
# Trading API
# Reference & Developer Guide

**Version 1.1**

**February 25, 2013**

# Revision History

| Version | Date | Changes |
|---|---|---|
| `1.0` | `2011-12-06` | Initial version. |
| `1.1` | `2013-02-25` | Fractional Trading support: Changed data types of `amount` parameters of `placeOrder` and `getOrders`. |

# Table of Contents

# 1 Overview

This document describes the VirWoX (Virtual World Exchange) Trading API (Application Programming Interface). This API is an easy-to-use, standards-based, programming-language-independent interface to Web Services, enabling access to the exchange functionality of VirWoX by programs.

**APPLICATIONS**

The Trading API gives developers access to the exchange functionality and market liquidity offered by VirWoX. Typical applications of the Trading API are:

- Automatic trading on the VirWoX exchange ("trading bots").
- Integration of virtual currency exchange functionality into other services (e.g. merchant services).
- Alternative user interfaces for the VirWoX exchange.

The protocol is completely stateless, i.e. the server does not remember state information between requests (e.g. there is no "login" request).

For using the Trading API you will need a VirWoX account, and an application key (see Chapter 2).

# 2 Getting Started

**GET AN APP KEY**

In order to access the production system, your application will need an app key to identify itself to the VirWoX server. In order to request such a key, log in to your VirWoX account and go to

**https://www.virwox.com/key.php**

**TESTING**

VirWoX runs a test system at

**https://www.virwox.com:8000**

where you can test your application with test accounts and fake money. You do not need an app key for the test system; just use TEST as the app key.

You can create register a test account like in the production system. You can also create test money in your test account with this page:

**https://www.virwox.com:8000/make_test_funds.php**

(you must be logged in to your test account).

# 3 Protocol

**SECURE HTTP**

In the Trading API we transmit sensitive information, so all data is sent over secure HTTP (https). In order to make access from a wide variety of programming language simple and lightweight, we use the JSON-RPC protocol over HTTPS.

**JSON-RPC**

JSON[1] (JavaScript Object Notation) is a lightweight data interchange format whose simplicity has resulted in widespread use among web developers. JSON is easy to read and write; you can parse it using any programming language, and its structures map directly to data structures used in most programming languages. JSON-RPC[2] is a lightweight Remote Procedure Call (RPC) protocol using JSON for object serialization.

The access point for the JSON-RPC over HTTP interface is:

**https://www.virwox.com/api/trading.php**

We support two request options: via HTTP POST and via HTTP GET.

**HTTP POST**

Using HTTP POST[3], the client sends a JSON-encoded request object with the following properties:

- **`method`** - A string containing the name of the method to be invoked.
- **`params`** - An array of objects to pass as arguments to the method.
- **`id`** - The request id. This can be of any type. It is used to match the response with the request that it is replying to.

The service responds with a JSON-encoded object with the following properties:

- **`result`** - The object that was returned by the invoked method. This is *null* in case there was an error invoking the method.
- **`error`** - An error object if there was an error invoking the method. It is *null* if there was no error.
- **`id`** - This is the same id as the request it is responding to. This allows to send and receive requests asynchronously, and tell which answer is for which request.

**EXAMPLE**

For example, to invoke the `getBalances` method (see Section 4.1), you would POST the following string (whitespace added for readability):

```
{
  "method" : "getBalances",
  "params" :
  {
    "key"  : "your_app_key",
    "user" : "your_username",
    "pass" : "your_password"
  },
  "id" : 1234
}
```

[1] http://www.ietf.org/rfc/rfc4627.txt

[2] http://json-rpc.org/wiki/specification

[3] The *Content-Type* header of the posted content must not be *application/x-www-form-urlencoded* or *multipart/form-data*.

The server would respond with something like this (again, whitespace has been added for readability):

```
{
  "result":
  {
    "errorCode" : "OK",
    "accountList" :
    [
      {
        "currency" : "SLL",
        "balance" : "20111.74"
      },
      {
        "currency" : "EUR",
        "balance" : "3.47"
      },
    ]
  },
  "error" : null,
  "id" : 1234
}
```

Library functions to produce the request and parse the response into objects are available for most programming languages.

