PHP EvE API / EvE-Central API Library (eveapi-php)

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.

Note that PHP itself will throw Warning: error messages if the API server cannot be resolved or cannot be reached. You may want to set error_reporting() in your script if you do not wish to have these Warning: messages displayed when the API server is down.

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);.

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?

getCredentials()

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

$credentials = $api->getCredentials();

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();

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(tolerance)

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

$tolerance = $api->getTimeTolerance();

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.

Note that all of the parsing 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 broken.

You can use “print_r($data);” to print the raw array data of any function, if you need to see how the array is formatted. If you are going to do that, a “header("Content-type: text/plain");” might help to keep the formatting – though that’ll screw up any HTML formatting on your page, of course. “For testing purposes only”.

Account Balances

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.balances.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”.

Alliance List

This is a global fetch, no authentication is actually needed. An APIv1 function that Rynlam added to the mix. Handy for getting all of the member corp IDs of a particular alliance, which can then be used when fetching the corp sheet data.

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);
```

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.

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

Character List (on account)

Needed to get the char ID, whithout 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.charselect.php');
```

Authentication needed:

User ID and API Key

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

Fetching and parsing:

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

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.

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

Fetching and parsing:

$dataxml=$api->getCharacterSheet([timeout]);
$data = CharacterSheet::getCharacterSheet($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.conquerablestations.php');

Authentication needed:

Authentication not required.

Fetching and parsing:

$dataxml = $api->getConquerableStations();
$data = ConquerableStations::getConquerableStations($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. 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.

Error List

An APIv2 function that has not been implemented yet. Have at it!

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.

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"

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.

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

Include needed:

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

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

Authentication needed:

This needs the character ID to work. 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 65 minutes, because the "cachedUntil" value for this function is broken as of Empyrian Age 1.0.1

Kill Log (Killmails)

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

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 a Director or CEO your corp.

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

Fetching and parsing:

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

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.

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

Include needed:

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

previously just class.transactions.php

Authentication needed:

This needs the character ID to work. 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 65 minutes, because the "cachedUntil" value for this function is broken as of Empyrian Age 1.0.1

Market Orders

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

Returns an Array if successful, and "null" if not

Include needed:

require_once('./classes/eveapi/class.api.php');
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();
$data = MarketOrders::getMarketOrders($dataxml);

Map: Jumps

Returns the number of ship jumps for each solarsystem, 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.api.php');
require_once('./classes/eveapi/class.jumps.php');
```

Authentication needed:

None

Fetching and parsing:

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

Map: Kills

Lists the number of kills within solarsystems in the last hour

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

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

Map: Sovereignty

Lists Solarsystems and which factions or alliances claim them

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getsovereignty();
$data = Sovereignty::getSovereignty($dataxml);
```

Name to ID Conversion

Lists ID's 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.api.php');
require_once('./classes/eveapi/class.characterid.php');
```

Authentication needed:

none

Fetching and parsing:

```
$dataxml = $api->getCharacterID('Yorick Downe,CCP Garthagk');
$data = CharacterID::getCharacterID($dataxml);
```

the only parameter is a comma separated list of names

ID to Name Conversion (undocumented)

Lists the names of the ID's past 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.api.php');
require_once('./classes/eveapi/class.charactername.php');
```

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getCharacterName('221710318,797400947');
$data = CharacterName::getCharacterName($dataxml);
```

The only parameter is a comma separated list of IDs

ID to Character Portrait

When called this function returns the portrait of the character that corresponds to the ID passed to it.

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

```
$path_large = $api->getCharacterPortrait(797400947,256);
```

The first parameter is the ID of the character whom portrait you wish to receive followed by the size of the portrait, Valid sizes are 64 and 256

RefTypes List

This is a global fetch, no authentication is actually needed. An APIv1 function that 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”.

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

$dataxml = $api->getRefTypes([timeout]);
$data = RefTypes::getRefTypes($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.charactersheet.php’);

Authentication needed:

This needs the character ID to work.

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

Fetching and parsing:

$dataxml=$api->getSkillInTraining([timeout]);
$data = SkillInTraining::getSkillInTraining($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.generic.php');

Authentication needed:

None

Fetching and parsing:

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

$api->setCredentials($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.

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

Fetching and parsing:

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

Factional Warfare Stats

Stats for Factional war

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

Include needed:

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

Authentication needed:

Limited/Full?

Fetching and parsing:

$dataxml = $api->getFacWarStats();
$data = FacWarStats::getFacWarStats($dataxml);

Factional Warfare Top 100 Stats

List the top 100 factional in warfare

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

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

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

Authentication needed:

None

Fetching and parsing:

```
$dataxml = $api->getFacWarSystems();
$data = Jumps::getFacWarSystems($dataxml);
```

MedalMembers

Lists medals awarded to corporations members (not sure on this as for me this is blank)

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

Include needed:

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

Authentication needed:

api key of director/ceo

Fetching and parsing:

```
$dataxml = $api->getMemeberMedals();
$data = MemberMedals::getMemberMedals($dataxml);
```

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

Authentication needed:

Api key, assuming director privileges for corporation

Fetching and parsing:

```
$dataxml = $api->getMedals();
$data = Medals::getMedals($dataxml);
```

CertificateTree

Lists certificates and their requirements

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

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

ServerStatus

Lists the status and number of pilots online.

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

Include needed:

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

Authentication needed:

None

Fetching and parsing:

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