();

printErrors()

Print debug messages. These may be actual errors, or merely informational debug.

Implemented EvE API functions

See the very handy API index at http://wiki.eve-id.net/APIv2_Page_Index for EvE server requirements for each API function and sample output, please.

The file name and class name of each API function is derived from its URL where possible, with the exception of CharacterPortrait and the EvE-Central Minerals function. E.g., the “Factional Warfare Top 100 Stats” are fetched from “/eve/FacWarTopStats.xml.aspx”, which means the class is called “FacWarTopStats”, is kept in “class.facwartopstats.php”, and the functions for fetching the data are both called getFacWarTopStats - one on the API class, and one on the FacWarTopStats class.

Note that all of the fetching functions have an optional “timeout” parameter. “timeout” is the time to cache the data in minutes. By default, functions will cache for a time specified by CCP in the returned data – with the current exception of WalletJournal and WalletTransactions, which default to 65 minutes because CCP’s “cachedUntil” value for these is different from other functions, and the library does not yet account for that.

You can use the test code in the sample directory to print the raw array data of any function, if you need to see how the array is formatted.

Account Functions

Characters

Needed to get the char ID, without which a lot of the other fetches don't work.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.characters.php');
```

Authentication needed:

User ID and limited API Key

```
$api->setCredentials($apiuser,$apipass);
```

Fetching and parsing:

```
$dataxml = $api->getCharacters([timeout]);
$data = Characters::getCharacters($dataxml);
```

Character Functions

Character Sheet

A detailed character sheet for one of your characters.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once(‘./classes/eveapi/class.charactersheet.php’);
```

Authentication needed:

This needs the character ID to work, and a limited API key.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml=$api->getCharacterSheet([timeout]);
$data = CharacterSheet::getCharacterSheet($dataxml);
```

Skill in Training

Returns the currently training skill for one of your characters. Use the array returned by “getSkillTree” to find the name of the skill that corresponds to the id that will be returned.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once(‘./classes/eveapi/class.skillintraining.php’);
```

Authentication needed:

This needs the character ID to work, and a limited API key.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml=$api->getSkillInTraining([timeout]);
$data = SkillInTraining::getSkillInTraining($dataxml);
```

Character or Corporation Functions

Account Balance

Grab the Account Balance for either the character, or the corp the character belongs to.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.accountbalance.php');
```

Authentication needed:

This needs the character ID to work.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml = $api->getAccountBalance([corp-bool],[timeout]);
$data = AccountBalance::getAccountBalance($dataxml);
```

- “corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”.

Asset List

Grab the Asset List for either the character, or the corp the character belongs to. In order to fetch for corp, the character has to have the “Director” or “CEO” role in the corp.

This function may return error code “115: Assets already downloaded”. If you are using caching, the library will use the “retry after” hint on the 115 error to change the cachedUntil value of the cache file, if the “timeout” parameter is “null”.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.assetlist.php');
```

Authentication needed:

This needs the character ID and full API key.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml = $api->getAssetList([corp-bool],[timeout]);
$data = AssetList::getAssetList($dataxml);
```

- “corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”
- “timeout” defaults to null, which tells the library to follow cachedUntil and use the 115 error to modify cachedUntil. Set to a specific value if you wish to avoid this behavior.

Factional Warfare Statistics

Statistics for Factional war

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.facwarstats.php');
```

Authentication needed:

This may need the limited or full API key. Exact requirements for fetching for corp unknown – we don't have a factional warfare character to test with.

Fetching and parsing:

```
$dataxml = $api->getFacWarStats([corp-bool],[timeout]);
$data = FacWarStats::getFacWarStats($dataxml);
```

“corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”

Industry Jobs

Grab the list of Industry Jobs for either the character, or the corp the character belongs to. To fetch for corp, the character has to have the ‘Factory Manager’ role.

This function may return error code “116: Industry Jobs already downloaded”. If you are using caching, the library will use the “retry after” hint on the 116 error to change the cachedUntil value of the cache file, if the “timeout” parameter is “null”.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.industryjobs.php');
```

Authentication needed:

This needs the character ID and full API key.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml = $api->getIndustryJobs([corp-bool],[timeout]);
$data = IndustryJobs::getIndustryJobs($dataxml);
```

- “corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”
- “timeout” defaults to null, which tells the library to follow cachedUntil and use the 116 error to modify cachedUntil. Set to a specific value if you wish to avoid this behavior.