**HTTP GET**

To make it even easier to invoke methods from some programming environments (and in fact, even interactively from a web browser), we also support the "Google AJAX API Style" of calling JSON functions, i.e. encoding the request as url-form-encoded parameters. To issue the same call as in the example above, you can fetch

**https://www.virwox.com/api/trading.php?method=getBalances&key=your_app_key&user=your_username&pass=your_password**

e.g. by entering it into the address bar of your browser. The service will respond as above.

Similarly, you can also POST

```
method=getBalances&key=your_app_key&user=your_username&pass=your_password
```

to the access point URL, with the *Content-Type* header of the POST request set to *application/x-www-form-urlencoded*.

# 3.1 Request Rate Limits

**BY APPLICATION**

To prevent abuse, VirWoX limits the rate with which API requests can be made. The limits are defined by application, as identified by the app key. It is therefore important to keep your app key secret so that nobody else can restrict the access of your application to the server.

**DEFAULT LIMITS**

The default request limits are:

- 60 requests per minute
- 600 requests per hour

Different limits are possible on request.

**HTTP RESPONSE DATA**

If the limit is exceeded, the server will not process the request. Instead, it will respond with HTTP status code `503 Service Temporarily Unavailable`, and set the HTTP response header `Retry-After` to the number of seconds the client should wait before sending the next request. Note that even a rejected request will count towards the limit, so you cannot just send requests as fast as possible and expect that some would be processed while some would be rejected. Instead, the client should try to avoid that requests are ever rejected because of the limit. To facilitate this, the server sends the following rate limit information in additional HTTP headers of every response:

| HTTP Header | Description |
|---|---|
| `X-Rate-Limit-Minute` | The number of requests per minute allowed for this app key (default 60). |
| `X-Rate-Remaining-Minute` | The number of requests remaining until the per-minute limit is exceeded. |
| `X-Rate-Reset-Minute` | The number of seconds until `X-Rate-Remaining-Minute` will be restored to `X-Rate-Limit-Minute`, assuming that no requests are made during this time. |
| `X-Rate-Limit-Hour` | The number of requests per hour allowed for this app key (default 600). |
| `X-Rate-Remaining-Hour` | The number of requests remaining until the per-hour limit is exceeded. |
| `X-Rate-Reset-Hour` | The number of seconds until `X-Rate-Remaining-Hour` will be restored to `X-Rate-Limit-Hour`, assuming that no requests are made during this time. |
| `X-Rate-Cost` | The cost of the current request (default 1). See below. |

**PENALTIES**

A normal request reduces the number of requests allowed until the limit is reached by 1. However, there are more costly requests:

- A `placeOrder` call costs the equivalent of 5 normal requests.
- Any request with a wrong username/password combination is penalized with the cost of 30 normal requests.

# 4 API Services Reference

**INHERITED METHODS**

This chapter contains descriptions of each of the methods supported by the Trading API. In addition to the methods listed below, the methods available in the Basic API are also available in the Trading API. Please refer to

http://api.virwox.com/api/documentation/Basic_API.pdf

for documentation of the following methods inherited from the Basic API:

- `getInstruments`
- `getBestPrices`
- `getMarketDepth`
- `estimateMarketOrder`
- `getTradedPriceVolume`
- `getRawTradeData`
- `getStatistics`
- `getTerminalList`
- `getGridList`
- `getGridStatistics`

**Note that all methods need the `key` parameter as described in Chapter 2, in addition to the parameters documented below. The same is also true for the methods inherited from the Basic API.**

Example code can be found in Chapter 6.

## 4.1 getBalances

**PURPOSE**

Use this method to get the current balances of the specified VirWoX account. In general, a VirWoX account has multiple balances in different currencies, so this method returns a list of balances.

