PAYMILL API/V2 DOCUMENTATION ¶

To get a foreseeable and resource-oriented function call we have implemented our API with REST. All response objects will be delivered as JSON objects.

For an easy switch from test to live mode PAYMILL supports test keys and live keys. The test key works in the exact same way as the live key, but doesn't do live credit card transactions. You can always use the test key even if you have activated the live key for your staging server.

The examples shown at the API can be used directly to be implemented in your code or if it is curl you can directly call it in the terminal. Your own test key is already used at the examples.

Check our API on:

apiary.io | mashape

API Endpoint

https://api.paymill.com/v2/

Authentication ¶

To authenticate at the Paymill API, you need the private key of your test or live account. You have to use http basic access authentication. Your key has to be set as the username. A password isn't required and you don't have to insert one. But if you want, feel free to insert an arbitrary string.

Note

- Please keep your private keys secure and don't pass them to anybody. These private keys have extreme secure information for handling the transactions of your shop.
- All your requests must be made via https. Requests which will be made in another way will fail. This is for security reasons of the submitted data.

Example

CURL

```
% curl https://api.paymill.com/v2/clients \
    -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$apiKey = '<YOUR_PRIVATE_KEY>';
$request = new Paymill\Request($apiKey);
```

JAVA

```
PaymillContext paymillContext = new PaymillContext(
    "<YOUR_PRIVATE_KEY>"
);
ClientService clientService = paymillContext.getClientService();
```

NODE.JS

```
var paymill = require('paymill-node')('<YOUR_PRIVATE_KEY>');
```

PYTHON

```
import paymill
paymill.api_key = "<YOUR_PRIVATE_KEY>"
```

RUBY

```
# Ruby >= 1.9.x
```

```
require 'paymill'
Paymill.api_key = "<YOUR_PRIVATE_KEY>"
```

.NET

```
PaymillContext paymillContext = new PaymillContext("<YOUR_PRIVATE_KEY
>");
ClientService clientService = paymillContext.getClientService();
```

JS

```
var pm = require('../paymill.node.js');
pm.initialize("<YOUR_PRIVATE_KEY>");
```

Response Codes ¶

Some JSON objects like transactions or refunds include a response code, which specifies more detailed information about the outcome of a preceding request.

The codes are numeric and have 5 digits, the first digit follows the rules of http codes so something like 1xxxx is informational (request received etc.), 2xxxx indicates a successful transaction whereas 4xxxx or 5xxxx are error codes.

Response Codes you will receive:

10001: General undefined response.
10002: Still waiting on something.

20000: General success response.

40000: General problem with data.
40001: General problem with payment data.
40100: Problem with credit card data.
40101: Problem with cvv.
40102: Card expired or not yet valid.
40103: Limit exceeded.
40104: Card invalid.
40105: Expiry date not valid.
40106: Credit card brand required.
40200: Problem with bank account data.
40201: Bank account data combination mismatch.
40202: User authentication failed.
40300: Problem with 3d secure data.
40301: Currency / amount mismatch
40400: Problem with input data.
40401: Amount too low or zero.
40402: Usage field too long.
40403: Currency not allowed.

50000: General problem with backend.
50001: Country blacklisted.
50002: IP address blacklisted.
50003: Anonymous IP proxy used.
50100: Technical error with credit card.
50101: Error limit exceeded.
50102: Card declined by authorization system.
50103: Manipulation or stolen card.
50104: Card restricted.
50105: Invalid card configuration data.
50200: Technical error with bank account.
50201: Card blacklisted.

50300: Technical error with 3D secure.

50400: Decline because of risk issues.

50401: Checksum was wrong.

50402: Bank account number was invalid (formal check).

50403: Technical error with risk check.

50404: Unknown error with risk check.

50405: Unknown bank code.

50406: Open chargeback.

50407: Historical chargeback.

50408: Institution / public bank account (NCA).

50409: KUNO/Fraud.

50410: Personal Account Protection (PAP).

50500: General timeout.

50501: Timeout on side of the acquirer.

50502: Risk management transaction timeout.

50600: Duplicate transaction.

Errors ¶

We've build a RESTful API - that's the reason why we are concerned about correct status codes which are returned as JSON objects. But in some cases we don't have the same syntax as the normal http response has. The basic status codes are:

- **2xx** indicates a successful request
- **4xx** informs you about an error
- **5xx** tells you that we did something wrong

Note

Do not just check the HTTP status code 2xx to verify a successful request, also check the expecting message information, for example transactions or refunds include a response code.

HTTP Status Codes we use

200 OK
Great, go ahead.

401 Unauthorized
Jim, You have to provide your private API Key.

403 Transaction Error
Transaction could not be completed, please check your payment data.

404 Not Found
There is no entity with this identifier, did you use the right one?

412 Precondition Failed
I guess you're missing at least one required parameter?

5xx Server Error
Doh, we did something wrong :/

Listviews ¶

We have many listviews for different entities in the API functions. The functionality of these listviews is mainly the same; they only differ in the selectable attributes.

Sort Entries ¶

The JSON response objects can be sorted the way you have requested. In this case you receive the result sorted in the required way to get the result sorted in ascending (**`[attributename]_asc`**) or descending (**`[attributename]_desc`**) order.

Note

Example: amount: **`?order=amount`** | **`?order=amount_asc`** | **`?order=amount_desc`**

Filter Entries ¶

The JSON response objects can be filtered by their attributes. In this case you can call the API to get the result filtered in the required way. This means that the result objects which don't fit the filter aren't delivered.

Note

Example: **`?created_at=<timestamp>`** | **`?created_at=<timestamp (from)>-<timestamp (to)>`**

Payments ¶

The Payment object represents a payment with a credit card or via direct debit. It is used for several function calls (e.g. transactions, subscriptions, clients, ...). To be PCI compliant these information is encoded by our Paymill PSP. You only get in touch with safe data (token) and needn't care about the security problematic of informations like credit card data.

`Payment` Object for credit card payments ¶

Attributes ¶

Example

```
{
    "id"           : "pay_3af44644dd6d25c820a8",
    "type"         : "creditcard",
    "client"       : null,
    "card_type"    : "visa",
    "country"      : null,
    "expire_month" : "10",
    "expire_year"  : "2013",
```

id : string
Unique identifier for this credit card payment
type : enum(creditcard,debit)
client : string or null
The identifier of a client (client-object)
card_type : string
Card type eg. visa, mastercard
country : string or null
Country
expire_month : string
Expiry month of the credit card
expire_year : string
Expiry year of the credit card
card_holder : string
Name of the card holder
last4 : string
The last four digits of the credit card
created_at : integer
Unix-Timestamp for the creation date
updated_at : integer
Unix-Timestamp for the last update
app_id : string or null
App (ID) that created this payment or null if created by yourself.
is_recurring : boolean
The payment is recurring (can be used more than once).
is_usable_for_p.. boolean
The payment is usable for preauthorization.


```
    "card_holder"  : "",
    "last4"        : "1111",
    "created_at"   : 1349942085,
    "updated_at"   : 1349942085,
    "app_id"       : null,
    "is_recurring" : true,
    "is_usable_for_preauthorization" : true
}
```


Payment Object for direct debit payments ¶

Attributes ¶

id : string
Unique identifier for this direct debit payment
type : enum(creditcard,debit)
client : string or null
The identifier of a client (client-object)
code : string
The used Bank Code
account : string
The used account number, for security reasons the number is masked
holder : string
Name of the account holder
created_at : integer
Unix-Timestamp for the creation date
updated_at : integer
Unix-Timestamp for the last update
app_id : string or null
App (ID) that created this payment or null if created by yourself.

