Basic API Reference & Developer Guide

Version 2.3

May 2, 2011

Revision History

Version	Date	Changes
1.0	2008-11-18	Initial version.
1.1	2009-11-09	Added `estimateMarketOrder` method, and everything that comes with it.
2.0	2010-03-08	Added `shortCurrency` parameter in getStatistics. Added `gridName` output parameter in `getTerminalList.` Added `getGridList` method. Added new error codes for completeness. Removed WSDL listing.
2.1	2010-04-14	Added `registered` output parameter of `getGridList.` Added `getGridStatistics` method.
2.2	2010-08-27	Added `wwwURL` output parameter of `getGridList.`
2.3	2011-05-02	Added `getRawTradeData` method.

Table of Contents

1 Overview

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

BASIC API VS. TRANSACTIONAL API

The Basic API is designed for applications that require simple read-only access to the exchange, whereas the more complex Transactional API also allows performing transactions on the exchange.

Typical applications of the Basic API include:

- display of current exchange rates on Web pages or in-world
- currency calculators (Web or in-world) using up-to-date exchange rates
- integration of current exchange rates into in-world items such as vendors or rental boxes
- advanced graphs of historical price and volume
- an alert service that sends an alarm message when a specified exchange rate is reached

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

Access to the Basic API is anonymous and does not require a VirWoX account.

2 Protocol Options

SIMPLE HTTP

Because in the Basic API we only transmit “public” information, all data is simply sent over unencrypted HTTP. In order to make access easy from a wide variety of programming language, we support two alternative protocol options: SOAP and JSON-RPC. While SOAP is the robust standard protocol for interfacing web services, supported by a variety of programming languages, JSON is more lightweight and simple.

2.1 SOAP

SOAP (Simple Object Access Protocol) is a robust standard interface to web services[1]. It is based on XML and can be sent over HTTP and HTTPS, and therefore works well with firewalls. Both request and response are XML documents.

WSDL

The definition of the web service’s methods, types, and messages is typically done using the Web Services Description Language (WSDL)[2], so that it can be automatically

[1] http://www.w3.org/TR/soap12-part1

[2] http://www.w3.org/TR/wsdl

processed. Software development tools for web services based on SOAP and WSDL are available for a wide range of programming languages.

The WSDL service description of the VirWoX Basic API is at:

http://api.virwox.com/api/basic.wsdl

This resource contains all the necessary information you will need to access the service (see the programming examples in Chapter 5). It is recommended to directly read the resource from above URL when starting the connection, rather than from a local copy, so that you always use the most up-to-date version of the protocol.

DIRECT ACCESS

In most programming environments, the WSDL file is all you need to invoke methods. However, if you want to access the SOAP protocol directly, e.g. from a programming environment that does not support WSDL, the access point for the API is at:

http://api.virwox.com/api/soap.php

EXAMPLE

For example, to invoke the `getMarketDepth` method (see Section 3.3), you would POST the following XML document (whitespace added for readability) to the above URL[3]:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
  xmlns:ns1="urn:types"
  xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
  xmlns:SOAP-ENC=”http://schemas.xmlsoap.org/soap/encoding/”
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
  <SOAP-ENV:Body>
    <ns1:getMarketDepth>
      <instruments SOAP-ENC:arrayType="xsd:string[2]"
                   xsi:type="SOAP-ENC:Array">
        <item xsi:type="xsd:string">EUR/SLL</item>
        <item xsi:type="xsd:string">USD/SLL</item>
      </instruments>
      <buyDepth xsi:type="xsd:int">1</buyDepth>
      <sellDepth xsi:type="xsd:int">1</sellDepth>
    </ns1:getMarketDepth>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1="urn:types"
  xmlns:SOAP-ENC=”http://schemas.xmlsoap.org/soap/encoding/”
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:getMarketDepthResponse>
      <result SOAP-ENC:arrayType="SOAP-ENC:Struct[2]"
              xsi:type="SOAP-ENC:Array">
        <item xsi:type="SOAP-ENC:Struct">
          <buy SOAP-ENC:arrayType="SOAP-ENC:Struct[1]"
               xsi:type="SOAP-ENC:Array">
```

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