**INPUT**

| Parameter | Optional | Type | Description |
|---|---|---|---|
| `user` | N | `string` | The username of the VirWoX account. |
| `pass` | N | `string` | The password of the VirWoX account. |

**OUTPUT**

The following attributes:

| Parameter | Type | Description |
|---|---|---|
| `errorCode` | `ErrorEnum` | `INVALID_USERNAME_OR_PASSWORD`, `ACOUNT_DISABLED`, or `OK` (see Chapter 5). |
| `accountList` | `AccountList` | An array of currencies and the associated balances (see below). |

The `accountList` array is a list of items of type `Account`, containing the following data:

| Parameter | Type | Description |
|---|---|---|
| `currency` | `string` | A currency code (e.g. “USD” or “SLL”). Today this is always 3 letters, but could be longer in the future. |
| `balance` | `Amount` | The current balance in this currency. |

# 4.2 getCommissionDiscount

**PURPOSE**

VirWoX offers heavy traders a discount on the commissions to be paid for limit orders, which is calculated based on the commission already paid in the previous 30 days (see https://www.virwox.com/help.php#_Commissions_and_Discounts).

This method returns both the commission paid to VirWoX by the specified user, plus the current discount on commissions in percent. You can use this method to calculate the commission that needs to be paid for a limit order before placing it (together with the `commissionRate` from the `getInstruments` method). Note that the commission for a limit order is determined at the time of placing it, so even if the discount changes after the order is placed, it remains the same.

**INPUT**

| Parameter | Optional | Type | Description |
|---|---|---|---|
| `user` | N | `string` | The username of the VirWoX account. |
| `pass` | N | `string` | The password of the VirWoX account. |
| `days` | Y | `int` | The number of elapsed days (optional; default=30) to take into account when generating the `commission` array. |

**OUTPUT** The following attributes:

| Parameter | Type | Description |
|---|---|---|
| `errorCode` | `ErrorEnum` | `INVALID_USERNAME_OR_PASSWORD`, `ACOUNT_DISABLED`, or `OK` (see Chapter 5). |
| `commission` | `CommissionList` | An array of currencies and the associated commission paid in that currency (see below). If no commission was paid, this attribute will be omitted from the output. |
| `discountPct` | `int` | The discount (in percent) for limit orders. Note that this will only be the actual value used in determining the commission when `days=30`. |

The `commissionList` array is a list of items of type `Commission`, containing the following data:

| Parameter | Type | Description |
|---|---|---|
| `currency` | `string` | A currency code (e.g. “SLL” or “ACD”). The special code “total” contains the total commission paid in all currencies, which is ultimately used in determining the discount. |
| `amount` | `Amount` | The commission paid in this currency in the specified period. |

# 4.3 getTransactions

**PURPOSE** Use this method to search for transactions by the specified user. A number of optional search criteria can be specified, which will be combined using a Boolean AND. Transactions returned will be ordered by decreasing `transactionID` (i.e. the latest transactions are returned first). At most 3000 transactions can be returned by a single call.

**INPUT**

| Parameter | Optional | Type | Description |
|---|---|---|---|
| `user` | N | `string` | The username of the VirWoX account. |
| `pass` | N | `string` | The password of the VirWoX account. |
| `transactionID` | Y | `int` | The ID of a specific transaction. |

| | | | |
|---|---|---|---|
| exchangeID | Y | int | The ID of an order the transaction is related to. You can use this attribute to retrieve all the transactions related to a specific order. |
| startdate | Y | string | A timestamp in the format “YYYY-MM-DD hh:mm:ss”. Only transactions at or after that time will be returned. |
| enddate | Y | string | A timestamp in the format “YYYY-MM-DD hh:mm:ss”. Only transactions at or before that time will be returned. |
| currency | Y | string | If specified, only transactions in this currency will be returned. |
| transactionType | Y | tType | If specified, only transactions of this type (see Notes below) will be returned |
| limit | Y | int | Only this many transactions will be returned (the most recent ones). The maximum (which is also the default) is 3000. |

**OUTPUT** The following attributes:

| Parameter | Type | Description |
|---|---|---|
| errorCode | ErrorEnum | INVALID_USERNAME_OR_PASSWORD, ACOUNT_DISABLED, or OK (see Chapter 5). |
| transactions | TransactionList | An array of transaction records (see below). If no transactions were found, this attribute will be omitted from the output. |