Kill Log (Killmails)

A array of killmails for a character or corporation, including all details that are available in game.

This function may return error code “119: Kills exhausted” during iteration. If you are using caching, the library will use the “retry after” hint on the 119 error to change the cachedUntil value of the cache file, if the “timeout” parameter is “null”.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once(‘./classes/eveapi/class.killlog.php’);
```

Authentication needed:

This needs the character ID to work. If used for corp, you’ll need to be a Director or CEO of your corp.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml = $api->getKilllog([killid],[corp-bool],[timeout]);
$data =  KillLog::getKillLog($dataxml);
```

- “killid” – if specified, you will get up to 1000 entries *before* that refid. If not specified, you will get the newest entries, up to 1000.
- “corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”
- “timeout” defaults to null, which tells the library to follow cachedUntil and use the 119 error to modify cachedUntil. Set to a specific value if you wish to avoid this behavior.

Market Orders

A list of market orders that are either not expired or have expired in the past week (at most).

This function may return error code “117: Market orders already downloaded”. If you are using caching, the library will use the “retry after” hint on the 117 error to change the cachedUntil value of the cache file, if the “timeout” parameter is “null”.

Returns an Array if successful, and “null” if not

Include needed:

```
require_once('./classes/eveapi/class.marketorders.php');
```

Authentication needed:

This needs the character ID to work. If used for corp, you’ll need to have the “accountant” role in that corp.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml = $api->getMarketOrders([corp-bool],[timeout]);
$data = MarketOrders::getMarketOrders($dataxml);
```

- “corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”
- “timeout” defaults to null, which tells the library to follow cachedUntil and use the 117 error to modify cachedUntil. Set to a specific value if you wish to avoid this behavior.

Medals

Lists the Medals received or for corporation medals created and awarded.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.medals.php');
```

Authentication needed:

This needs the character ID to work, and a limited API key.

Fetching and parsing:

```
$dataxml = $api->getMedals([corp-bool],[timeout]);
$data = Medals::getMedals($dataxml);
```

“corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”

Standings

Lists the standings towards other alliances/corporations/individuals, for either a character or a corporation

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.standings.php');
```

Authentication needed:

This needs the character ID to work, and a limited or full API key.

Fetching and parsing:

```
$dataxml = $api->getStandings([corp-bool],[timeout]);
$data = Standings::getStandings($dataxml);
```

“corp-bool” is “true” if you wish to fetch for corp, and “false” if you wish to fetch for character. It defaults to “false”

Wallet Journal Entries

A detailed wallet journal (like the one in the EvE UI) showing transactions up to one week back. Returns a maximum of 1000 entries in one go, but can be run multiple times to get more than 1000 entries, as long as they are not older than one week.

CCP recommends to run this function, and run it again until it returns less than 1000 entries, for simplicity's sake.

Common error codes to be returned by this function during iteration are "101: Wallet exhausted" and "103: Already returned one week of data" as well as "100: Expected before ref/trans ID = 0: wallet not previously loaded". If you are using caching, the library will use the "retry after" hint on 101 and 103 errors to change the cachedUntil value of the cache file, if the "timeout" parameter is "null".

Returns an array if successful, and "null" if not.

Include needed:

require_once('./classes/eveapi/class.walletjournal.php');

Authentication needed:

This needs the character ID to work, and a full API key. If used for corp, you'll need to have the "junior accountant" role in that corp.

$api->setCredentials($apiuser,$apipass,$apichar);

Fetching and parsing:

$dataxml=$api->getWalletJournal([refid],[corp-bool],[accountkey],[timeout]);
$data = WalletJournal::getWalletJournal($dataxml);

- "refid" – if specified, you will get up to 1000 entries *before* that refid. If not specified, you will get the newest entries, up to 1000.
- "corp-bool" is "true" if you wish to fetch for corp, and "false" if you wish to fetch for character. It defaults to "false".
- "accountkey" defaults to "1000", which is the corp master wallet or the char wallet. Other possible values are 1001 through 1006, for the available corp wallet divisions.
- "timeout" defaults to null, which tells the library to follow cachedUntil and use the 101/103 errors to modify cachedUntil. Set to a specific value if you wish to avoid this behavior.

Wallet (Market)Transactions

A detailed view of wallet market transactions (like the one in the EvE UI) showing transactions up to one week back. Returns a maximum of 1000 entries in one go, but can be run multiple times to get more than 1000 entries, as long as they are not older than one week.

CCP recommends to run this function, and run it again until it returns less than 1000 entries, for simplicity's sake.

Common error codes to be returned by this function during iteration are "101: Wallet exhausted" and "103: Already returned one week of data" as well as "100: Expected before ref/trans ID = 0: wallet not previously loaded". If you are using caching, the library will use the "retry after" hint on 101 and 103 errors to change the cachedUntil value of the cache file, if the "timeout" parameter is "null".

Returns an array if successful, and "null" if not.

Include needed:

