MicroPayment API
Reference & Developer Guide

Version 2.1

May 30, 2012

Revision History

Version	Date	Changes
1.0	2011-09-29	Initial version.
1.1	2011-12-06	A number of small editorial corrections.
2.0	2012-04-27	Payment targets fully implemented.
2.1	2012-05-30	Added Section on the Payment Authorization Page (including how to earn commission for currency conversion during payments).

Table of Contents

1 Overview

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

APPLICATIONS

The MicroPayment API gives developers access to the payment functionality implemented by VirWoX. While this API has been designed for payments in virtual worlds such as the OpenSim Hypergrid, in can in fact be used for all kind of payments and is not limited to the OpenSim environment or even virtual worlds in general.

Typical applications of the Micropayment API are:

- Accepting (virtual) money from VirWoX users for the purchase of (virtual) goods and services, either in-world or on a website.
- Sending (virtual) money to VirWoX users or directly to avatars or other payment systems.
- Accepting payments in real-world currency from a variety of payment providers, and automatic conversion to virtual currency.

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 MicroPayment API you will need a VirWoX account, and an application key (see Chapter 2).

1.1 The Payment Process

A TYPICAL SCENARIO

In a typical scenario, a merchant wants to accept money from a buyer, and delivers something in return. An overview of the payment process for this case is shown in Figure 1, and described in detail below:

1. The buyer starts the payment process, e.g. on the merchant’s website, or by touching an in-world vending machine of the merchant.
2. The merchant now calls the `requestPayment` method (see section 4.1) to the API. The payment request stores various pieces of information about the payment, some of it public (i.e. to be displayed to the buyer), but also some private information intended to uniquely identify the payment to the merchant.
3. The payment request is identified by a unique payment `token`, which is returned to the merchant.
4. The merchant redirects the buyer to the payment authorization page on the VirWoX website, which is:

   https://www.virwox.com/pay?token=<payment-token>

   where <payment-token> is the result of the `requestPayment` call.

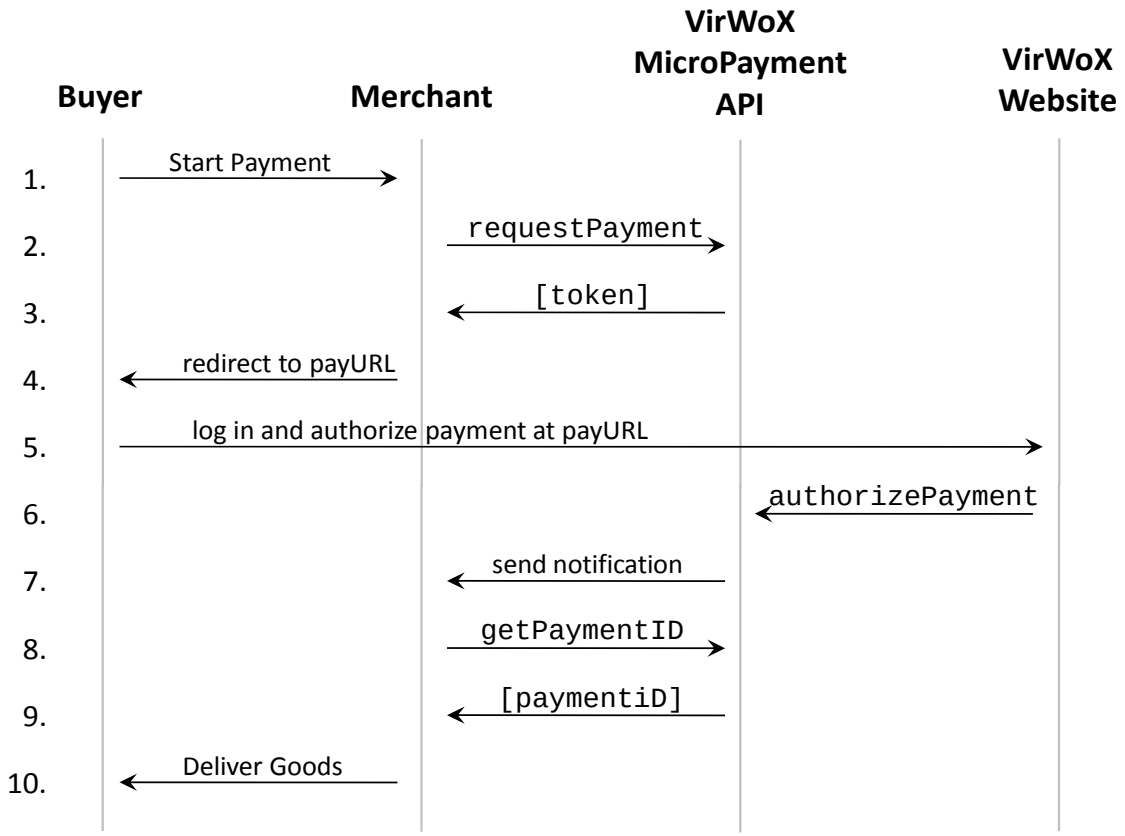

Figure 1: Overview of the payment process (see text)