The transactions array is a list of items of type Transaction, containing the following data:

| Parameter | Type | Description |
|---|---|---|
| transactionID | int | The unique ID of the transaction. |
| currency | string | The currency of the transaction. |
| amount | Amount | The amount of the transaction. |
| balance | Amount | The balance of the account associated with the transaction after the transaction has been performed (see also getBalances). |

| | | |
|---|---|---|
| transactionType | tType | The type of the transaction (see Notes below). |
| exchangeID | int | The orderID the transaction is associated with (see also getOrders and Notes below). |
| processedAt | string | A timestamp in the format “YYYY-MM-DD hh:mm:ss” when the transaction was processed. See Notes below. |

**NOTES**

The following transaction types are currently defined:

- PLACE_ORDER: An order has been placed
- PARTIAL_FILL: An order has been partially filled
- FULL_FILL: An order has been fully filled
- CANCEL_ORDER: An order has been cancelled
- WITHDRAW: A withdrawal request
- DEPOSIT: A bank deposit
- PAYPAL: A PayPal deposit
- MONEYBOOKERS: A Skrill (Moneybookers) deposit
- NETELLER: A NETELLER deposit
- PAYSAFECARD: A paysafecard deposit
- BITCOIN: A bitcoin deposit or withdrawal
- INACTIVITY: Inactivity fee
- VOUCHER: A gift voucher
- PARTNER: Partner commission
- PAYMENT: A payment between VirWoX accounts (see MicroPayment API documentation)
- FEE: A fee associated with certain payments
- OTHER: Anything else (e.g. manually cancelled transactions)

The exchangeID parameter of the transaction record only contains a valid orderID if the transactionType is one of PLACE_ORDER, PARTIAL_FILL, FULL_FILL, or CANCEL_ORDER. If the transactionType is PAYMENT, the exchangeID will be negative and contain a paymentID instead.

All timestamps are in Central European Time (CET).

# 4.4 getOrders

**PURPOSE**

Use this method to search for exchange orders by the specified user. A number of optional search criteria can be specified, which will be combined using a Boolean AND. Transactions returned will be ordered by decreasing `orderID` (i.e. the latest orders are returned first). At most 3000 orders can be returned.

**INPUT**

| Parameter | Optional | Type | Description |
|---|---|---|---|
| `user` | N | `string` | The username of the VirWoX account. |
| `pass` | N | `string` | The password of the VirWoX account. |
| `selection` | Y | `string` | One of `OPEN` or `HISTORIC` (default). Depending on this parameter, the call will return the currently active (open or partially filled) orders, or inactive (already fully executed or cancelled) orders. |
| `orderID` | Y | `int` | If this parameter is set, only the specified order will be returned. |
| `limit` | Y | `int` | Only this many orders will be returned (the most recent ones). The maximum (which is also the default) is 3000. |

**OUTPUT**

The following attributes:

| Parameter | Type | Description |
|---|---|---|
| `errorCode` | `ErrorEnum` | `INVALID_USERNAME_OR_PASSWORD`, `ACOUNT_DISABLED`, or `OK` (see Chapter 5). |
| `orders` | `OrderList` | An array of order records (see below). |

The `orders` array is a list of items of type `Order`, containing the following data:

| Parameter | Type | Description |
|---|---|---|
| `orderID` | `int` | The unique ID of the order. |
| `instrument` | `string` | The name of the instrument traded. E.g, in the instrument `EUR/SLL`, `EUR` is called the `longCurrency`, and `SLL` the `shortCurrency` (see also the `getInstruments` method). |
| `orderType` | `string` | Can be `BUY` or `SELL`, and refers to the |