```
            <item xsi:type="SOAP-ENC:Struct">
              <price xsi:type="xsd:string">371.5</price>
              <volume xsi:type="xsd:string">24</volume>
            </item>
          </buy>
          <sell SOAP-ENC:arrayType="SOAP-ENC:Struct[1]"
                xsi:type="SOAP-ENC:Array">
            <item xsi:type="SOAP-ENC:Struct">
              <price xsi:type="xsd:string">388</price>
              <volume xsi:type="xsd:string">18</volume>
            </item>
          </sell>
          <symbol xsi:type="xsd:string">EUR/SLL</symbol>
          <errorCode xsi:type="xsd:string">OK</errorCode>
          <bestBuyPrice xsi:type="xsd:string">371.5</bestBuyPrice>
          <bestSellPrice xsi:type="xsd:string">388</bestSellPrice>
        </item>
        <item xsi:type="SOAP-ENC:Struct">
          <buy SOAP-ENC:arrayType="SOAP-ENC:Struct[1]"
               xsi:type="SOAP-ENC:Array">
            <item xsi:type="SOAP-ENC:Struct">
              <price xsi:type="xsd:string">277</price>
              <volume xsi:type="xsd:string">448</volume>
            </item>
          </buy>
          <sell SOAP-ENC:arrayType="SOAP-ENC:Struct[1]"
                xsi:type="SOAP-ENC:Array">
            <item xsi:type="SOAP-ENC:Struct">
              <price xsi:type="xsd:string">281</price>
              <volume xsi:type="xsd:string">211</volume>
            </item>
          </sell>
          <symbol xsi:type="xsd:string">USD/SLL</symbol>
          <errorCode xsi:type="xsd:string">OK</errorCode>
          <bestBuyPrice xsi:type="xsd:string">277</bestBuyPrice>
          <bestSellPrice xsi:type="xsd:string">281</bestSellPrice>
        </item>
      </result>
    </ns1:getMarketDepthResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.2 JSON-RPC

JSON[4] (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[5] is a lightweight Remote Procedure Call (RPC) protocol using JSON for object serialization.

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

http://api.virwox.com/api/json.php

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

4 http://www.ietf.org/rfc/rfc4627.txt

5 http://json-rpc.org/wiki/specification

HTTP POST Using HTTP POST[6], 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.

EXAMPLE For example, to invoke the `getMarketDepth` method (see Section 3.3), you would POST the following string (whitespace added for readability):

```
{
  "method": "getMarketDepth",
  "params":
  {
    "symbols": ["EUR\/SLL", "USD\/SLL"],
    "buyDepth": 1,
    "sellDepth": 1
  },
  "id":1
}
```

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

```
{
  "result":
  [
    {
      "buy":
      [
        {
          "price": "370",
          "volume": "165"
        }
      ],
      "sell":
      [
        {
          "price": "387.9",
          "volume": "63"
        }
      ],
      "symbol": "EUR\/SLL",
      "errorCode": "OK",
      "bestBuyPrice": "370",
      "bestSellPrice": "387.9"
    },
    {
```

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

```
      "buy":
      [
        {
          "price": "277",
          "volume": "487"
        }
      ],
      "sell":
      [
        {
          "price": "281",
          "volume" : "149"
        }
      ],
      "symbol": "USD\/SLL",
      "errorCode": "OK",
      "bestBuyPrice": "277",
      "bestSellPrice":"281"
    }
  ],

  "error": null,
  "id":1
}
```

Library functions to produce the request and parse the response into objects are available for most programming languages (see Chapter 5).

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

http://api.virwox.com/api/json.php?method=getMarketDepth&symbols[0]=EUR/SLL&symbols[1]=USD/SLL&buyDepth=1&sellDepth=1&id=1

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

Similarly, you can also POST

```
method=getMarketDepth&symbols[0]=EUR/SLL&symbols[1]=USD/SLL
&buyDepth=1&sellDepth=1&id=1
```

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

3 API Services Reference

This chapter contains descriptions of each of the methods supported by the Basic API. Coding examples using these methods can be found in Chapter 5.