```
require_once('./classes/eveapi/class.wallettransactions.php');
```

Authentication needed:

This needs the character ID to work, and a full API key. If used for corp, you'll need to have the "junior accountant" role in that corp.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml=$api->getWalletTransactions([transid],[corp-bool],[accountkey],[timeout]);
$data = WalletTransactions::getWalletTransactions($dataxml);
```

- "transid" – if specified, you will get up to 1000 entries *before* that transid. If not specified, you will get the newest entries, up to 1000.
- "corp-bool" is "true" if you wish to fetch for corp, and "false" if you wish to fetch for character. It defaults to "false".
- "accountkey" defaults to "1000", which is the corp master wallet or the char wallet. Other possible values are 1001 through 1006, for the available corp wallet divisions.
- "timeout" defaults to null, which tells the library to follow cachedUntil and use the 101/103 errors to modify cachedUntil. Set to a specific value if you wish to avoid this behavior.

Corporation Functions

Container Log

Return a list of all containers owned by the corporation, and their contents.

Returns an array if successful, and “null” if not.

Include needed:

require_once(‘./classes/eveapi/class.containerlog.php’);

Authentication needed:

This needs the character ID to work, and a full API key. A CEO or Director key is required.

$api->setCredentials($apiuser,$apipass,$apichar);

Fetching and parsing:

```
$dataxml=$api->getContainerLog([timeout]);
$data = ContainerLog::getContainerLog($dataxml);
```

Corporation Sheet

A detailed corporation sheet for the corporation you are a member of. Can also be used to see information about corporations in an alliance.

Returns an array if successful, and “null” if not.

Include needed:

require_once(‘./classes/eveapi/class.corporationsheet.php’);

Authentication needed:

This needs the character ID to work, and a limited or full API key. A CEO or Director will see more data than a member.

$api->setCredentials($apiuser,$apipass,$apichar);

Fetching and parsing:

```
$dataxml=$api->getCorporationSheet([corpid],[timeout]);
$data = CorporationSheet::getCorporationSheet($dataxml);
```

- “corpid” is to be used if you wish to see information about a corporation in an alliance. If omitted, information about your character’s corp is returned.

Member Medals

List the medals awarded to a corporation's members.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.membermedals.php');
```

Authentication needed:

This needs the character ID to work, and a limited or full API key.

Fetching and parsing:

```
$dataxml = $api->getMemberMedals([timeout]);
$data = MemberMedals::getMemberMedals($dataxml);
```

Member Security

List the security roles of a corporation's members.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.membersecurity.php');
```

Authentication needed:

This needs the character ID to work, and a full API key. A CEO or Director key is required.

Fetching and parsing:

```
$dataxml = $api->getMemberSecurity([timeout]);
$data = MemberSecurity::getMemberSecurity($dataxml);
```

Member Security Log

List information about the changes in security roles of a corporation's members, and the member who made the changes.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.membersecuritylog.php');
```

Authentication needed:

This needs the character ID to work, and a full API key. A CEO or Director key is required.

Fetching and parsing:

```
$dataxml = $api->getMemberSecurityLog([timeout]);
$data = MemberSecurity::getMemberSecurityLog($dataxml);
```

Member Tracking

Get a list of a corporation's members. CEOs and Directors will see more information.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.membertracking.php');
```

Authentication needed:

This needs the character ID to work, and a full API key.