Example


```
{
    "id"           : "pay_917018675b21ca03c4fb",
    "type"         : "debit",
    "client"       : null,
    "code"         : "12345678",
    "holder"       : "Max Mustermann",
    "account"      : "******2345",
    "created_at"   : 1349944973,
    "updated_at"   : 1349944973,
    "app_id"       : null,
    "is_recurring" : true,
    "is_usable_for_preauthorization" : true
}
```

is_recurring : boolean
The payment is recurring (can be used more than once).
is_usable_for_p.. boolean
The payment is usable for preauthorization.

Payment Object for direct debit payments (sepa) ¶

Attributes ¶

id : string
Unique identifier for this direct debit payment
type : enum(creditcard,debit)
client : string or null
The identifier of a client (client-object)
code : string
The used Bank Code
account : string
The used account number, for security reasons the number is masked
holder : string
Name of the account holder
iban : string
International Bank Account Number
bic : string
Business Identifier Code
created_at : integer
Unix-Timestamp for the creation date
updated_at : integer
Unix-Timestamp for the last update
app_id : string or null
App (ID) that created this payment or null if created by yourself.
is_recurring : boolean
The payment is recurring (can be used more than once).
is_usable_for_p.. boolean
The payment is usable for preauthorization.

Create new **Credit Card Payment** with ... ¶

Example


```
{
    "id"            : "pay_917018675b21ca03c4fb",
    "type"          : "debit",
    "client"        : null,
    "code"          : "12345678",
    "holder"        : "Max Mustermann",
    "account"       : "******2345",
    "iban"          :"DE1250010517******9890",
    "bic"           :"BENEDEPPYYY",
    "created_at"    : 1349944973,
    "updated_at"    : 1349944973,
    "app_id"        : null,
    "is_recurring" : true,
    "is_usable_for_preauthorization" : true
}
```


Token

Request

CURL

```
curl https://api.paymill.com/v2/payments \
```

Attributes ¶

token : string
Unique credit card token
client : string or null
Unique client identifier

Creates a credit card payment from a given token, if you're providing the **client**-property, the payment will be created and subsequently be added to the client.

Note

- You always need a token to create a new credit card payment.

```
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=098f6bcd4621d373cade4e832627b4f6"
```

PHP

```
$payment = new Paymill\Models\Request\Payment();
$payment->setToken('098f6bcd4621d373cade4e832627b4f6');

$response = $request->create($payment);
```

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
Payment payment = paymentService.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.create(
    {
     token: '098f6bcd4621d373cade4e832627b4f6',
 },
    function(err, payment) {
        if (err) {
            console.log("Couldn't create the payment record");
            return;
        }
        console.log("payment id " + payment.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
payment = p.newcard(
  token='098f6bcd4621d373cade4e832627b4f6'
)
```

RUBY

```
Paymill::Payment.create token: "098f6bcd4621d373cade4e832627b4f6"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.CreateWithTokenAsync("098f6bcd4621d3
73cade4e832627b4f6").Result;
```

JS

```
pm.payments.create("098f6bcd4621d373cade4e832627b4f6").then(function(
payment) {
 console.log("payment:" + payment.id);
}, function(error) {
 console.log("couldnt create payment:" + error);
});
```

Response

```
{
    "data" : {
        "id"            : "pay_3af44644dd6d25c820a8",
        "type"          : "creditcard",
        "client"        : null,
```

```
        "card_type"    : "visa",
        "country"      : null,
        "expire_month" : "10",
        "expire_year"  : "2013",
        "card_holder"  : "",
        "last4"        : "1111",
        "created_at"   : 1349942085,
        "updated_at"   : 1349942085,
        "app_id"       : null,
        "is_recurring" : true,
        "is_usable_for_preauthorization" : true
    },
    "mode" : "test"
}
```


Token & Client

Request

CURL

```
curl https://api.paymill.com/v2/payments \
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=098f6bcd4621d373cade4e832627b4f6" \
  -d "client=client_88a388d9dd48f86c3136"
```

PHP

```
$payment = new Paymill\Models\Request\Payment();
$payment->setToken('098f6bcd4621d373cade4e832627b4f6')
        ->setClient('client_88a388d9dd48f86c3136');

$response = $request->create($payment);
```

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
Payment payment = paymentService.createWithTokenAndClient(
    "098f6bcd4621d373cade4e832627b4f6",
    "client_88a388d9dd48f86c3136"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.create(
    {
     token: '098f6bcd4621d373cade4e832627b4f6',
        client: 'client_88a388d9dd48f86c3136'
 },
    function(err, payment) {
        if (err) {
            console.log("Couldn't create the payment record");
            return;
        }
        console.log("payment id " + payment.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
payment = p.newcard(
  token='098f6bcd4621d373cade4e832627b4f6',
  client='client_88a388d9dd48f86c3136'
)
```

RUBY

```
Paymill::Payment.create token: "098f6bcd4621d373cade4e832627b4f6",
    client: "client_88a388d9dd48f86c3136"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.CreateWithTokenAndClientAsync(
    "098f6bcd4621d373cade4e832627b4f6",
    "client_88a388d9dd48f86c3136"
).Result;
```

JS

```
pm.payments.create("098f6bcd4621d373cade4e832627b4f6", "client_88a388
d9dd48f86c3136").then(function(payment) {
 console.log("payment:" + payment.id);
}, function(error) {
 console.log("couldnt create payment:" + error);
});
```

Response

```
{
    "data" : {
        "id"            : "pay_3af44644dd6d25c820a9",
        "type"          : "creditcard",
        "client"        : "client_88a388d9dd48f86c3136",
        "card_type"     : "visa",
        "country"       : null,
        "expire_month"  : "10",
        "expire_year"   : "2013",
        "card_holder"   : "",
        "last4"         : "1111",
        "created_at"    : 1349942085,
        "updated_at"    : 1349942085,
        "app_id"        : null,
        "is_recurring"  : true,
        "is_usable_for_preauthorization" : true
    },
    "mode" : "test"
}
```

Create new **`Debit Payment`** with ... ¶

Token

Request

CURL

```
curl https://api.paymill.com/v2/payments \
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=12a46bcd462sd3r3care4e8336ssb4f5"
```

Attributes ¶

token	:	string Unique direct debit token
client	:	string or null Unique client identifier

PHP

```
$payment = new Paymill\Models\Request\Payment();
$payment->setToken('12a46bcd462sd3r3care4e8336ssb4f5');

$response = $request->create($payment);
```

Creates a direct debit payment from a given token, if you're providing the **`client`**-property, the payment will

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
```

be created and subsequently be added to the client.