3.1 getInstruments

PURPOSE Use this method to retrieve a list of the tradable instruments on VirWoX. In Forex terminology, an “instrument” is a currency pair, sometimes also called ‘cross’, such as EUR/SLL.

INPUT None.

OUTPUT An array of objects of type `Instrument`; each of which has the following attributes:

Parameter	Type	Description
`symbol`	`string`	A human-readable name of the instrument, e.g. “EUR/SLL” or “VirWoX Stock”.
`longCurrency`	`string`	The name of the tradable unit that you get when you go long (i.e. buy) this instrument.
`shortCurrency`	`string`	The name of the tradable unit that you get when you go short (i.e. sell) this instrument.
`decimals`	`int`	The number of decimal digits (to the right of the decimal point) used for prices in this instrument.
`commissionRate`	`double`	The base commission rate (i.e. without any discounts) for Limit Order for this instrument.
`commissionRateMkt`	`double`	The variable commission rate for Market Orders for this instrument.
`commissionConstMkt`	`double`	The constant commission rate for Market Orders for this instrument.

NOTES Currently, we trade only four real-world currencies (EUR, USD, CHF, GBP) for SLL (Second Life Lindens). Therefore, the `shortCurrency` parameter is currently always “SLL”, and the `decimals` and `commissionRate` parameters are currently set to 1 decimal digit and 0.039 (i.e. 3.9%), respectively, for all these instruments. Likewise, the `commissionRateMkt` is at 0.025 (i.e. 2.5%) and `commissionConstMkt` is 50 (i.e. 50 SLL) for all instruments. See also https://www.virwox.com/help.php#_Commissions_and_Discounts.

However, the system is designed to trade arbitrary commodities (or even stocks), for arbitrary currencies (real or virtual), or against each other. Therefore the developer should not make assumptions about the settings of these parameters, but use this method to determine them if needed.

3.2 getBestPrices

PURPOSE Use this method to quickly retrieve the currently best available prices for one or more instruments.

INPUT An array named `symbols` containing the symbols of the desired instruments.

OUTPUT An array of objects of type `BestPriceItem` for each symbol requested; each of which has the following attributes:

Parameter	Type	Description
`errorCode`	`ErrorEnum`	"NO_SUCH_INSTRUMENT" if this symbol does not exist, or "OK" if it does (Chapter 0).
`symbol`	`string`	The human-readable name of the instrument.
`bestBuyPrice`	`double`	The best (i.e. highest) price somebody is willing to buy this instrument for.
`bestSellPrice`	`double`	The best (i.e. lowest) price somebody is willing to sell this instrument for.

3.3 getMarketDepth

PURPOSE Use this method to retrieve detailed market depth information, i.e. the currently offered prices and volume, for one or more instruments. For each instruments, the method will return a list containing the specified number of price/volume pairs, starting from the currently best available buy and sell prices. For convenience, the method also returns the currently best available buy and sell prices.

INPUT The following parameters:

Parameter	Optional	Type	Description
`instruments`	N	`SymbolList`	An array of symbols (type `string`).
`buyDepth`	Y	`int`	The number of items to return for buy prices (starting at the best buy price). Default 0.
`sellDepth`	Y	`int`	The number of items to return for sell prices (starting at the best sell price). Default 0.

OUTPUT An array of objects of type `MarketDepthItem` for each symbol requested; each of which has the following attributes:

Parameter	Type	Description

`buy`	`PriceVolumeList`	List of the best buy prices and volumes (see below).
`sell`	`PriceVolumeList`	List of the best sell prices and volumes (see below).
`errorCode`	`ErrorEnum`	“NO_SUCH_INSTRUMENT” if this symbol does not exist, or “OK” if it does (Chapter 4).
`symbol`	`string`	The human-readable name of the instrument.
`bestBuyPrice`	`double`	The best (i.e. highest) price somebody is willing to buy this instrument for.
`bestSellPrice`	`double`	The best (i.e. lowest) price somebody is willing to sell this instrument for.

The `PriceVolumeList` is an array of items of type `PriceVolumeItem`; each of which has the following attributes:

Parameter	**Type**	**Description**
`price`	`double`	The price.
`volume`	`int`	The volume at this price, measured in `longCurrency`. Volume in `shortCurrency` can easily be calculated as `price * volume`.