| | | |
|---|---|---|
| | | longCurrency. E.g. in the case of EUR/SLL, BUY means to buy EUR for SLL, SELL means to sell EUR for SLL. |
| price | double | The price at which the limit order is placed. If price=0, this is a market order. |
| amountOpen | Amount | The amount of longCurrency that is currently open (i.e. not yet filled). If amountOpen=0, the order is fully filled. |
| amountFilled | Amount | The amount of longCurrency that is already filled. Orders can be partially filled; the total order size is amountOpen+amountFilled. |
| commission | double | The commission that has been paid on this order (0 if order has not yet been filled). |
| discountPct | int | The discount (in percent) that this order was placed with (see getCommissionDiscount method). |
| orderStatus | string | one of OPEN (the order has been placed but not yet been filled), PARTIAL (the order has been filled in part but is still active), FILLED (the order has been fully filled) or CANCELLED (the order has been cancelled; it may have been partially filled before, see amountFilled). |
| placedAt | string | A timestamp in the format “YYYY-MM-DD hh:mm:ss” when the order was placed. |
| cancelledAt | string | A timestamp in the format “YYYY-MM-DD hh:mm:ss” when the order was cancelled. Contains null if not cancelled. |
| filledAt | string | A timestamp in the format “YYYY-MM-DD hh:mm:ss” when the order was (at least partially) filled. In case of multiple partial fills, this is the time of the latest part. Contains null if not filled. |
| volumeFilled | Amount | The volume (measured in shortCurrency) that was filled. The average price the order was filled with (e.g. in case of a market order) can be calculated as amountFilled/volumeFilled. |

# 4.5 placeOrder

**PURPOSE** This method places an order. It is used for both Market and Limit orders.

**INPUT**

| Parameter | Optional | Type | Description |
|---|---|---|---|
| user | N | string | The username of the VirWoX account. |
| pass | N | string | The password of the VirWoX account. |
| instrument | N | string | The name of the instrument (e.g. EUR/SLL). The getInstruments method returns a list of tradeable instruments. E.g, in the instrument EUR/SLL, EUR is called the longCurrency, and SLL the shortCurrency. |
| orderType | N | string | Can be BUY or SELL, and refers to the longCurrency E.g. in the case of EUR/SLL, BUY means to buy EUR for SLL, SELL means to sell EUR for SLL. |
| amount | N | Amount | The number of units of longCurrency to buy or sell. Must be at least minimumOrder, (see getInstruments), or the error INVALID_AMOUNT_OR_PRICE will be returned. Values will be rounded to decimalsOrder digits (see getInstruments). |
| price | Y | double | If this parameter is specified, the order is a limit order, and this is the price of the order limit. Omit this parameter (or pass 0) for placing a market order. |
| estimate | Y | double | Only relevant for market orders. If this parameter is specified, the server will return a PRICE_CHANGED error code, if the cost of a buy order is higher than this estimate or the proceeds of a sell order are lower. The value is specified in shortCurrency. In a typical use case, you would use the output of an estimateMarketOrder call as this parameter, to make sure that the order is executed only if the market has not moved against your favour between estimateMarketOrder and placeOrder. |

**OUTPUT** The following attributes:

| Parameter | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| `errorCode` | `ErrorEnum` | `OK`, or one of these Error Codes (see Chapter 5):<br>`INVALID_USERNAME_OR_PASSWORD`<br>`ACCOUNT_DISABLED`<br>`NO_SUCH_INSTRUMENT`<br>`INVALID_ORDER_TYPE`<br>`INVALID_AMOUNT_OR_PRICE`<br>`INSUFFICIENT_FUNDS`<br>`INSUFFICIENT_LIQUIDITY`<br>`PRICE_CHANGED`<br>`DATABASE_TIMEOUT` |
| `orderID` | `int` | Only returned if `errorCode=OK` (i.e. the order has been placed successfully). This is the ID of the order (which can be used as input to the `getOrders` and `getTransactions` methods to retrieve details about it, or `cancelOrder` to cancel it. |

# 4.6 cancelOrder

**PURPOSE** This method cancels a (limit) order.

**INPUT**

| Parameter | Optional | Type | Description |
|---|---|---|---|
| `user` | N | `string` | The username of the VirWoX account. |
| `pass` | N | `string` | The password of the VirWoX account. |
| `orderID` | N | `int` | The ID of the order to cancel. |

**OUTPUT** The following attribute:

| Parameter | Type | Description |
|---|---|---|
| `errorCode` | `ErrorEnum` | `OK`, or one of these Error Codes (Chapter 5):<br>`INVALID_USERNAME_OR_PASSWORD`<br>`ACCOUNT_DISABLED`<br>`NO_SUCH_ORDER`<br>`DATABASE_TIMEOUT` |