5. The buyer logs in on this page (with username and password) and authorizes the payment[1]. In doing so, the user will be shown the public information specified in the `requestPayment` call in step 3.
6. The code on the VirWoX website will issue an `authorizePayment` call to the API (using the username and password supplied by the buyer). This performs the payment. If no `notifyURL` was specified on the `requestPayment` call in step 3, the process ends here.
7. If a `notifyURL` was specified on the `requestPayment` call in step 3, the system sends a notification message with the payment status to the specified URL (see section 0).
8. **IMPORTANT SECURITY CONSIDERATION**: The merchant should not yet deliver the goods, as such a notification message could be faked by a third party knowing or guessing the `notifyURL`. Instead, the merchant calls the `getPaymentID` method of the API to verify the status of the payment.
9. The API answers with the `paymentID` and `status` of the payment, if the payment has in fact been authorized, or an error message if not.

[1] Alternatively, the buyer may directly pay using a third-party payment provider, in which case no login is required.

10. If the payment has been made and the status is `OK`, the merchant should make sure that this `paymentID` has not yet been delivered, and if so, deliver the goods to the buyer.

SENDING MONEY

In many cases, the process described above may be simplified considerably. For example, let us assume we want to send money to a user. In this scenario, we only need these steps:

1. We send a `requestPayment` call (see section 4.1) to the API. In the parameters of this call we specify the intended recipient. We do not need to specify a `notifyURL`.
2. The API responds with a `token`.
3. With this `token` and our own username and password, we send an `authorizePayment` call to the API. This performs the payment.

   IMPORTANT SECURITY CONSIDERATION: Note that this requires storing our own username and password somewhere in the code that performs the payments (assuming that these payments should be performed automatically without human intervention). You need to make sure that nobody else has access to the code with this data!

REQUEST A PAYMENT

We can also request money from others by generating a payment request (`requestPayment`), and sending them the resulting pay URL as specified above (step 4) in the first scenario, e.g. by email. The payment requests (tokens) are valid for 1 day. As soon as the other user pays, we will get notified via the `notifyURL`.

SEND MONEY TO THIRD PARTY ACCOUNTS

Using the `targetType` and `withdrawTo` parameters of the `requestPayment` method (see section 4.1), it is possible to automatically forward the funds to other accounts, e.g. to send Linden Dollars to Second Life avatars. The recipient does not even need a VirWoX account in this case.

CURRENCY CONVERSION

The payment authorization page (step 5) allows the buyer to pay not only from the buyer's VirWoX account, but also directly from a range of third-party payment providers. In this case, a currency conversion may take place and is performed automatically by the VirWoX exchange. Note that the buyer does not need a VirWoX account in this case. See also Section 1.2.

1.2 The Payment Authorization Page

By default, the payment authorization page (see step 4 of the payment process described above) offers the buyer the options to pay from a VirWoX account (requires login), or directly pay from an external payment provider (such as Credit Card, PayPal, paysafecard, etc.).

EARNING COMMISSION FROM PAYMENTS

If the buyer pays using an external payment provider, there is a behind-the-scenes conversion of the source currency (i.e. the one the buyer pays in) to the target currency. Currently this feature is only supported if the target currency of the payment is `SLL`, `ACD`, or `OMC`. In this case, VirWoX charges the buyer a conversion fee, and it is possible for the developer to participate in VirWoX's fee for the conversion. The commission schedule is the same as for the "Currency Shop" (i.e. depending on volume), as documented at https://www.virwox.com/currency_shop.php.

The following optional parameters control this feature. The need to be appended to the URL where the buyer is redirected to (see example below):

`pk` Partner Key. This specifies the where the commission payment is sent to. You can use the VirWoX username (requires that the recipient of the commission has a VirWoX account), or the avatar name if you want to keep the VirWoX username secret (requires a validated avatar in the VirWoX account). Alternatively, you can set the UUID of a Second Life avatar. In this case, the commission will be sent directly to Second Life (requires the target currency to be `SLL`). No VirWoX account is required in this case. However, it is recommended that you send the commission to a VirWoX account, so that you can later see the payment details in the VirWoX payment history.