NOTES On the VirWoX website, we display only the volume of the 5 best buy and sell prices offered. With this method, you have access to the full market.

While the `getMarketDepth` method also retrieves the best prices for convenience, the `getBestPrices` method is faster and returns less data, so it should be preferred if you just need the prices and not the available volume.

3.4 estimateMarketOrder

PURPOSE

Use this method estimate the results of a Market Order, taking into account the exchange fees. For a SELL order this method returns the amount of `shortCurrency` that would be gained from selling the specified amount of `longCurrency`. For a BUY order it returns the amount of `shortCurrency` that would be needed to buy the specified amount of `longCurrency`. As orders may be entered and executed any time, the real amount of a subsequently placed Market Order may by slightly different, which is why this is only an estimate.

INPUT

The following parameters:

Parameter	Optional	Type	Description
`orderType`	N	`OrderTypeEnum`	"BUY" or "SELL", with reference to `longCurrency`, i.e. BUY means buying `longCurrency` and selling `shortCurrency`.
`Amount`	N	`int`	Number of units to buy or sell, measured in `longCurrency.`
`Instrument`	N	`string`	The human-readable name of the instrument.

OUTPUT

The following attributes:

Parameter	Type	Description
`errorCode`	`ErrorEnum`	"OK", or one of these Error Codes (Chapter 4): "NO_SUCH_INSTRUMENT" "INVALID_ORDER_TYPE" "INVALID_AMOUNT_OR_PRICE" "INSUFFICIENT_LIQUIDITY"
`Amount`	`Amount`	Amount of shortCurrency necessary to BUY the specified amount of longCurrency, or amount of shortCurrency gained from a SELL of the specified amount of longCurrency. Already including fees.

NOTES

The average exchange rate can be computed by dividing the returned amount of shortCurrency by the specified amount of longCurrency.

3.5 getTradedPriceVolume

PURPOSE This method retrieves historical prices and traded volumes per time interval, for example to create charts.

INPUT The following parameters:

Parameter	Optional	Type	Description
`instrument`	N	`string`	The human-readable name of the instrument.
`startDate`	N	`string`	The beginning of the interval, in the format "YYYY-MM-DD hh:mm:ss".
`endDate`	N	`string`	The end of the interval, in the format "YYYY-MM-DD hh:mm:ss".
`precision`	Y	`int`	The output is grouped by this many digits of the date & time in format "YYYY-MM-DD hh:mm:ss". Meaningful values are: 4 : year 7 : month 10 : day 13 : hour 15 : 10-minutes 16 : minute Default is 10 (day).
`HLOC`	Y	`int`	When set to 1 (true), the method returns High/Low/Open/Close data. Default is 0 (false). See Notes.

OUTPUT The following attributes:

Parameter	Type	Description
`errorCode`	`ErrorEnum`	"NO_SUCH_INSTRUMENT" if this symbol does not exist, or "OK" if it does (Chapter 4).
`priceVolumeList`	`HLOCPriceVolumeList`	Map (associative array) of price & volume data (see below).

The `HLOCPriceVolumeList` is a map (associative array) of items of type `HLOCPriceVolumeItem.` The keys are strings containing the date and time of the interval (**`precision`** digits long). The `HLOCPriceVolumeItem` contains the following data:

Parameter	Type	Description

`longVolume`	`int`	The volume traded in this interval, specified in the instrument's `longCurrency.`
`shortVolume`	`int`	The volume traded in this interval, specified in the instrument's `shortCurrency.`
`high`	`double`	The highest actually traded price in this interval. Only present if `HLOC` is set to true.
`low`	`double`	The lowest actually traded price in this interval. Only present if `HLOC` is set to true.
`open`	`double`	The first actually traded price in this interval. Only present if `HLOC` is set to true.
`close`	`double`	The last actually traded price in this interval. Only present if `HLOC` is set to true. See Notes.

NOTES

Calculating `shortVolume` / `longVolume` yields the average price for the interval.

Depending on the chosen parameters, this method may return a lot of data. Don't select `HLOC` if you do not need it.