# 5 Error Codes

**ERRORENUM**

The following error codes are defined:

| Value | Meaning |
|---|---|
| OK | No error. |
| INVALID_USERNAME_OR_PASSWORD | The specified username and/or password is invalid[4]. |
| NO_TARGET_CUSTOMER | The recipient does not exist. |
| NO_SOURCE_ACCOUNT_FOR_THIS_CURRENCY | Unsupported currency. |
| NO_TARGET_ACCOUNT_FOR_THIS_CURRENCY | Invalid combination of currency and payment target. |
| INVALID_AMOUNT_OR_PRICE | Invalid amount or price. |
| INSUFFICIENT_FUNDS | The account has not enough funds in the necessary currency for the requested operation. |
| NO_SUCH_INSTRUMENT | The requested instrument cannot be traded. |
| NO_SUCH_ORDER | The specified order does not exist, is not from the specified user, or has been filled or cancelled in the meantime. |
| INVALID_ORDER_TYPE | Invalid order type. |
| DATABASE_TIMEOUT | The request for a lock in the database has timed out. This is a temporary problem, and the operation should be retried. |
| NOT_UNIQUE | The username or SL-username is not unique. |
| ILLEGAL_PARAMETER | An illegal parameter has been sent to the server. |
| MANUAL_INTERVENTION_REQUIRED | The payout request could not be fulfilled instantly. A manual check is required. |
| ACCOUNT_DISABLED | The specified VirWoX account has been disabled. |
| LIMIT_EXCEEDED | The deposit or withdrawal amount exceeds the current limit of the user. |

[4] If this error code is generated, the response is delayed for 3 seconds.

| | |
|---|---|
| | The user should specify a smaller amount. |
| INSUFFICIENT_LIQUIDITY | There is not enough liquidity available for a market order and the specified amount. The user should specify a smaller amount. |
| PRICE_CHANGED | The estimated price of a market order has changed against the user's favor. The market order has not been placed. The user can repeat the request with the new estimate. |
| COULD_NOT_SEND_EMAIL | An email could not be sent. |
| PAYPAL_API_ERROR | An error in the PayPal API has occurred. |
| NETELLER_API_ERROR | An error in the NETELLER API has occurred. |
| PSC_API_ERROR | An error in the **paysafe**card API has occurred. |
| TOKEN_EXPIRED | The token has expired (or never existed). |
| UNSUPPORTED_PAYMENT_TARGET | The payment target specified in the `targetType` parameter is not allowed. |
| NO_SUCH_PAYMENT | The payment with the specified `paymentID` does not exist. |

**NOTES** Not all error codes are used in the Trading API, but are listed here for consistency and completeness.

# 6 Programming Examples

This chapter contains an example on how to access the Trading API. For the time being, only PHP examples are supplied.

## 6.1 PHP Class VirWOXTradingAPI

The following is a generic PHP class interface to the Trading API, which makes the sample code simpler. Construct the class with your app key as parameter (‘TEST’ accesses the test system). There also is an optional debug mode.

**TRADINGAPI.PHP**

```
<?php
/*****************************************************************************
 * Class VirWOXTradingAPI
 *
 * This is a proxy for the remote API endpoint, using JSON-RPC.
 *
 * Construct an object of this class with your application key.
 * If key=='TEST', we connect to the test system.
 */

define('VIRWOX_TRADING_API_URL', 'https://www.virwox.com/api/trading.php');

class VirWoXAPIException extends Exception {}

class VirWoXTradingAPI
{
  private $id = 1;                   // id of the JSON RPC calls (incremented on each call)
  private $key;
  private $debug;
  private $http_header;

  public function __construct($key, $debug = false)
  {
    $this->key = $key;
    $this->debug = $debug;
  }

  public function __call($name, $arguments)     // arguments is a map with named arguments
  {
    $keyparam = array('key' => $this->key);
    $params = $arguments ? array_merge($arguments[0], $keyparam) : $keyparam;

    $request = array(
      'method'  => $name,
      'params'  => $params,
      'id'      => $this->id++
    );

    if ($this->debug)
      print"<pre>Request: ".print_r($request, true);

    $context = stream_context_create(array(
      'http' => array(
         'method'  => 'POST',
         'header'  => 'Content-Type: application/json\r\n',
         'content' => json_encode($request)
      )
    ));

    @$response = file_get_contents(VIRWOX_TRADING_API_URL, false, $context);
    $this->http_header = $http_response_header;    // remember HTTP header

    if ($response === false)    // rate limit reached or other HTTP error:
      throw new VirWoXAPIException('HTTP Error: '.$this->http_header[0]);

    $JSON = json_decode($response);
```

```
    if ($this->debug)
      print"Response: ".print_r($JSON,true)."</pre>\n";

    if (!$JSON->result)
      throw new VirWoXAPIException("VirWoX API error: $JSON->error");

    return $JSON->result;
  }

  // return the server's rate limit info sent with the last request:
  public function getRateLimitInfo()
  {
    if ($this->http_header)
      foreach ($this->http_header as $line)
        if (preg_match('/^X-Rate-([^:]*): (.*)/', $line, $matches))
          $output[$matches[1]] = $matches[2];
    return $output;
  }
}
?>
```