```
Payment payment = paymentService.createWithToken(
    "12a46bcd462sd3r3care4e8336ssb4f5"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.create(
    {
     token: '12a46bcd462sd3r3care4e8336ssb4f5'
 },
    function(err, payment) {
        if (err) {
            console.log("Couldn't create the payment record");
            return;
        }
        console.log("payment id " + payment.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
payment = p.newcard(
  token='12a46bcd462sd3r3care4e8336ssb4f5'
)
```

RUBY

```
Paymill::Payment.create token: "12a46bcd462sd3r3care4e8336ssb4f5"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.CreateWithTokenAsync(
    "12a46bcd462sd3r3care4e8336ssb4f5"
).Result;
```

JS

```
pm.payments.create("12a46bcd462sd3r3care4e8336ssb4f5").then(function(
payment) {
 console.log("payment:" + payment.id);
}, function(error) {
 console.log("couldnt create payment:" + error);
});
```

Response


```
{
    "data" : {
        "id"            : "pay_917018675b21ca03c4fb",
        "type"          : "debit",
        "client"        : null,
        "code"          : "12345678",
        "holder"        : "Max Mustermann",
        "account"       : "******2345",
        "created_at"    : 1349944973,
        "updated_at"    : 1349944973,
        "app_id"        : null,
        "is_recurring"  : true,
        "is_usable_for_preauthorization" : true
    },
    "mode" : "test"
}
```


Token & Client

Request

CURL

```
curl https://api.paymill.com/v2/payments \
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=12a46bcd462sd3r3care4e8336ssb4f5" \
  -d "client=client_88a388d9dd48f86c3136"
```

PHP

```
$payment = new Paymill\Models\Request\Payment();
$payment->setToken('12a46bcd462sd3r3care4e8336ssb4f5');
$payment->setClient('client_88a388d9dd48f86c3136');

$response = $request->create($payment);
```

JAVA

```
 PaymentService paymentService = paymillContext.getPaymentService();
 Payment payment = paymentService.createWithTokenAndClient(
    "12a46bcd462sd3r3care4e8336ssb4f5",
    "client_88a388d9dd48f86c3136"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.create(
    {
     token: '12a46bcd462sd3r3care4e8336ssb4f5',
     client: 'client_88a388d9dd48f86c3136'
 },
    function(err, payment) {
        if (err) {
            console.log("Couldn't create the payment record");
            return;
        }
        console.log("payment id " + payment.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
payment = p.newcard(
  token='12a46bcd462sd3r3care4e8336ssb4f5',
  client='client_88a388d9dd48f86c3136'
)
```

RUBY

```
Paymill::Payment.create token: "12a46bcd462sd3r3care4e8336ssb4f5",
    client: "client_88a388d9dd48f86c3136"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
 Payment payment = paymentService.CreateWithTokenAndClientAsync(
    "12a46bcd462sd3r3care4e8336ssb4f5",
    "client_88a388d9dd48f86c3136"
).Result;
```

JS

```
pm.payments.create("12a46bcd462sd3r3care4e8336ssb4f5", "client_88a388
d9dd48f86c3136").then(function(payment) {
```

```
 console.log("payment:" + payment.id);
}, function(error) {
 console.log("couldnt create payment:" + error);
});
```

Response


```
{
    "data" : {
        "id"                : "pay_917018675b21ca03c4fc",
        "type"              : "debit",
        "client"            : "client_88a388d9dd48f86c3136",
        "code"              : "12345678",
        "holder"            : "Max Mustermann",
        "account"           : "******2345",
        "created_at"        : 1349944973,
        "updated_at"        : 1349944973,
        "app_id"            : null,
        "is_recurring"      : true,
        "is_usable_for_preauthorization" : true
    },
    "mode" : "test"
}
```


Payment Details ¶

Returns data of a specific payment.

Request

CURL

```
curl https://api.paymill.com/v2/payments/pay_3af44644dd6d25c820a8 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$payment = new Paymill\Models\Request\Payment();
$payment->setId('pay_3af44644dd6d25c820a8');

$response = $request->getOne($payment);
```

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
Payment payment = paymentService.get("pay_3af44644dd6d25c820a8");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.details('pay_3af44644dd6d25c820a8',
    function(err, payment) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("payment id " + payment.data.id);
    }
);
```

PYTHON

Attributes ¶

id : string
Unique identifier for the payment

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
payment = p.getcarddetails(cardid='pay_3af44644dd6d25c820a8')
```

RUBY

```
Paymill::Payment.find "pay_3af44644dd6d25c820a8"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.GetAsync("pay_3af44644dd6d25c820a8")
.Result;
```

JS

```
pm.payments.detail("pay_3af44644dd6d25c820a8").then(function(payment)
{
    console.log("payment:" + payment.id);
}, function(error) {
    console.log("couldnt get payment:" + error);
});
```

Response


```
{
    "data" : {
        "id"            : "pay_3af44644dd6d25c820a8",
        "type"          : "creditcard",
        "client"        : null,
        "card_type"     : "visa",
        "country"       : null,
        "expire_month"  : "10",
        "expire_year"   : "2013",
        "card_holder"   : "",
        "last4"         : "1111",
        "created_at"    : 1349942085,
        "updated_at"    : 1349942085,
        "app_id"        : null,
        "is_recurring"  : true,
        "is_usable_for_preauthorization" : true
    },
    "mode" : "test"
}
```


List **Payments** ¶

This function returns a JSON object with a list of payments. In which order this list is returned depends on the optional parameter order:

- **count**
- **offset**
- **created_at**

Available filters:

- **card_type=<card_type>**
- **created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>**
- **type=creditcard | debit**

Request

CURL

```
curl https://api.paymill.com/v2/payments \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$payment = new Paymill\Models\Request\Payment();

$response = $request->getAll($payment);
```

JAVA

Available status for **card_type**:

- **visa**
- **mastercard**
- **maestro**
- **amex**
- **jcb**
- **diners**
- **discover**
- **china_union_pay**
- **unknown (= other not supported brand)**

```
PaymentService paymentService = paymillContext.getPaymentService();
PaymillList<Payment> payments = paymentService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.list({},
    function(err, payment) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("payment data " + payment.data);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
payments = p.getcards()
```

RUBY