Fetching and parsing:

```
$dataxml = $api->getMemberTracking([timeout]);
$data = MemberTracking::getMemberTracking($dataxml);
```

Shareholders

Returns the character and corporation share holders of a corporation.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.shareholders.php');
```

Authentication needed:

This needs the character ID to work, and a full API key. A CEO or Director key is required.

Fetching and parsing:

```
$dataxml = $api->getShareHolders([timeout]);
$data = ShareHolders::getShareHolders($dataxml);
```

Starbase (POS) Details

Returns details about a particular starbase that belongs to your corp. You’ll need to have CEO or Director roles to get this list.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once(‘./classes/eveapi/class.starbasedetail.php’);
```

Authentication needed:

This needs the character ID to work, and a full API key. You’ll need to be a Director or CEO of your corp

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml=$api-> getStarbaseDetail ($id,[timeout]);
$data = StarbaseDetail:: getStarbaseDetail ($dataxml);
```

- id – This is the ID of the POS as returned by getStarbaseList

Starbase (POS) List

Returns a list of starbases that belong to your corp. You'll need to have CEO or Director roles to get this list.

Returns an array if successful, and "null" if not.

Include needed:

```
require_once('./classes/eveapi/class.starbaselist.php');
```

Authentication needed:

This needs the character ID to work, and a full API key. You'll need to be a Director or CEO of your corp.

```
$api->setCredentials($apiuser,$apipass,$apichar);
```

Fetching and parsing:

```
$dataxml=$api-> getStarbaseList ([timeout]);
$data = StarbaseList:: getStarbaseList ($dataxml);
```

Titles

List the titles held by a corporation's members.

Returns an array if successful, or "null" if not

Include needed:

```
require_once('./classes/eveapi/class.titles.php');
```

Authentication needed:

This needs the character ID to work, and a full API key. A CEO or Director key is required.

Fetching and parsing:

```
$dataxml = $api->getTitles([timeout]);
$data = Titles::getTitles($dataxml);
```

EvE Global Functions

Alliance List

Returns all of the member corp IDs of a particular alliance, which can then be used when fetching the corp sheet data. This is a global fetch, no authentication is needed.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.alliancelist.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getAllianceList([timeout]);
$data = AllianceList::getAllianceList($dataxml);
```

Certificate Tree

Lists certificates and their requirements.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.certificatetree.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getCertificateTree([timeout]);
$data = CertificateTree::getCertificateTree($dataxml);
```

Conquerable Station/Outpost List

Lists station ID, name,station type ID, solar system ID, corporation ID and corporation name of conquerable stations.

Include needed:

```
require_once('./classes/eveapi/class.conquerablestationlist.php');
```

Authentication needed:

None.

Fetching and parsing:

```
$dataxml = $api->getConquerableStationList([timeout]);
$data = ConquerableStationList::getConquerableStationList($dataxml);
```

Error List

A list of API error codes and their plain-text descriptions. This is a global fetch, no authentication is needed.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.alliancelist.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getAllianceList([timeout]);
$data = AllianceList::getAllianceList($dataxml);
```

Factional Warfare Top 100 Stats

List the top 100 in factional warfare

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.facwartopstats.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getFacWarTopStats([timeout]);
$data = FacWarTopStats::getFacWarTopStats($dataxml);
```

ID to Name Conversion (undocumented, “CharacterName”)

Lists the names of the IDs passed to it. I can resolve most IDs, including alliances, corporations, characters, factions and item types.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.charactername.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getCharacterName(ids,[timeout]);
$data = CharacterName::getCharacterName($dataxml);
```

“ids” is a comma-separated list of IDs

Name to ID Conversion (“CharacterID”)

Lists IDs of the names you have passed to it. It can be alliance, corporation or characters.

Returns an array if successful, or “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.characterid.php');
```

Authentication needed:

none

Fetching and parsing:

```
$dataxml = $api->getCharacterID(names,[timeout]);
$data = CharacterID::getCharacterID($dataxml);
```

“names” is a comma-separated list of names

ID to Character Portrait

When called this function returns the portrait of the character that corresponds to the ID passed to it. Unlike all other EvE API functions, it does not return XML, but a path to a JPEG file. Therefore, there is no corresponding “parse” function or class.

Returns the path to the fetched JPEG if successful, or “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.api.php');
```

Authentication needed:

None

Fetching:

```
$path = $api->getCharacterPortrait(id,[size],[timeout]);
```

- “id” – The ID of the character whose portrait you are fetching.
- “size” - The size of the portrait. The EvE API currently recognizes the values “64” and “256”
- “timeout” defaults to 1440 minutes

RefTypes List

Gets a complete list of refTypes, which are needed to map the data returned by getWalletJournal and getWalletTransactions into types of transactions, such as “Player Donation”. This is a global fetch, no authentication is needed.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.reftypes.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getRefTypes([timeout]);
$data = RefTypes::getRefTypes($dataxml);
```

Server Status

Lists the status of the server and number of pilots online.

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.serverstatus.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getServerStatus([timeout]);
$data = ServerStatus::getServerStatus($dataxml);
```

Skill Tree

Returns the complete EvE-Online Skill Tree. Useful, for example, for mapping the id of SkillInTraining to a skill name.