# 6.2 Demo Program

This is a comprehensive example that uses all of the methods of the Trading API, plus some methods inherited from the Basic API. We place an order, cancel it, and display order data and related transactions. Before placing the order, we check whether we have enough money to do so, which requires reading our current balance, the current discount level, as well as the commission rate for the traded instrument. The buy order is placed 10 units below the currently best buy price, so that it will not get filled before being cancelled.

At the end, the program also displays the current rate limit info (see Section 3.1). Should we trade too fast, the trading API will throw an Exception with HTTP error code 503. In a real application, you should check the rate limit info to avoid this.

**TRADING_DEMO.PHP**

```
<?php
/*****************************************************************************
 * Demo program that shows how to use the trading API.
 *
 * We will first get the current account balance. If enough SLL are available we will
 * place a limit order to buy 1 EUR for SLL, then cancel it, display information
 * related to the order; and finally display the API rate limiting info.
 *****************************************************************************/

require_once('tradingAPI.php');       // the API proxy class VirWoXPaymentAPI

define('APP_KEY', 'your_app_key');    // use 'TEST' for the test system

define('VIRWOX_USERNAME', 'your_username');
define('VIRWOX_PASSWORD', 'your_password');

define('INSTRUMENT', 'EUR/SLL');      // the financial instrument we want to trade

try
{
  $api = new VirWoXTradingAPI(APP_KEY, false);   // open API (true means debug on)

  // get & reorganize instruments as associative array:
  $result = $api->getInstruments();
  foreach ($result as $instr)
    $instrument[$instr->symbol] = $instr;

  // get current best BUY price for INSTRUMENT
  $result = $api->getBestPrices(array('symbols' => array(INSTRUMENT)));
  if ($result[0]->errorCode != 'OK')
    die ("Could not determine current price. Error Code = {$result[0]->errorCode}\n");

  // make sure there actually is a best offer to buy:
  $best_buy = $result[0]->bestBuyPrice;
  if (!$best_buy)
    die ("No offer to buy ".INSTRUMENT." available");
```

```
  // we want to place our order 10 units below the current best price:
  $our_price = $best_buy - 10 / pow(10, $instrument[INSTRUMENT]->decimals);

  // get current balances of our account:
  $result = $api->getBalances(array('user' => VIRWOX_USERNAME,
                                    'pass' => VIRWOX_PASSWORD));
  if ($result->errorCode != 'OK')
    die("Could not access account. Error Code = {$result->errorCode}\n");

  // reorganize account list as associative array; and display it for debugging
  foreach ($result->accountList as $account)
    $balance[$account->currency] = $account->balance;
  print "Balances: ".print_r($balance,true);

  // get our current discount level:
  $result = $api->getCommissionDiscount(array('user' => VIRWOX_USERNAME,
                                              'pass' => VIRWOX_PASSWORD));
  if ($result->errorCode != 'OK')
    die("Could not get discount. Error Code = {$result->errorCode}\n");
  $discountPct = $result->discountPct;
  $my_commission_rate = $instrument[INSTRUMENT]->commissionRate * (1 - $discountPct / 100);
  print "Current discount: $discountPct%, my commission: ".(100*$my_commission_rate)."%\n";

  // determine cost (in shortCurrency) needed for the buy order:
  $amount = 1;    // buy 1 unit of longCurrency
  $cost = $amount * $our_price * (1 + $my_commission_rate);
  print "Cost = $cost {$instrument[INSTRUMENT]->shortCurrency}\n";
  if ($balance[$instrument[INSTRUMENT]->shortCurrency] < $cost)
    die ("Not enough {$instrument[INSTRUMENT]->shortCurrency} for buy order!\n");

  // place the order:
  $result = $api->placeOrder(array('user' => VIRWOX_USERNAME,
                                   'pass' => VIRWOX_PASSWORD,
                                   'instrument' => INSTRUMENT,
                                   'orderType' => 'BUY',
                                   'price' => $our_price,
                                   'amount' => $amount));
  if ($result->errorCode != 'OK')
    die("Could not place order. Error Code = {$result->errorCode}\n");
  $orderID = $result->orderID;    // remember order ID
  print "Placed order ID=$orderID.\n";

  // now cancel it:
  $result = $api->cancelOrder(array('user' => VIRWOX_USERNAME,
                                    'pass' => VIRWOX_PASSWORD,
                                    'orderID' => $orderID));
  print "Cancel Order Error Code = {$result->errorCode}\n";

  // show order info:
  $result = $api->getOrders(array('user' => VIRWOX_USERNAME,
                                  'pass' => VIRWOX_PASSWORD,
                                  'orderID' => $orderID));
  if ($result->errorCode != 'OK')
    die("Could not get order info. Error Code = {$result->errorCode}\n");
  print "\nOrder Info: ".print_r($result->orders, true);

  // show related transactions:
  $result = $api->getTransactions(array('user' => VIRWOX_USERNAME,
                                        'pass' => VIRWOX_PASSWORD,
                                        'exchangeID' => $orderID));
  if ($result->errorCode != 'OK')
    die("Could not get related transactions. Error Code = {$result->errorCode}\n");
  print "\nRelated transactions: ".print_r($result->transactions, true);

  print "\nRate Limit Info: ".print_r($api->getRateLimitInfo(), true);
}
catch (Exception $e)
{
  print "Exception: ".$e->getMessage();
  print "\nRate Limit Info: ".print_r($api->getRateLimitInfo(), true);
}
?>
```