`rp` Referrer Page. Use this parameter to keep track of your commissions. The contents of this parameter will be stored as "tracking ID" of the commission payment. If omitted, and the `pk` parameter is set, the URL of the page referring the user to the payment page will be used.

`mode` If this parameter is set to the value `direct_only`, the buyer can only pay using an external payment source, and the user interface is somewhat simplified.

`provider` Most useful if `mode=direct_only`, you can pre-select a certain external payment provider. Currently, supported values are `CreditCard`, `PayPal`, `paysafecard`, `Sofortbanking`, `Skrill`, `Moneybookers`, `NETELLER`. Default value is `PayPal`. The user may still choose a different payment method.

`currency` Most useful if `mode=direct_only`, you can pre-select a certain currency. Currently, supported values are `EUR`, `USD`, `GBP`, `CHF`. However, not all values are available for all payment providers. The default value is determined by the user's country as determined from the IP address. The user may still choose a different currency.

EXAMPLE

This would be an example of a payment URL with added parameters:

https://www.virwox.com/pay/?token=<payment_token>
&mode=direct_only&pk=VirWoX.Inventor@SL&provider=paysafecard

1.3 Notifications

Notifications are HTTP requests sent to a `notifyURL` which is a parameter of the `requestPayment` call. They are sent whenever a payment changes its status (see section 1.4).

RETRIES

The system will keep trying to deliver the notification (in increasing time intervals) until the HTTP server responds with an HTTP status of 200 (OK) and some data (do not send an empty document). It will give up after about 1 day or 28 retries. If the payment changes its status (see Section 1.4) in the meantime, it will stop trying to send the notifications for the old status, and instead start trying to deliver the notifications for the new status.

FORMAT OF THE NOTIFYURL

The `notifyURL` can be up to 1023 characters long, and has the following format:

```
[PROTOCOL] [http|https]://host[:port]/path
```

Where [`PROTOCOL`] is empty or one off the following:

`GET`	The HTTP request is sent as a GET request to the URL specified. The parameters (see below) are appended to the URL. This is the default behaviour if no `PROTOCOL` is specified.
`POST`	The HTTP request is sent as a POST request to the specified URL. The parameters (see below) are sent as POST parameters.
`XMLRPC`	The HTTP request is sent as a POST request to the specified URL. The parameters (see below) are sent as XML-RPC request[2] in the message body.

PARAMETERS SENT

The following parameters are sent on the notification (so that the recipient knows which payment it belongs to):

`paymentID`	The `paymentID` of the payment, as it appears in the payment history on the VirWoX website.
`token`	The original `token` of the payment request, so the recipient knows which payment request the notification belongs to
`status`	The `status` of the payment (see section 1.4).

In the case of a GET request, the parameters are appended to the URL, i.e. the string

```
?paymentID=...&token=...&status=...
```

will be appended to the `notifyURL`, with ... replaced by the actual value of the parameter. If the `notifyURL` already contains a '?', the string above will be appended starting with a '&' instead of the '?'. This allows the recipient to specify additional parameters to the `notifyURL`.

IMPORTANT SECURITY CONSIDERATION

The notification can be faked by a third party that knows or guesses the `notifyURL`, and therefore cannot be trusted. The recipient MUST call back the VirWoX server using the `getPaymentID` method (as described in section 1.1, step 8) to make sure it is a genuine notification!

Also, a third party knowing the `notifyURL` of a valid but old transaction could replay this notification. So, the recipient of the notification must keep track of already processed payments to avoid processing the same payment more than once.

[2] http://en.wikipedia.org/wiki/XML-RPC

1.4 Payment Status

PAYMENT STATUS VALUES

The following payment status values are defined:

- `OK`: The payment has been authorized; the funds have been transferred from the sender's account to the recipient's account. Payments in this state cannot be cancelled, i.e. payments cannot be reversed once they have reached the `OK` state.
- `UNCLAIMED`: The payment has been authorized, but the recipient does not (yet) have a VirWoX account. The funds have been deducted from the sender's account. It the recipient registers within 30 days, the funds will be released to the recipient (and the status will change to `OK`), if not they will be sent back to the sender and the status will change to `CANCELLED`. The sender may also cancel the payment before the 30 days have passed using the `cancelPayment` method.
- `HOLD`: The payment has been authorized, but put on temporary hold for security screening. The funds have been deducted from the sender's account. Only an administrator can either release the payment (i.e. send the funds to the recipient) or cancel it (send funds back to the sender).
- `ESCROW`: The payment has been authorized, but the sender requested it to be escrowed, i.e. the funds have been removed from the sender and sent to a special escrow account, until either the sender releases them to the recipient (using the `releasePayment` method), e.g. after the goods have been delivered, or the recipient cancels the payment (sending funds back to the sender).
- `CANCELLED`: The payment has been authorized but later cancelled (e.g. from status `UNCLAIMED`).