For performance reasons, the `close` price per interval is simply set to the open price of the next interval. The last **`close`** price, however, is set to the last actually traded price before the specified `endDate`.

3.6 getRawTradeData

PURPOSE

Use this method to retrieve the raw trade information of recent trades. In contrast to getTradedPriceVolume, trades are not grouped by time interval.

INPUT

The following parameters:

Parameter	Optional	Type	Description
`instrument`	N	`string`	The human-readable name of the instrument.
`timespan`	N	`int`	Only trades which have occurred this many seconds in the past are returned. E.g., 3600 will return the trades of the last hour, 86400 the last day, etc.

OUTPUT

The following attributes:

Parameter	Type	Description
`errorCode`	`ErrorEnum`	"NO_SUCH_INSTRUMENT" if this symbol does not exist, or "OK" if it does (Chapter 4).
`data`	`TradeDataList`	Array of trade data items (see below).

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

Parameter	Type	Description
`time`	`int`	The time of the trade as a UNIX time stamp (i.e. in seconds since Jan.1, 1970).
`price`	`double`	The price at which the trade took place.
`vol`	`int`	The volume of the trade in the `longCurrency` of the instrument. Calculating `price*vol` yields the volume in the `shortCurrency`.

NOTES This call returns the raw trades for each partial fill of an order. It is therefore quite possible to get a number of small trades matched at the same time instead of one large trade.

3.7 getStatistics

PURPOSE This method returns general performance statistics about the VirWoX exchange.

INPUT The following parameter:

Parameter	Optional	Type	Description
`shortCurrency`	Y	`string`	Use this optional parameter to restrict the trading volume to a set of instruments with this currency as `shortCurrency` (currently, only `SLL` or `OMC` make sense as value).

OUTPUT The following attributes:

Parameter	Type	Description
`registeredUsers`	`int`	The number of registered users.
`volumeTotal`	`string`	The all-time volume (in `shortCurrency`) traded by VirWoX.
`volume24hours`	`string`	The total volume (in `shortCurrency`) traded in the last 24 hours.
`volume30days`	`string`	The total volume (in `shortCurrency`) traded in the last 30 days.

NOTES

Currently, the only supported `shortCurrency` is SLL. Therefore, all volumes are SLL amounts.

The trading volumes returned by this method are compensated for trading activity by certain exchange system users. Therefore, this method may return volume figures slightly smaller than `getTradedPriceVolume.`

3.8 getTerminalList

PURPOSE This method returns a list of our in-world terminals.

INPUT None.

OUTPUT An array of items of type `Terminal`:

Parameter	Type	Description
`atmID`	`int`	An internal unique ID of the terminal.
`Public`	`BoolEnum`	'Y' if this terminal should be shown to the user, 'N' if not (e.g. if it is offline).
`Busy`	`BoolEnum`	'Y' if this terminal is busy, i.e. the last request was due to an interaction with an agent.
`Region`	`string`	The region name of the terminal's location.
`X`	`int`	The x coordinate of the terminal's location.
`Y`	`int`	The y coordinate of the terminal's location.
`Z`	`int`	The z coordinate of the terminal's location.
`Language`	`string`	The terminal's default language[7].
`lastRequest`	`string`	Date and time of the last request we received from the terminal.
`nextRequest`	`string`	Date and time of the next request we expect to receive from the terminal.
`Delta`	`int`	Difference (in seconds) between the current time and `nextRequest`. If negative, the request is overdue for this many seconds, and the terminal probably offline.
`gridName`	`string`	Short name of the Grid the terminal is in (see `getGridList` method).

[7] An ISO 639-1 language code, optionally followed by a country code; e.g. en_US

3.9 getGridList

PURPOSE This method returns a list of known grids. It will be sorted by decreasing number of (validated) VirWoX users in this grid, i.e. more popular grids will appear before less popular ones.

INPUT None.

OUTPUT An array of items of type `Grid`:

Parameter	Type	Description
`gridID`	`int`	An internal unique ID of the grid.
`shortName`	`string`	The sort name (nickname) of the grid
`longName`	`string`	The long name (description) of the grid
`loginURL`	`string`	The grid's login URL (unique).
`wwwURL`	`string`	The URL of the website for the grid (if available)
`defaultCurrency`	`string`	The default currency used on the grid (currently, either `SLL` or `OMC`).
`active`	`BoolEnum`	'Y' if this grid has at least one validation terminal.
`registered`	`int`	The number of registered VirWoX users on this grid.
`validated`	`int`	The number of validated VirWoX users on this grid.