**OUTPUT** This example will produce an output similar to this:

```
Balances: Array
(
    [SLL] => 5137.80
    [EUR] => 7.47
)
Current discount: 0%, my commission: 3.9%
Cost = 358.5589 SLL
Placed order ID=1191423.
Cancel Order Error Code = OK

Order Info: Array
(
    [0] => stdClass Object
        (
            [orderID] => 1191423
            [discountPct] => 0
            [orderType] => BUY
            [price] => 345.1
            [amountOpen] => 1
            [amountFilled] => 0
            [commission] => 0.00
            [volumeFilled] => 0.00
            [orderStatus] => CANCELLED
            [placedAt] => 2011-12-06 18:42:46
            [cancelledAt] => 2011-12-06 18:42:46
            [filledAt] =>
            [instrument] => EUR/SLL
        )
)

Related transactions: Array
(
    [0] => stdClass Object
        (
            [transactionID] => 10478781
            [currency] => SLL
            [amount] => 358.56
            [balance] => 5137.80
            [transactionType] => CANCEL_ORDER
            [exchangeID] => 1191423
            [processedAt] => 2011-12-06 18:42:46
        )

    [1] => stdClass Object
        (
            [transactionID] => 10478775
            [currency] => SLL
            [amount] => -358.56
            [balance] => 4779.24
            [transactionType] => PLACE_ORDER
            [exchangeID] => 1191423
            [processedAt] => 2011-12-06 18:42:46
        )
)

Rate Limit Info: Array
(
    [Limit-Minute] => 60
    [Remaining-Minute] => 48
    [Reset-Minute] => 12
    [Limit-Hour] => 600
    [Remaining-Hour] => 588
    [Reset-Hour] => 72
    [Cost] => 1
)
```