```
Paymill::Payment.all
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
PaymillList<Payment> payments = paymentService.ListAsync().Result;
```

JS

```
pm.payments.list().then(function(pmlist) {
 console.log(pmlist.items.length + " payments from total of " + pmlis
t.count);
}, function(error) {
 console.log("couldnt list payments:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id"           : "pay_3af44644dd6d25c820a8",
            "type"         : "creditcard",
            "client"       : null,
            "card_type"    : "visa",
            "country"      : null,
            "expire_month" : "10",
            "expire_year"  : "2013",
            "card_holder"  : "",
            "last4"        : "1111",
            "created_at"   : 1349942085,
            "updated_at"   : 1349942085,
            "app_id"       : null,
            "is_recurring" : true,
            "is_usable_for_preauthorization" : true
        }
    ],
    "data_count" : "1",
    "mode" : "test"
}
```

Remove **Payment** ¶

Deletes the specified payment.

Attributes ¶

id : string
Unique identifier for the payment

Request

CURL

```
curl https://api.paymill.com/v2/payments/pay_3af44644dd6d25c820a8 \
  -u <YOUR_PRIVATE_KEY>: \
  -X DELETE
```

PHP

```
$payment = new Paymill\Models\Request\Payment();
$payment->setId('pay_3af44644dd6d25c820a8');

$response = $request->delete($payment);
```

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
paymentService.delete("pay_3af44644dd6d25c820a8");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.payments.remove('pay_88a388d9dd48f86c3136',
    function(err, payment) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("deleted the payment");
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
response = p.delcard(cardid='pay_3af44644dd6d25c820a8')
```

RUBY

```
Paymill::Payment.delete "pay_8203c63949dfa4d8809aa6d3"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
paymentService.DeleteAsync("pay_3af44644dd6d25c820a8").Result;
```

JS

```
pm.payments.remove("pay_3af44644dd6d25c820a8").then(function(payment)
{
 console.log("payment deleted:" + payment.id);
}, function(error) {
 console.log("couldnt remove payment:" + error);
});
```

Response

```
{
    "data":[

    ],
    "mode" : "test"
}
```

Export `Payment` List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of clients. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `card_type`
- `created_at`
- `type`
- `updated_at`

Available filters:

- `card_type`
- `created_at`
- `type`
- `updated_at`

Request

CURL

```
curl https://api.paymill.com/v2/payments \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response

```
"id";"type";"client";"card_type";"country";"bin";"expire_month";"expi
re_year";"card_holder";"last4";"updated_at";"created_at";"app_id";"is
_recurring";"is_usable_for_preauthorization"
"pay_2311e5a076ab0b9c2cdb0399";"creditcard";"client_33c8f8c13d759d00b
144";"visa";"DE";"411111";"2";"2016";"test card holder";"1111";"13424
27064";"1342427064";"";"1";"1"
```

Preauthorizations ¶

If you'd like to reserve some money from the client's credit card but you'd also like to execute the transaction itself a bit later, then use preauthorizations. This is NOT possible with direct debit.

A preauthorization is valid for 7 days.

Preauthorization Object ¶

Attributes ¶

- id : string
  Unique identifier of this preauthorization
- description : string or null
  Description for this preauthorization (max. 255 chars)
- amount : string
  Formatted amount which will be reserved for further transactions
- status : enum(open, pending, closed, failed, deleted, preauth)
  Indicates the current status of this preauthorization
- livemode : boolean
  Whether this preauthorization was issued while being in live mode or not
- payment : payment object for credit card or null
- client : client object or null
- created_at : integer
  Unix-Timestamp for the creation date
- updated_at : integer
  Unix-Timestamp for the last update
- app_id : string or null
  App (ID) that created this preauthorization or null if created by yourself.

Example

```
{
    "id" : "preauth_0b771c503680c341548e",
    "description" : null,
    "amount" : "4200",
    "currency" : "EUR",
    "status" : "closed",
    "livemode" : false,
    "created_at" : 1349950324,
    "updated_at" : 1349950324,
    "payment" : "<Object>",
    "client" : "<Object>",
    "app_id" : null
}
```

Sub objects

- preauthorization.payment returns a payment object for credit card
- preauthorization.client returns a client object

Create new **Preauthorization** with ... ¶

Use either a **token** or an existing **payment** to authorize the given **amount**.

Notice: The response is a transaction object with the preauthorization as sub object.

Token

Request

CURL

```
curl https://api.paymill.com/v2/preauthorizations \
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=098f6bcd4621d373cade4e832627b4f6" \
```

Attributes ¶

amount : integer (>0)
Amount (in cents) which will be charged

currency : string
ISO 4217 formatted currency code

token : either token or payment
string
The identifier of a token

payment : either token or payment
string
The identifier of a payment (only creditcard-object)

description : string or null
Description for this preauthorization (max. 255 chars)

```
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "description=description example"
```

PHP

```
$preAuth = new Paymill\Models\Request\Preauthorization();
$preAuth->setToken('098f6bcd4621d373cade4e832627b4f6')
        ->setAmount(4200)
        ->setCurrency('EUR')
        ->setDescription('description example');

$response = $request->create($preAuth);
```

JAVA

```
PreauthorizationService preauthorizationService = paymillContext.getP
reauthorizationService();
Transaction transaction = preauthorizationService.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "description example"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.preauthorizations.create(
    {
        token: '098f6bcd4621d373cade4e832627b4f6',
        amount: 4200,
        currency: 'EUR',
        description: 'description example'
    },
    function(err, preauthorization) {
        if (err) {
            console.log("Couldn't create the preauthorization record"
);
            return;
        }
        console.log("preauthorization id " + preauthorization.data.id
);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
token = '098f6bcd4621d373cade4e832627b4f6'
p = pymill.Pymill(private_key)
preauth = p.preauth(
    amount=100,
    currency="EUR",
    description="description example",
    token=token
)
```

RUBY

```
# please note: adding a description is not implemented yet
Paymill::Preauthorization.create token: "098f6bcd4621d373cade4e832627
b4f6",
    amount: 4200, currency: "EUR"
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Preau
thorizationService;
```

```
Transaction transaction = preauthorizationService.CreateWithTokenAsyn
c(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "description example"
).Result;
```

JS

```
pm.preauthorizations.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "description example"
).then(function(preauth) {
    console.log("preauth:" + preauth.id);
}, function(error) {
    console.log("couldnt create preauth:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_de120f6f57e7cfc40833",
        "amount" : 4200,
        "origin_amount" : 4200,
        "currency" : "EUR",
        "status" : "preauth",
        "description" : "description example",
        "livemode" : false,
        "refunds" : null,
        "created_at" : 1350324120,
        "updated_at" : 1350324120,
        "response_code": 20000,
        "short_id" : "7357.7357.7357",
        "is_fraud" : false,
        "fees" : [],
        "invoices": [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```