3.10 getGridStatistics

PURPOSE This method returns real-time or historic data on the size and performance on the Open Metaverse Economy, i.e. the grids that have adopted the OMC as their virtual currency.

INPUT The following parameters:

Parameter	Optional	Type	Description
`mode`	Y	`string`	One of the following values: `NOW` (=default): return real-time data `HOURLY`: one dataset per hour `DAILY`: one dataset per day
`gridID`	Y	`int`	If `0` (default), return data about the OMC economy as a whole. Else return data on the specified grid only. Use `getGridList`

			to get the `gridID` for a specific grid.
`startDate`	N	`string`	The beginning of the interval, in the format “YYYY-MM-DD hh:mm:ss”.
`endDate`	N	`string`	The end of the interval, in the format “YYYY-MM-DD hh:mm:ss”.

OUTPUT The following attributes:

Parameter	Type	Description
`time`	`string`	The time when this dataset was recorded, or the current server time if `mode=NOW.`
`gridID`	`int`	The grid ID, as specified on the input parameter.
`grids`	`int`	The number of OMC-enabled grids, if `gridID=0`.
`regions`	`int`	The number of OMC-enabled regions.
`avatarsOnline`	`int`	The number of avatars online at `time`.
`avatars24hours`	`int`	The number of unique avatars online in the last 24 hours before `time`.
`avatars30days`	`int`	The number of unique avatars online in the last 30 days before `time`.
`registeredUsers`	`int`	The number of registered VirWoX users on this grid or all OMC-enabled grids if `gridID=0`.
`validatedUsers`	`int`	The number of validated VirWoX users on this grid or all OMC-enabled grids if `gridID=0`.
`u2u24hours`	`Amount`	The amount of OM¢ transferred between users in the specified grid in the last 24 hours before `time`.
`u2u30days`	`Amount`	The amount of OM¢ transferred between users in the specified grid in the last 24 hours before `time`.
`u2uTotal`	`Amount`	The total amount of OM¢ transferred between users in the specified grid.
`circulating`	`Amount`	The amount (in OM¢) held by all users on this grid.

NOTES The data for `gridID=0` (the whole Open Metaverse Economy) are not necessarily the sum of the individual grids. If `gridID=0`, also transactions outside the grids (e.g. on websites) are taken into account.

`startDate` and `endDate` are ignored if `mode=NOW.`

4 Error Codes

ERRORENUM The following error codes are defined in the SOAP data type **`ErrorEnum`**:

Value	Meaning
`OK`	No error.
`INVALID_USERNAME_OR_PASSWORD`	The specified username and/or password is invalid[8].
`NO_TARGET_CUSTOMER`	The recipient does not exist.
`NO_SOURCE_ACCOUNT_FOR_THIS_CURRENCY`	The sender has no account for this currency.
`NO_TARGET_ACCOUNT_FOR_THIS_CURRENCY`	The recipient has no account for this currency.
`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 of 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 account has been disabled.
`LIMIT_EXCEEDED`	The deposit or withdrawal amount exceeds the current limit of the user. 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.

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

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 specified payment token has expired or does not exist.
UNSUPPORTED_PAYMENT_TYPE	An unsupported payment type has been specified.
UNSUPPORTED_PAYMENT_TARGET	An unsupported payment target has been specified.
NO_SUCH_PAYMENT	The specified payment does not exist.
INTERNAL_ERROR	An internal error has occurred.

NOTES Most error codes are not used in the Basic API, but are listed here for consistency and completeness.

5 Programming Examples

This chapter contains a few (small) examples on how to access the VirWoX API in a number of programming languages.