IMPORTANT SECURITY CONSIDERATION

Note that notifications are sent on every status change of a payment. It is therefore important to verify that the payment has reached status `OK` before acting on it (e.g. delivering goods to the buyer).

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, and use

https://www.virwox.com:8000/pay?token=<payment-token>

as the payment authorization page in step 4 of the payment process.

You can 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 MicroPayment 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[3] (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[4] 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/payment.php

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

HTTP POST

Using HTTP POST[5], 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 `getPaymentStatus` method (see Section 4.5), you would POST the following string (whitespace added for readability):

```
{
  "method" : "getPaymentStatus",
  "params" :
  {
    "token" : "a33056ca-1fed-102d-937c-db625a9d0339"
  },
  "id" : 1234
}
```

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

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

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

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

```
{
  "result":
  {
    "errorCode" : "NO_SUCH_PAYMENT"
  },
  "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/payment.php?method=getPaymentStatus&token=a33056ca-1fed-102d-937c-db625a9d0339&key=<your-app-key>

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

Similarly, you can also POST

```
method=getPaymentStatus&token=a33056ca-1fed-102d-937c-
db625a9d0339&key=<your-app-key>
```

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 your 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 `requestPayment` 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

This chapter contains descriptions of each of the methods supported by the MicroPayment API. Coding examples using these methods can be found in Chapter 6.

4.1 requestPayment

PURPOSE

Use this method to start a payment process as described in Chapter 1. This method will usually be invoked by the recipient of a payment, but can also be called by the sender. In fact it can be called by anybody. The system will check the supplied parameters. If everything is OK, a *payment request* will be generated and stored. This *payment request* is identified by a token, which needs to be communicated to the sender of the payment, in order to authorize the payment. The payment request and the associated token will become invalid after a predetermined period of time.

INPUT

Parameter	Optional	Type	Description
`recipientName`	N	`string`	The VirWoX username, or a fully qualified avatar name (i.e. `firstname.lastname@gridname`) of the recipient, or the UUID of the recipient's avatar. The UUID will only work if the recipient has already validated the avatar with VirWoX.
`amount`	N	`Amount`	The amount the recipient will get paid.
`currency`	N	`string`	The currency the recipient will get paid in. Currently, the following currencies are supported: `SLL`, `ACD`, `OMC`, `BTC`, `EUR`, `USD`, `GBP`, and `CHF`.
`description`	Y	`string`	A human-readable description of the reason for this payment. It will be displayed to the sender when authorizing the payment. This parameter needs to be URL-encoded.
`paymentType`	Y	`string`	The type of payment (see notes below). Default: `TRANSFER`.
`regionCode`	Y	`int`	The ID of the region the payment was made in. Defaults to 0.
`agentName`	Y	`string`	The fully qualified avatar name (i.e. `firstname.lastname@gridname`) of the agent (avatar) on whose behalf the payment is made (e.g. who should

			receive the item that is being paid for). This allows paying for a number of avatars with a single VirWoX account. This parameter needs to be URL-encoded.
trackingID	Y	string	An ID the recipient can use to identify this payment. Will not be shown to the sender. This parameter needs to be URL-encoded.
targetType	Y	string	See notes below. The default value is ACCOUNT.
withdrawTo	Y	string	See notes below. This parameter needs to be URL-encoded.
notifyURL	Y	string	The URL used for payment notification (see Chapter 1).
returnURL	Y	string	If specified, the user will be redirected to this page after the payment. The same parameters as with the notifyURL will be automatically appended to this URL.

OUTPUT

Parameter	Type	Description
errorCode	ErrorEnum	OK, or one of these error codes (Chapter 5): INVALID_AMOUNT_OR_PRICE NO_SOURCE_ACCOUNT_FOR_THIS_CURRENCY NO_TARGET_ACCOUNT_FOR_THIS_CURRENCY UNSUPPORTED_PAYMENT_TARGET NO_TARGET_CUSTOMER ACCOUNT_DISABLED
token	string	If errorCode is OK, this attribute contains the token associated with the generated payment request (to be used in other API methods).

NOTES

Currently, the following payment types are defined (for the purpose of easy filtering of payments according to type):

- TRANSFER (default): A general transfer of money between accounts.
- PAY_OBJECT: pay money to an in-world object
- BUY_OBJECT: buy an in-world object.
- BUY_LAND: buy virtual land.
- OBJECT_PAYS: an in-world object pays money to a user.
- GIFT: an avatar sends virtual currency to another avatar.