Sub objects

- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Payment

Request

CURL

```
curl https://api.paymill.com/v2/preauthorizations \
  -u <YOUR_PRIVATE_KEY>: \
  -d "payment=pay_d43cf0ee969d9847512b" \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "description=description example"
```

PHP

```
$preAuth = new Paymill\Models\Request\Preauthorization();
$preAuth->setPayment('pay_d43cf0ee969d9847512b')
        ->setAmount(4200)
```

```
        ->setCurrency('EUR')
        ->setDescription('description example');

$response = $request->create($preAuth);
```

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
Payment payment = paymentService.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6"
);
PreauthorizationService preauthorizationService = paymillContext.getP
reauthorizationService();
Transaction transaction = preauthorizationService.createWithPayment(
    payment,
    4200,
    "EUR",
    "description example"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.preauthorizations.create(
    {
        payment: 'pay_d43cf0ee969d9847512b',
        amount: 4200,
        currency: 'EUR',
        description: 'description example'
    },
    function(err, preauthorization) {
        if (err) {
            console.log("Couldn't create the preauthorization record"
);
            return;
        }
        console.log("preauthorization id " + preauthorization.data.id
);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
preauth = p.preauth(
    amount=100,
    currency='EUR',
    description='description example',
    payment='pay_d43cf0ee969d9847512b'
)
```

RUBY

```
# please note: adding a description is not implemented yet
Paymill::Preauthorization.create payment: "pay_d43cf0ee969d9847512b",
    amount: 4200, currency: "EUR"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.CreateWithTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6"
).Result;
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;
Transaction transaction = preauthorizationService.CreateWithPaymentAs
ync(
    payment,
    4200,
```

```
    "EUR",
    "description example"
).Result;
```

JS

```
pm.preauthorizations.createWithPayment(
    "pay_d43cf0ee969d9847512b",
    4200,
    "EUR",
    "description example"
).then(function(preauth) {
    console.log("preauth:" + preauth.id);
}, function(error) {
    console.log("couldnt create preauth:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_616bc3b1f36cc4323be8",
        "amount" : 4200,
        "origin_amount" : 4200,
        "currency" : "EUR",
        "status" : "preauth",
        "description" : "description example",
        "livemode" : false,
        "refunds" : null,
        "created_at" : 1349948920,
        "updated_at" : 1349948920,
        "response_code": 20000,
        "short_id" : "7357.7357.7357",
        "is_fraud" : false,
        "fees" : [],
        "invoices": [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```


Sub objects

- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Preauthorization Details ¶

Returns data of a specific preauthorization.

Request

CURL

```
curl https://api.paymill.com/v2/preauthorizations/preauth_31eb90495837447f76b7 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

Attributes ¶

id : string
Unique identifier of this preauthorization

```php
$preAuth = new Paymill\Models\Request\Preauthorization();
$preAuth->setId('preauth_31eb90495837447f76b7');

$response = $request->getOne($preAuth);
```

JAVA

```java
PreauthorizationService preauthorizationService = paymillContext.getP
reauthorizationService();
Preauthorization preauthorization = preauthorizationService.get(
    "preauth_31eb90495837447f76b7"
);
```

NODE.JS

```node
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.preauthorizations.details('preauth_31eb90495837447f76b7',
    function(err, preauthorization) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("preauthorization id " + preauthorization.data.id
);
    }
);
```

PYTHON

```python
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
preauth = p.getpreauthdetails(
    preid='preauth_31eb90495837447f76b7'
)
```

RUBY

```ruby
Paymill::Preauthorization.find "preauth_31eb90495837447f76b7"
```

.NET

```dotnet
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;
Preauthorization preauthorization = preauthorizationService.GetAsync(
    "preauth_31eb90495837447f76b7"
).Result;
```

JS

```javascript
pm.preauthorizations.detail("preauth_31eb90495837447f76b7").then(func
tion(preauth) {
 console.log("preauth:" + preauth.id);
}, function(error) {
 console.log("couldnt get preauths:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "preauth_31eb90495837447f76b7",
        "description" : "description example",
        "amount" : "4200",
        "currency" : "EUR",
        "status" : "closed",
        "livemode" : false,
        "created_at" : 1349948920,
```

```
        "updated_at" : 1349948920,
        "payment" : "<Object>",
        "client" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```

Sub objects

- preauthorization.payment returns a payment object for credit card
- preauthorization.client returns a client object

Remove `Preauthorizations` ¶

This function deletes a preauthorization.

Attributes ¶

id : string
Unique identifier for the preauthorization

Request

CURL

```
curl https://api.paymill.com/v2/preauthorizations/preauth_31eb90495837447f76b7 \
  -u <YOUR_PRIVATE_KEY>: \
  -X DELETE
```

PHP

```
$preAuth = new Paymill\Models\Request\Preauthorization();
$preAuth->setId('preauth_31eb90495837447f76b7');

$response = $request->delete($preAuth);
```

JAVA

```
PreauthorizationService preauthorizationService = paymillContext.getPreauthorizationService();
preauthorizationService.delete( "preauth_31eb90495837447f76b7" );
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.preauthorizations.remove('preauth_88a388d9dd48f86c3136',
    function(err, preauthorization) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("deleted the preauthorization");
    }
);
```

PYTHON

```
# Not implemented yet
```

RUBY

```
Paymill::Preauthorization.delete "preauth_78ddcedb546909304d43"
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Preauthorizationservice;
preauthorizationService.DeleteAsync( "preauth_31eb90495837447f76b7" ).Result;
```

JS

```
/* Not implemented yet */
```

Response

```
{
    "data":[

    ],
    "mode" : "test"
}
```

List `Preauthorizations` ¶

This function returns a JSON object with a list of preauthorizations. In which order this list is returned depends on the optional parameter order:

- `count`
- `offset`
- `created_at`

Available filters:

- `client=<client id>`
- `payment=<payment id>`
- `amount=[>|<]<integer>` e.g. “300” or with prefix: “>300” or “<300”
- `created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`

Request

CURL

```
curl https://api.paymill.com/v2/preauthorizations \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$preAuth = new Paymill\Models\Request\Preauthorization();

$response = $request->getAll($preAuth);
```

JAVA

```
PreauthorizationService preauthorizationService = paymillContext.getPreauthorizationService();
PaymillList<Preauthorization> preauthorizations = preauthorizationService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.preauthorizations.list({},
    function(err, preauthorization) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("preauthorization data " + preauthorization.data)
;
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
```

```
preauths = p.getpreauth()
```

RUBY

```
Paymill::Preauthorization.all
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Preauthorizationservice;
PaymillList<Preauthorization> preauthorizations = preauthorizationService.ListAsync().Result;
```