Include needed:

```
require_once('./classes/eveapi/class.skilltree.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getSkillTree([timeout]);
$data = SkillTree::getSkillTree($dataxml);
```

Map Functions

Factional Warfare Occupancy Map

Lists the systems occupied by factions in the faction war system .

Returns an array if successful, or “null” if not

Include needed:

```
require_once('./classes/eveapi/class.facwarsystems.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getFacWarSystems([timeout]);
$data = FacWarSystems::getFacWarSystems($dataxml);
```

Jumps

Returns the number of ship jumps for each solar system in the last hour. If a system is not listed it has had no jumps.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.jumps.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getJumps([timeout]);
$data = Jumps::getJumps($dataxml);
```

Kills

Lists the number of kills within solar systems in the last hour. If a system is not listed it has had no kills.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once('./classes/eveapi/class.kills.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getKills([timeout]);
$data = Kills::getKills($dataxml);
```

Sovereignty

Lists Solar systems and which factions or alliances claim them.

Returns an array if successful, or “null” if not

Include needed:

require_once('./classes/eveapi/class.sovereignty.php');

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getSovereignty([timeout]);
$data = Sovereignty::getSovereignty($dataxml);
```

EvE-Central functions

The EvE-Central API is provided by eve-central.com. Neither the API nor the data returned by it are maintained by CCP.

Minerals (“EvEMon”)

Retrieve the median mineral prices in the empire regions in a format digestable by EvEMon.

Include needed:

require_once('./classes/eveapi/class.minerals.php');

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getMinerals([timeout]);
$data = Minerals::getMinerals($dataxml);
```

- “timeout” defaults to 30 minutes.

Quick Look

Retrieve all of the available market orders, including prices, stations, order IDs, volumes, etc.

Returns an array if successful, and “null” if not.

Include needed:

```
require_once(‘./classes/eveapi/class.quicklook.php’);
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml=$api->getQuickLook(typeid,[sethours],[regionlimit],[usesystem],[setminQ],[timeout]);
$data = QuickLook::getQuickLook($dataxml);
```

- “typeid” – The type ID of the item you wish to see market orders for. This ID can be gotten from the EvE data dump.
- “sethours” – Get orders which have been posted within the last X hours. Defaults to 360.
- “regionlimit” – Restrict the view to only show from within the specified region IDs. These Ids can be gotten from the EvE data dump. Can be specified multiple times. If specifying multiple times, pass an array of region IDs; otherwise, pass a numeric value.
- usesystem – Restrict the view to this system only, specified by system ID. This ID can be gotten from the EvE data dump.
- setminQ – Restrict the view to show only orders above the specified quantity.
- “timeout” defaults to 30 minutes.

Market Statistics

Retrieve aggregate statistics for the items specified.

Returns an array if successful, and “null” if not.

Include needed:

require_once(‘./classes/eveapi/class.marketstat.php’);

Authentication needed:

None

Fetching and parsing:

```
$dataxml=$api->getMarketStat(typeid,[sethours],[regionlimit],[setminQ],[timeout]);
$data = MarketStat::getMarketStat($dataxml);
```

- “typeid” – The type ID of the item you wish to see market orders for. This ID can be gotten from the EvE data dump. Can be specified multiple times. If specifying multiple times, pass an array of type IDs; otherwise, pass a numeric value.
- “sethours” – Get orders which have been posted within the last X hours. Defaults to 360??.
- “regionlimit” – Restrict statistics to specific region IDs. These IDs can be gotten from the EvE data dump. Can be specified multiple times. If specifying multiple times, pass an array of region IDs; otherwise, pass a numeric value.
- setminQ – The minimum quantity in an order to consider it for the statistics .
- “timeout” defaults to 30 minutes.