5.1 PHP5

SOAP

The following is a minimalistic example showing how simple it is to retrieve the current best prices in PHP5 using the SOAP interface:

```
<?php
   // open the SOAP client in WSDL mode:
   $virwox = new SoapClient('http://api.virwox.com/api/basic.wsdl');

   // retrieve the best prices for EUR/SLL and USD/SLL:
   $result = $virwox->getBestPrices(array('EUR/SLL','USD/SLL'));

   // output the result for demonstration:
   print_r($result);
?>
```

This will produce an output similar to this:

```
Array
(
    [0] => stdClass Object
        (
            [symbol] => EUR/SLL
            [errorCode] => OK
            [bestBuyPrice] => 340.1
            [bestSellPrice] => 344
        )

    [1] => stdClass Object
        (
            [symbol] => USD/SLL
            [errorCode] => OK
            [bestBuyPrice] => 276.1
            [bestSellPrice] => 280
        )
)
```

JSON-RPC

Now, the same example using JSON-RPC and curl:

```
<?php
  // prepare request object
  $request->method = 'getBestPrices';
  $request->params = array(symbols => array('EUR/SLL','USD/SLL'));
  $request->id = 1;
  $request_string = json_encode($request);

  //set the curl parameters:
  $ch = curl_init();
  curl_setopt($ch, CURLOPT_URL, 'http://api.virwox.com/api/json.php');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
  curl_setopt($ch, CURLOPT_POST, 1);
  curl_setopt($ch, CURLOPT_POSTFIELDS, $request_string);

  //get response from server:
  $response = curl_exec($ch);
  curl_close($ch);

  //parse reponse:
  $result = json_decode($response);

  // print debug output:
  print "Request:\n$request_string\n";
  print "\nResponse:\n";
  print_r($result);
?>
```

This will produce an output similar to this:

```
Request:
{"method":"getBestPrices","params":{"symbols":["EUR\/SLL","USD\/SLL"]},
"id":1}


Response:
stdClass Object
(
    [result] => Array
        (
            [0] => stdClass Object
                (
                    [symbol] => EUR/SLL
                    [errorCode] => OK
                    [bestBuyPrice] => 340.1
                    [bestSellPrice] => 344
                )

            [1] => stdClass Object
                (
                    [symbol] => USD/SLL
                    [errorCode] => OK
                    [bestBuyPrice] => 276.1
                    [bestSellPrice] => 280
                )
        )
    [error] => null
    [id] => 1
)
```

5.2 JavaScript

AJAX AND JSON

This in an example on how to asynchronously retrieve data from VirWoX using JavaScript's `XMLHttpRequest` object. For simplicity, we use the JSON interface so we can directly `eval` the result into a JavaScript object.

The following is a simple but complete HTML page[9] that displays a table with the current best prices for EUR/SLL and USD/SLL. Pressing the “refresh prices” button will update the table.

```
<html>
<head>
<title>Current VirWoX exchange rates</title>

<script LANGUAGE=javascript>
// requires Mozilla, Opera, Safari or Internet Explorer >= 7:
var xmlHttp = new XMLHttpRequest();

// send request:
function getBestPrices() {
  var method = 'getBestPrices';
  var params = '{"symbols":["EUR/SLL","USD/SLL"]}';

  xmlHttp.open('POST', 'http://api.virwox.com/api/json.php', true);
  xmlHttp.onreadystatechange = displayPrices;
  xmlHttp.send('{"method":"'+method+'","params":'+params+',"id":"1"}');
}

// handle response:
function displayPrices() {
  if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
//  alert(xmlHttp.responseText);
    // parse JSON response to a JavaScript object:
    var result = eval('('+xmlHttp.responseText+')').result;

    // modify output table:
    var p1 = document.getElementById('prices').rows[1].cells;
    var p2 = document.getElementById('prices').rows[2].cells;
    p1[0].innerHTML = result[0].symbol;
    p1[1].innerHTML = result[0].bestBuyPrice;
    p1[2].innerHTML = result[0].bestSellPrice;
    p2[0].innerHTML = result[1].symbol;
    p2[1].innerHTML = result[1].bestBuyPrice;
    p2[2].innerHTML = result[1].bestSellPrice;
  }
};
</script>
</head>

<body>
<input type="button" onclick="getBestPrices() "value="refresh prices"/>

<table id='prices' border='1'>
  <tr><th>Symbol</th><th>Bid</th><th>Ask</th></tr>
  <tr><td>-</td><td>-</td><td>-</td></tr>
  <tr><td>-</td><td>-</td><td>-</td></tr>
</table>
</body>
</html>
```

[9] Available at http://api.virwox.com/api/ajaxdemo.html

5.3 LSL

ACCESSING VIRWOX DIRECTLY FROM SECOND LIFE

The following is an example on how to access the VirWoX server from the Linden Scripting Language (LSL), i.e. directly from an object in the virtual world “Second Life”. We use the JSON interface as JSON can be efficiently converted to an LSL list.