- GROUP_CREATION_FEE: fee for creating a group.
- UPLOAD_FEE: fee for uploading an asset.

- `BUY_CURRENCY`: buy virtual currency for real currency.
- `COMMISSION`: commission paid to a user.
- `OTHER`: any other reason.

Currently, these payment target types are supported:

- `ACCOUNT`: pay to the VirWoX account of the recipient
- `SL_AVATAR`: immediately forward the paid amount from the recipient's VirWoX account to a specified Second Life avatar. This requires `currency` to be set to `SLL`, and the `withdrawTo` parameter needs to be set to the avatar's unique key (which can be found using the name2Key method, see section 4.8).
- `AVN_AVATAR`: immediately forward the paid amount from the recipient's VirWoX account to a specified Avination avatar. This requires `currency` to be set to `ACD`, and the `withdrawTo` parameter needs to be set to the avatar's unique key (which can be found using the name2Key method, see section 4.8).
- `BITCOIN`: immediately forward the paid amount from the recipient's VirWoX account to a specified Bitcoin address. This requires `currency` to be set to `BTC`, and the `withdrawTo` parameter needs to be set to a valid Bitcoin address.
- `PAYPAL`: immediately forward the paid amount from the recipient's VirWoX account to a specified PayPal account. This requires `currency` to be set to `EUR`, `USD`, `GBP`, or `CHF`, and the `withdrawTo` parameter needs to be set to a valid PayPal account (E-Mail address).
- `MONEYBOOKERS`: immediately forward the paid amount from the recipient's VirWoX account to a specified Skrill (Monebookers) address. This requires `currency` to be set to `EUR`, `USD`, or `GBP`, and the `withdrawTo` parameter needs to be set to a valid Skrill/Moneybookers account (E-Mail address).
- `NETELLER`: immediately forward the paid amount from the recipient's VirWoX account to a specified PayPal account. This requires `currency` to be set to `EUR`, `USD`, or `GBP`, and the `withdrawTo` parameter needs to be set to a valid NETELLER account (12 digits).
- `BANK`: immediately forward the paid amount from the recipient's VirWoX account to a specified bank account. This requires `currency` to be set to `EUR`, `USD`, `GBP`, or `CHF`, and the `withdrawTo` parameter needs to be set to a bank account number in the format `IBAN:BIC`.

4.2 getPaymentRequest

PURPOSE

This method can be used to retrieve the information associated with a payment request token, e.g. to display it to the user. Only "public" information is returned by this method (not the `notifyURL` of the recipient, for example).

INPUT

Parameter	Optional	Type	Description
`token`	N	`string`	The payment request ID returned by the `getPaymentRequest` method.

OUTPUT

Parameter	Type	Description
`errorCode`	`ErrorEnum`	`ERR_TOKEN_EXPIRED` if this token does not exist, or `OK` if it does.
`amount`	`Amount`	The amount the recipient will get paid.
`currency`	`string`	The currency the recipient will get paid in.
`recipientName`	`string`	A human-readable name of the recipient.
`paymentType`	`string`	The type of payment (see `requestPayment`)
`regionCode`	`int`	The region code (see `requestPayment`)
`regionName`	`String`	A human-readable region name (if available)
`gridID`	`Int`	The ID of the grid the region is in (if available)
`agentName`	`string`	Name of the avatar on whose behalf the payment is made (see `requestPayment`)
`description`	`string`	A human-readable description of the reason for this payment.
`createdAt`	`string`	The time this payment request was created, in ISO 8601 format.
`expiresAt`	`string`	The time this payment request was created, in ISO 8601 format.

4.3 cancelPaymentRequest

PURPOSE

Use this method to cancel a payment before it expires, so that it cannot be authorized any more. Useful if the sender cannot fulfill the promise to deliver something in exchange.

INPUT

Parameter	Optional	Type	Description
`token`	`N`	`string`	The payment request ID returned by the `getPaymentRequest` method.

OUTPUT

Parameter	Type	Description
`errorCode`	`ErrorEnum`	`ERR_TOKEN_EXPIRED` if this token does not exist, or `OK` if it does.

4.4 authorizePayment

PURPOSE

This method authorizes a payment. The sender of the payment has to identify herself with username and password. This method will often not be invoked directly by an application. Instead, the sender is directed to the payment authorization page (see Chapter 1), which will indirectly call this method.

INPUT

Parameter	Optional	Type	Description
username	N	string	The VirWoX username of the sender of the payment.
password	N	string	The password associated with username.
token	N	string	The payment request ID returned by the getPaymentRequest method.
note	Y	string	An optional note from the payer to the recipient (url-encoded).

OUTPUT

Parameter	Type	Description
errorCode	ErrorEnum	OK, or one of these error codes (Chapter 5): INVALID_USERNAME_OR_PASSWORD ACCOUNT_DISABLED TOKEN_EXPIRED NO_TARGET_CUSTOMER NO_SOURCE_ACCOUNT_FOR_THIS_CURRENCY NO_TARGET_ACCOUNT_FOR_THIS_CURRENCY INVALID_AMOUNT_OR_PRICE INSUFFICIENT_FUNDS DATABASE_TIMEOUT