JS

```
pm.preauthorizations.list().then(function(pmlist) {
 console.log(pmlist.items.length + " preauths from total of " + pmlist.count);
}, function(error) {
 console.log("couldnt list preauths:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id" : "preauth_31eb90495837447f76b7",
            "description" : "description example",
            "amount" : "4200",
            "currency" : "EUR",
            "status" : "closed",
            "livemode" : false,
            "created_at" : 1349948920,
            "updated_at" : 1349948920,
            "payment" : "<Object>",
            "client" : "<Object>",
            "app_id" : null
        }
    ],
    "data_count" : "1",
    "mode" : "test"
}
```


Sub objects

- preauthorization.payment returns a payment object for credit card
- preauthorization.client returns a client object

Export **Preauthorizations** List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of preauthorizations. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

> - **amount**
> - **created_at**
> - **updated_at**

Request

CURL

```
curl https://api.paymill.com/v2/preauthorizations \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

Available filters:

- **amount**
- **client**
- **created_at**
- **payment**
- **updated_at**

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response


```
"id";"amount";"currency";"description";"status";"livemode";"created_at";"updated_at";"app_id";"payment_id";"payment_type";"payment_client";"payment_card_type";"payment_country";"payment_bin";"payment_expire_month";"payment_expire_year";"payment_card_holder";"payment_last4";"payment_updated_at";"payment_created_at";"payment_app_id";"payment_is_recurring";"payment_is_usable_for_preauthorization";"payment_client_id";"payment_client_email";"payment_client_description";"payment_client_app_id";"payment_client_updated_at";"payment_client_created_at";"payment_client_payment_subscription"
"preauth_595d96437ad81d5ca965";"499";"EUR";"Subscription#sub_5dd7af6fa6d58c60a4e9";"preauth_subscription";"";"1342427064";"1342427064";"";"pay_2311e5a076ab0b9c2cdb0399";"creditcard";"client_33c8f8c13d759d00b144";"visa";"DE";"411111";"2";"2016";"test card holder";"1111";"1342427064";"1342427064";"";"1";"1";"client_33c8f8c13d759d00b144";"testclient@example.com";"test client";"";"1342427064";"1342427064";""
```


Transactions ¶

A transaction is the charging of a credit card or a direct debit. In this case you need a new transaction object with either a valid token, payment, client + payment or preauthorization. Every transaction has a unique identifier which will be generated by Paymill to identify every transaction. You can issue/create, list and display transactions in detail. Refunds can be done in an extra entity.

`Transaction` Object ¶

Attributes ¶

id : string
Unique identifier of this transaction.

amount : string
Formatted amount of this transaction.

origin_amount : integer (>0)
The used amount, smallest possible unit per currency (for euro, we're calculating the amount in cents).

currency : string
ISO 4217 formatted currency code.

status : enum(open, pending, closed, failed, partial_refunded, refunded, preauthorize, chargeback)
Indicates the current status of this transaction, e.g closed means the transaction is sucessfully transfered, refunded means that the amount is fully or in parts refunded.

description : string or null
Need a additional description for this transaction? Maybe your shopping cart ID or something like that?

livemode : boolean
Whether this transaction was issued while being in live mode or not.

is_fraud : boolean
The transaction is marked as fraud or not.

refunds : list
refund objects or null

payment : creditcard-object or directdebit-object or null

client : clients-object or null

preauthorization : preauthorizations-object or null

created_at : integer
Unix-Timestamp for the creation date.

updated_at : integer
Unix-Timestamp for the last update.

response_code : integer
Response code

short_id : string
Unique identifier of this transaction provided to the acquirer for the statements.

invoices : list
PAYMILL invoice where the transaction fees are charged or null.

fees : list
App fees or null.

app_id : string or null
App (ID) that created this transaction or null if created by yourself.

mandate_refere.. string or null
SEPA mandate reference, can be optionally specified for direct debit transactions. If specified for other payment methods, it has no effect but must still be valid. If specified, the string must not be empty, can be up to 35

Example


```
{
    "id" : "tran_54645bcb98ba7acfe204",
    "amount" : "4200",
    "origin_amount" : 4200,
    "status" : "closed",
    "description" : null,
    "livemode" : false,
    "is_fraud" : false,
    "refunds" : null,
    "currency" : "EUR",
    "created_at" : 1349946151,
    "updated_at" : 1349946151,
    "response_code" : 20000,
    "short_id" : "0000.1212.3434",
    "invoices" : [],
    "payment" : "<Object>",
    "client" : "<Object>",
    "preauthorization" : null,
    "fees" : [],
    "app_id" : null,
    "mandate_reference" : null,
    "is_refundable" : true,
    "is_markable_as_fraud" : true
}
```


Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

characters long and may contain
digits 0-9
letters a-z A-Z
allowed special characters: ‘ , . : + - / ( ) ?

is_refundable: boolean
The transaction is refundable.

is_markable_as… boolean
The transaction is markable as fraud.

Fee object

type
string Fee type
application
string Unique identifier of the app which charges the fee
payment
string Unique identifier of the payment from which the fee will be charged
amount
integer Fee amount in the smallest currency unit e.g. “420” for 4.20 €
currency
string ISO 4217 formatted currency code.
billed_at
integer or null Unix-Timestamp for the billing date.

Create new `Transaction` with ... ¶

You have to create at least either a token or a payment object before you can execute a transaction. You get back a response object indicating whether a transaction was successful or not.

Note

The transaction will not be charged at the bank if the test keys are implemented in your code. Please use only the test credit cards mentioned in the documentation.

Attributes ¶

amount: integer (>0)
Amount (in cents) which will be charged

currency: string
ISO 4217 formatted currency code

description: string or null
A short description for the transaction

client: string or null
The identifier of a client (client-object)
When this parameter is used, you have also to specify a payment method which is not assigned to a client yet. If you attempt to use

Payment

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "payment=pay_2f82a672574647cd911d" \
  -d "description=Test Transaction"
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setAmount(4200) // e.g. "4200" for 42.00 EUR
            ->setCurrency('EUR')
            ->setPayment('pay_2f82a672574647cd911d')
            ->setDescription('Test Transaction');

$response = $request->create($transaction);
```

JAVA

```
PaymentService paymentService = paymillContext.getPaymentService();
Payment payment = paymentService.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6"
);
TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = transactionService.createWithToken(
    payment,
    4200,
    "EUR",
    "Test Transaction"
```

this parameter when creating a transaction and when specifying a token or preauthorization, the specified client will be ignored.

token : string
A token generated through our JavaScript-Bridge
When this parameter is used, none of the following should be used: payment, preauthorization.

payment : string
The identifier of a payment (creditcard-object or directdebit-object)
When this parameter is used, none of the following should be used: token, preauthorization.

preauthorization : string
The identifier of a preauthorization (preauthorizations-object)
When this parameter is used, none of the following should be used: token, payment.