INTERFACE SCRIPT

An LSL script has only 16 KB of data available, including the byte-code of the script itself. Therefore, it is good practice to encapsulate functions in scripts of their own, which are attached to linked prims and accept commands via link messages. The following script is designed to be put into a linked prim. It accepts link messages with commands (below we implement only the GET_RATE command) and parameters, calls the corresponding VirWoX API method, and asynchronously returns a formatted response to the caller:

```
// Interface to VirWoX Basic API
// -----------------------------
//
// - waits for link message with currency pair passed as the string
//   parameter
// - sends VirWoX Basic API request to retrieve current price
// - sends average price back to sender of link message as a formatted
     number (f.2)
//

// constants:
string SERVER = "http://api.virwox.com/api/json.php";
list HTTP_OPTS = [HTTP_METHOD, "POST",
                  HTTP_MIMETYPE,"application/x-www-form-urlencoded" ];

// commands:
integer GET_RATE = 1;      // get current rate

// we use the simple url-encoded API, simulating a form
string REQUEST_RATE    = "&method=getBestPrices&symbols[0]=";

// remove these separators in parsing the JSON response:
list JSON_SEPARATORS = [",", ":", "{", "}", "[", "]", "\""];

// global variables:
integer return_to;     // link number to return data to

default
{
  link_message(integer sender, integer cmd, string s, key k)
  {
    return_to = sender;   // remember where to send the answer to

    if (cmd == GET_RATE)
      llHTTPRequest(SERVER, HTTP_OPTS, "id=" + (string)cmd +
                    REQUEST_RATE + s);
  }

  // process response from server.
  // Parse returned JSON string into a flat list of strings
  http_response(key id, integer status, list meta, string response)
  {
    // parse JSON response into a flat list, removing all unnecessary
    // characters
    list l = llParseString2List(response, JSON_SEPARATORS, []);

    integer id = llList2Integer(l, -1);   // last element is always ID
    string json_error = llList2String(l, -3);  // JSON error code

    if (json_error == "null")     // OK
    {
      if (id == GET_RATE)
      {
        if (llList2String(l, 4) == "OK")
        {
          // calculate average rate in a way that eliminates these LSL
          // rounding errors:
          integer ratex100 = (integer)((llList2Float(l, 6) +
                                 llList2Float(l, 8)) * 50);
          string rate = (string)(ratex100/100) + "." +
                        llGetSubString((string)ratex100,-2,-1);

          llMessageLinked(return_to, id, rate, NULL_KEY);  // return
        }
        else        // unsupported instrument or the like; return error
          llMessageLinked(return_to, 0, llList2String(l, 4), NULL_KEY);
      }
      else
        llMessageLinked(return_to, 0, "unknown command", NULL_KEY);
    }
    else   // JSON error
      llMessageLinked(return_to, 0, json_error, NULL_KEY);
  }
}
```

USING THE API

The following LSL code snippet shows how to call the method from another script:

```
integer GET_RATE = 1;      // get current rate
integer API_PRIM = 2;      // linked prim #2 has our API code

// this function sends a request to the API prim
// it returns nothing; a link_message will return the answer
// from the API prim
getCurrentPrice()
{
  llMessageLinked(API_PRIM, GET_RATE, "EUR/SLL", NULL_KEY);
}

...

default
{
  ...
  link_message(integer sender, integer cmd, string response, key k)
  {
    if (sender == API_PRIM)    // answer for our request
      if (cmd == GET_RATE)     // data received OK from GET_RATE
      {
        rate = (float)response;
        ...                    // do something with it
      }
  }
  ...
}
```