withdrawalErrorCode	ErrorEnum	If targetType is not ACCOUNT, the system will try to forward the payment to the specified payment provider (see notes at requestPayment), which may fail. In this case, this attribute contains one of the following error codes (Chapter 5): ACCOUNT_DISABLED ILLEGAL_PARAMETER MANUAL_INTERVENTION_REQUIRED
paymentID	integer	The paymentID of the payment (only if errorCode is OK). Sender and Recipient can use this information to find the payment in their transaction and payment lists on the VirWoX web interface.

NOTE

This method is repeatable, i.e. if the payment with the same token has already been performed before, it will not be made again. In this case, this method will just return the paymentID of the already-executed payment; notifications will not be re-sent. This

behavior enables the application to simply re-invoke the method in case of a communication problem, until an answer is received.

4.5 getPaymentStatus

PURPOSE

This method should be used to verify that a payment (identified by a payment request token) has been performed.

The receiver should not trust the notifications sent to the `notifyURL` and the recipient's email address, as they can be faked. It is therefore important to use this method when receiving such a notification to make sure that the payment really has been made!

INPUT

Parameter	Optional	Type	Description
`token`	`N`	`string`	The payment request ID returned by the `getPaymentRequest` method.

OUTPUT

Parameter	Type	Description
`status`	`string`	The status of the payment (see section 1.4). Only if the status is `OK`, the payment has been fully processed. If the payment has not yet been made at all, `NO_SUCH_PAYMENT` is returned.
`paymentID`	`integer`	If the payment has already been made, this is its `paymentID`. Sender and Recipient can use this information to find the payment in their transaction and payment lists on the VirWoX web interface.

4.6 cancelPayment

PURPOSE

This method can be used to cancel a payment (i.e. send the funds back to the sender), depending on the payment status (see section 1.4):

`UNCLAIMED`: Only the sender may cancel the payment.

`ESCROW`: Only the recipient may cancel the payment.

If the payment is in a different status, it cannot be cancelled by a user.

INPUT

Parameter	Optional	Type	Description
`username`	`N`	`string`	The VirWoX username of the sender of the

			payment.
password	N	string	The password associated with username.
paymentID	N	string	The payment ID returned by the getPaymentStatus method.

OUTPUT

Parameter	Type	Description
errorCode	ErrorEnum	OK, or one of these error codes (Chapter 5): INVALID_USERNAME_OR_PASSWORD ACCOUNT_DISABLED DATABASE_TIMEOUT NO_SUCH_PAYMENT

4.7 releasePayment

PURPOSE

This method can be used to release a payment (i.e. send the funds to the recipient), depending on the payment status (see section 1.4):

ESCROW: Only the sender may cancel the payment.

If the payment is in a different status, it cannot be cancelled by a user.

Payments of status UNCLAIMED are automatically released once the recipient registers a VirWoX account.

INPUT

Parameter	Optional	Type	Description
username	N	string	The VirWoX username of the sender of the payment.
password	N	string	The password associated with username.
paymentID	N	string	The payment ID returned by the getPaymentStatus method.

OUTPUT

Parameter	Type	Description
errorCode	ErrorEnum	OK, or one of these error codes (Chapter 5): INVALID_USERNAME_OR_PASSWORD ACCOUNT_DISABLED DATABASE_TIMEOUT NO_SUCH_PAYMENT ILLEGAL_PARAMETER MANUAL_INTERVENTION_REQUIRED

4.8 name2Key

PURPOSE

Use this method to find an avatars UUID (also known as a key) from the avatar's name. This is required for the `withdrawTo` parameter on the `requestPayment` method, among other uses.

This method consults the extensive avatar database of VirWoX, and only if the name cannot be found there performs a (slower) search on the grid itself. Currently this method works only for Second Life and Avination.

INPUT

Parameter	Optional	Type	Description
`firstName`	`N`	`string`	The first name of the avatar.
`lastName`	`N`	`string`	The last name of the avatar.
`gridName`	`Y`	`string`	The grid name. Currently only `SL` and `AVN` are supported. If not specified, `SL` is default.

OUTPUT

Parameter	Type	Description
`key`	`string`	The avatars key (a `UUID`). If the avatar does not exist, the NULL key (`00000000-0000-0000-0000-000000000000`) is returned.
`birthday`	`string`	The avatars birthday, if known, or `0000-00-00` if not.
`name`	`string`	The avatars name, with the correct upper/lowercase.

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[6].
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 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 specified VirWoX account has been disabled.
LIMIT_EXCEEDED	The deposit or withdrawal amount exceeds the current limit of the user.

[6] 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 MicroPayment API, but are listed here for consistency and completeness.

6 Programming Examples

This chapter contains a few (small) examples on how to access the MicroPayment API. For the time being, only PHP examples are supplied.

6.1 PHP Class VirWOXPaymentAPI

The following is a generic PHP class interface to the MicroPayment 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.

PAYMENTAPI.PHP