fee_amount : integer or null
Fee included in the transaction amount (set by a connected app).
Mandatory if fee_payment is set

fee_payment : string or null
The identifier of the payment from which the fee will be charged (creditcard-object or directdebit-object).
Mandatory if fee_amount is set

fee_currency : string or unset
The currency of the fee (e.g. EUR, USD). If it´s not set, the currency of the transaction is used.
We suggest to always use as it might cause problems, if your account does not support the same currencies as your merchants accounts.

mandate_refere.. string or null
SEPA mandate reference, can be optionally specified for direct debit transactions. If specified for other payment methods, it has no effect but must still be valid. If specified, the string must not be empty, can be up to 35 characters long and may contain
digits 0-9
letters a-z A-Z
allowed special characters: ‘ , . : + - / ( ) ?

```
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.create(
    {
     amount: 4200,
     currency: 'EUR',
     payment: 'pay_a818b847db6ce5ff636f'
     description: 'Test Transaction'
 },
    function(err, transaction) {
        if (err) {
            console.log("Couldn't create the transaction record");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
transaction = p.transact(
    amount=4200,
    currency='EUR',
    description='Test Transaction',
    payment='pay_2f82a672574647cd911d'
)
```

RUBY

```
Paymill::Transaction.create amount: 4200, currency: "EUR",
    payment: "pay_2f82a672574647cd911d",
    description: "Test Transaction"
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.CreateWithTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6"
).Result;
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.CreateWithTokenAsync(
    payment,
    4200,
    "EUR",
    "Test Transaction"
).Result;
```

JS

```
pm.transactions.createWithPayment("pay_2f82a672574647cd911d", 4200, "
EUR", "Test Transaction").then(function(transaction) {
 console.log("transaction:" + transaction.id);
}, function(error) {
 console.log("couldnt create transaction:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "tran_1f42e10cf14301067332",
        "amount" : "4200",
```

```
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : null,
        "fees" : [],
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```


Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Token

When using a credit card or direct debit account for the first time, you can use a token. For the second transaction and on, use the payment object created for this token. Tokens are not reusable

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "token=098f6bcd4621d373cade4e832627b4f6" \
  -d "description=Test Transaction"
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setAmount(4200) // e.g. "4200" for 42.00 EUR
            ->setCurrency('EUR')
            ->setToken('098f6bcd4621d373cade4e832627b4f6')
            ->setDescription('Test Transaction');

$response = $request->create($transaction);
```

JAVA

```
TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = transactionService.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "Test Transaction"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);
```

```
paymill.transactions.create(
    {
        amount: 4200,
        currency: 'EUR',
        token: '098f6bcd4621d373cade4e832627b4f6',
        description: 'Test Transaction'
    },
    function(err, transaction) {
        if (err) {
            console.log("Couldn't create the transaction record");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
transaction = p.transact(
    amount=4200,
    currency='EUR',
    description='Test Transaction',
    token='098f6bcd4621d373cade4e832627b4f6'
)
```

RUBY

```
Paymill::Transaction.create amount: 4200, currency: "EUR",
    token: "098f6bcd4621d373cade4e832627b4f6",
    description: "Test Transaction"
```

.NET

```
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.CreateWithTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "Test Transaction"
).Result;
```

JS

```
pm.transactions.createWithToken("098f6bcd4621d373cade4e832627b4f6", 4
200, "EUR", "Test Transaction").then(function(transaction) {
 console.log("transaction:" + transaction.id);
}, function(error) {
 console.log("couldnt create transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_b3692e8e063900d27a40",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
```

```
        "invoices" : [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : null,
        "fees": [],
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```

Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Client & Payment

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "client=client_c781b1d2f7f0f664b4d9" \
  -d "payment=pay_a818b847db6ce5ff636f" \
  -d "description=Test Transaction"
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setAmount(4200) // e.g. "4200" for 42.00 EUR
            ->setCurrency('EUR')
            ->setClient('client_c781b1d2f7f0f664b4d9')
            ->setPayment('pay_2f82a672574647cd911d')
            ->setDescription('Test Transaction');

$response = $request->create($transaction);
```

JAVA

```
TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = transactionService.createWithPaymentAndClie
nt(
    "pay_a818b847db6ce5ff636f",
    "client_c781b1d2f7f0f664b4d9",
    4200,
    "EUR"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.create(
    {
     amount: 4200,
     currency: 'EUR',
     client: 'client_c781b1d2f7f0f664b4d9',
     payment: 'pay_a818b847db6ce5ff636f'
     description: 'Test Transaction'
 },
    function(err, transaction) {
        if (err) {
```

```
            console.log("Couldn't create the transaction record");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
transaction = p.transact(
    amount=4200,
    currency='EUR',
    description='Test Transaction',
    client='client_c781b1d2f7f0f664b4d9',
    payment='pay_a818b847db6ce5ff636f'
)
```

RUBY

```
Paymill::Transaction.create amount: 4200, currency: "EUR",
    client: "client_c781b1d2f7f0f664b4d9",
    payment: "pay_2f82a672574647cd911d",
    description: "Test Transaction"
```

.NET

```
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.CreateWithPaymentAndClie
ntAsync(
    "pay_a818b847db6ce5ff636f",
    "client_c781b1d2f7f0f664b4d9",
    4200,
    "EUR"
).Result;
```

JS

```
pm.transactions.createWithPayment("pay_2f82a672574647cd911d", 4200, "
EUR", "Test Transaction", "client_c781b1d2f7f0f664b4d9").then(functio
n(transaction) {
 console.log("transaction:" + transaction.id);
}, function(error) {
 console.log("couldnt create transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_663dada2ffd9b47bd1bf",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : null,
        "fees": [],
        "app_id" : null,
```

```
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```

Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Preauthorization

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "preauthorization=preauth_ec54f67e52e92051bd65" \
  -d "description=Test Transaction"
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setAmount(4200) // e.g. "4200" for 42.00 EUR
            ->setCurrency('EUR')
            ->setPreauthorization('preauth_ec54f67e52e92051bd65')
            ->setDescription('Test Transaction');

$response = $request->create($transaction);
```

JAVA

```
PreauthorizationService preauthorizationService = paymillContext.getP
reauthorizationService();
Preauthorization preauthorization = preauthorizationService.createWit
hToken(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR"
).getPreauthorization();

TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = this.transactionService.createWithPreauthor
ization(
    preauthorization,
    4200,
    "EUR",
    "Test Transaction"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.create(
    {
     amount: 4200,
     currency: 'EUR',
     preauthorization: 'preauth_ec54f67e52e92051bd65'
     description: 'Test Transaction'
 },
    function(err, transaction) {
        if (err) {
```

```
            console.log("Couldn't create the transaction record");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
    transaction = p.transact(
    amount=4200,
    currency='EUR',
    description='Test Transaction',
    preauth='preauth_ec54f67e52e92051bd65'
)
```