```
<?php

/*****************************************************************************
 * Class VirWOXPaymentAPI
 *
 * 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.
 *
 * The user should be redirected to VirWoXPaymentAPI::paymentURL for authorization.
 */

define('VIRWOX_PAY_API_URL', 'https://www.virwox.com/api/payment.php');

class VirWoXAPIException extends Exception {}

class VirWoXPaymentAPI
{
  public  $paymentURL;            // the payment URL (users are redirected here)
  private $id = 1;                // id of the JSON RPC calls (incremented on each call)
  private $key;
  private $debug;

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

    if ($key == 'TEST')
      $this->paymentURL = 'https://www.virwox.com:8000/pay/';    // test system
    else
      $this->paymentURL = 'https://www.virwox.com/pay/';         // production system
  }

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

    $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_PAY_API_URL, false, $context);
    if ($response === false)
```

```
      throw new VirWoXAPIException('Could not connect to VirWoX MicroPayment API');

    $JSON = json_decode($response);

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

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

    return $JSON->result;
  }
}
?>
```

6.2 Sending a Payment

This is a simple example of how to send money to another user using the class above.

SEND_PAYMENT_DEMO.PHP

```
<?php
/*****************************************************************************
 * Mimimalistic demo program that shows how to make a payment to another VirWoX account
 *
 * Warning: Script contains the VirWoX username and password of the sending account.
 *          Keep this script in a safe place !!!
 *****************************************************************************/

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

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

define('PAYER_USERNAME', 'your_username');
define('PAYER_PASSWORD', 'your_password');

try
{
  $api = new VirWoXPaymentAPI(KEY, true); // turn debug on/off

  // make a payment request:
  $request = array('recipientName' => 'demo.user@SL',     // where should the money go to
                   'amount'        => 10,
                   'currency'      => 'OMC',
                   'description'   => 'API demo payment', // human-readable description of
                                                          // the reason for payment
                   'paymentType'   => 'GIFT');            // see API documentation for list
                                                          // of supported types
  $result = $api->requestPayment($request);
  if ($result->errorCode == 'OK')
  {
    $token =  $result->token;             // remember token

    // authorize the payment:
    $result = $api->authorizePayment(array('username' => PAYER_USERNAME,
                                           'password' => PAYER_PASSWORD,
                                           'token'    => $token));

    if ($result->errorCode == 'OK')
    {
      $status = $api->getPaymentStatus(array('token' => $token));
      print "The payment has been sent with payment ID {$status->paymentID}. "
           ."Status = {$status->status}\n";
    }
    else
      print "Could not authorize payment. Error code = {$result->errorCode}\n";
  }
  else
    print "Could not create payment request. Error code = {$result->errorCode}\n";
}
catch (Exception $e)
{
  print "Exception: ".$e->getMessage();
}
?>
```

6.3 Receiving a Payment

Receiving money is a bit more complex than sending, and requires (at least) two scripts: The first script generates the payment request and sends the user to the payment authorization page (steps 1 to 5 of the payment process described in Section 1.1), and the second script receives the payment notification and acts on it (steps 7 to 10).

The second part can either be done by a script called by a notification (see Section 0), or on the page that the user is sent to after the payment authorization (as specified by the `returnURL` parameter of the `requestPayment` method), or a combination of that. The following example uses the `returnURL` method.

GET_PAYMENT_DEMO.PHP

```
<?php
/*******************************************************************************************
 * Demo program that shows how to accept a payment for an object (e.g. in a web shop)
 *
 * Upon completion (successful or not), the user will be redirected to RETURN_URL
 *******************************************************************************************/

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

define('KEY', 'your_app_key');        // use 'TEST' for the test system
define('NOTIFY_URL', '');
define('RETURN_URL', 'http://www.example.com/completed_payment_demo.php');

session_start();

try
{
  $api = new VirWoXPaymentAPI(KEY);

  // make a payment request
  $request = array('recipientName' => 'demo.user@SL',   // where should the money go to
                   'amount'        => 10,               // price
                   'currency'      => 'OMC',            // currency (SLL, ACD, or OMC)
                   'description'   => 'Super Widget',   // human-readable description
                                                        // of the item you sell
                   'paymentType'   => 'BUY_OBJECT',     // buy an object
                   'regionCode'    => 0,                // region code (0 if not inworld)
                   'agentName'     => '',               // object will be sent to this
                                                        // user instead of payer (optional)
                   'trackingID'    => '',               // tracking ID (only visible for
                                                        // the recipient)
                   'targetType'    => 'ACCOUNT',        // send money to VirWoX account
                   'withdrawTo'    => '',               // leave empty for type ACCOUNT
                   'notifyURL'     => NOTIFY_URL,
                   'returnURL'     => RETURN_URL);

  $result = $api->requestPayment($request);
  if ($result->errorCode == 'OK')
  {
    // in a real application, we would now save the token and order data in a database so
    // that we know which order was associated with the token, and can deliver the right
    // things when the payment gets confirmed to the notify URL.
    //
    // In this demo, the page at the return URL will process the payment. We remember the
    // current order data in the PHP session. This is much simpler, but means that the item
    // can only be shipped automatically if the user's browser actually returns to the
    // return URL after the payment has been processed and while the PHP session is still
    // active.

    $_SESSION[$result->token] = $request;

    header("Location: $api->paymentURL?token=$result->token");  // redirect user to payment
                                                                // authorization page
  }
  else
    print "Could not create payment request. Error code = {$result->errorCode}\n";
}
catch (Exception $e)
{
  print "Exception: ".$e->getMessage();
}
?>
```

COMPLETED_PAYMENT_DEMO.PHP

```
<?php
/*******************************************************************************************
 * Demo program that shows how to receive a payment (e.g. in a web shop).
 *
 * The user's browser will be redirected here after successful payment.
 *******************************************************************************************/

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

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

$token = $_REQUEST['token'];
$paymentID = $_REQUEST['paymentID'];
$status = $_REQUEST['status'];

session_start();

try
{
  $api = new VirWoXPaymentAPI(KEY);
  $paymentStatus = $api->getPaymentStatus(array('token' => $token));

  if ($paymentStatus->status == 'OK' && $paymentStatus->paymentID == $paymentID)
  {
    // IMPORTANT: in a real application, we must make sure that the token is really one of
    // our payments, and that it has not yet been processed already (e.g. look it up in a
    // database), before shipping the item.
    // Also, we would prefer shiping the item from the notify URL, not the return URL
    // because sometimes the browser does not get here.
    //
    // In this simple demo we just use the data stored in the PHP session.
    // We delete it from the session once we have shipped it, so that the page cannot be
    // re-played.

    if ($paymentrequest = $_SESSION[$token])
    {
      print "Congratulations! You have just bought a {$paymentrequest['description']}\n";
      // now ship it....
      unset($_SESSION[$token]);
    }
    else
      print "This payment has already been processed. ID = {$paymentStatus->paymentID}\n";
  }
  else
    print "Payment Status = {$paymentStatus->status}\n";
}
catch (Exception $e)
{
  print "Exception: ".$e->getMessage();
}
?>
```