RUBY

```
Paymill::Transaction.create amount: 4200, currency: "EUR",
    preauthorization: "preauth_ec54f67e52e92051bd65",
    description: "Test Transaction"
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Preauthorizationservice;
Preauthorization preauthorization = preauthorizationService.CreateWithTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR"
).Result;

TransactionService transactionService = paymillContext.TransactionService;
Transaction transaction = transactionService.CreateWithPreauthorizationAsync(
    preauthorization,
    4200,
    "EUR",
    "Test Transaction"
).Result;
```

JS

```
pm.transactions.createWithPreauthorization("preauth_ec54f67e52e92051bd65", 4200, "EUR", "Test Transaction").then(function(transaction) {
 console.log("transaction:" + transaction.id);
}, function(error) {
 console.log("couldnt create transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_ca3e7d41fb16d0157a99",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
```

```
        "invoices": [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : "<Object>",
        "fees": [],
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```

Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

App fee

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "token=098f6bcd4621d373cade4e832627b4f6" \
  -d "description=Test Transaction" \
  -d "fee_amount=420" \
  -d "fee_payment=pay_3af44644dd6d25c820a8" \
  -d "fee_currency=EUR"
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setAmount(4200) // e.g. "4200" for 42.00 EUR
            ->setCurrency('EUR')
            ->setToken('098f6bcd4621d373cade4e832627b4f6')
            ->setDescription('Test Transaction')
            ->setFeeAmount(420)
            ->setFeePayment('pay_3af44644dd6d25c820a8')
            ->setFeeCurrency('EUR');

$response = $request->create($transaction);
```

JAVA

```
Fee fee = new Fee();
fee.setAmount( 420 );
fee.setPayment( "pay_3af44644dd6d25c820a8" );
TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = transactionService.createWithTokenAndFee(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    fee
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.create(
    {
       amount: 4200,       // e.g. "4200" for 42.00 EUR
       currency: 'EUR',
```

```
        token: '098f6bcd4621d373cade4e832627b4f6',
        description: 'Test Transaction',
        fee_amount: 420,    // e.g. "420" for 4.20 EUR
        fee_payment: 'pay_3af44644dd6d25c820a8'
    },
    function(err, transaction) {
        if (err) {
            console.log("Couldn't create the transaction record");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
# Not implemented yet
```

RUBY

```
Paymill::Transaction.create amount: 4200, currency: "EUR",
    token: "098f6bcd4621d373cade4e832627b4f6",
    description: "Test Transaction",
    fee_amount: 420,
    fee_payment: "pay_3af44644dd6d25c820a8",
    fee_currency: "EUR"
```

.NET

```
Fee fee = new Fee();
fee.Amount =  420;
fee.Payment = "pay_3af44644dd6d25c820a8";
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.CreateWithTokenAndFeeAsy
nc(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    fee
).Result;
```

JS

```
pm.transactions.createWithToken("098f6bcd4621d373cade4e832627b4f6", 4
200, "EUR", "Test Transaction", null, 420, "pay_3af44644dd6d25c820a8"
).then(function(transaction) {
 console.log("transaction:" + transaction.id);
}, function(error) {
 console.log("couldnt create transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_ca3e7d41fb16d0157a99",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "invoices": [],
        "payment" : "<Object>",
        "client" : "<Object>",
```

```
        "preauthorization" : "<Object>",
        "fees" : [
          {
            "type" : "application",
            "application" : "app_1d70acbf80c8c35ce83680715c06be0",
            "payment" : "pay_3af44644dd6d25c820a8",
            "amount" : 420,
            "currency": "EUR",
            "billed_at": null
          }
        ],
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```

Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Transaction Details by ... ¶

To receive the details of an existing transaction, call the unique transaction ID or the unique slv number. You can find the ID in the response of the previous request (or the slv number on your account statement). The return is a refund object with the information of the used payment, client and transaction attributes.

Attributes ¶

id :	string Unique identifier of this transaction
slv number :	string Unique 10 character long slv number of this transaction

Transaction Id

Request

CURL

```
curl https://api.paymill.com/v2/transactions/tran_023d3b5769321c64943
5 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setId('tran_023d3b5769321c649435');

$response = $request->getOne($transaction);
```

JAVA

```
TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = transactionService.get("tran_023d3b5769321c
649435");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.details('tran_023d3b5769321c649435',
    function(err, transaction) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
```

```
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
transaction = p.gettransdetails('tran_023d3b5769321c649435')
```

RUBY

```
Paymill::Transaction.find "tran_023d3b5769321c649435"
```

.NET

```
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.GetAsync("tran_023d3b576
9321c649435").Result;
```

JS

```
pm.transactions.detail("tran_023d3b5769321c649435").then(function(tra
nsaction) {
    console.log("transaction:" + transaction.id);
}, function(error) {
    console.log("couldnt get transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_023d3b5769321c649435",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices": [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : null,
        "fees" : [],
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```


Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

SLV

Request

CURL

```
curl https://api.paymill.com/v2/transactions/slv_4125875679 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setId('slv_4125875679');

$response = $request->getOne($transaction);
```

JAVA

```
TransactionService transactionService = paymillContext.getTransaction
Service();
Transaction transaction = transactionService.get("slv_4125875679");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.details('slv_4125875679',
    function(err, transaction) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
transaction = p.gettransdetails('slv_4125875679')
```

RUBY

```
Paymill::Transaction.find "slv_4125875679"
```

.NET

```
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.GetAsync("slv_4125875679
").Result;
```

JS

```
pm.transactions.detail("slv_4125875679").then(function(transaction) {
    console.log("transaction:" + transaction.id);
}, function(error) {
    console.log("couldnt get transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_023d3b5769321c649435",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "Test Transaction",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
```

```
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices": [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : null,
        "fees" : [],
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```

Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Update `Transaction` ¶

This function updates the description of a transaction.

Attributes ¶

id : string
Unique identifier of this transaction

description : string or null
Description for the transaction

status : string “chargeback”
You can only use chargeback at a successful direct debit transaction.

Request

CURL

```
curl https://api.paymill.com/v2/transactions/tran_023d3b5769321c649435 \
  -u <YOUR_PRIVATE_KEY>:  \
  -d "description=My updated transaction description" \
  -X PUT
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();
$transaction->setId('tran_023d3b5769321c649435')
            ->setDescription('My updated transaction description');

$response = $request->update($transaction);
```

JAVA

```
TransactionService transactionService = paymillContext.getTransactionService();
Transaction transaction = transactionService.get("tran_023d3b5769321c649435");
transaction.setDescription("My updated transaction description");
transactionService.update( transaction );
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.update('tran_023d3b5769321c649435',
  {
      description: 'My updated transaction description'
```

```
  },
    function(err, transaction) {
        if (err) {
            console.log("Couldn't update the transaction record");
            return;
        }
        console.log("transaction id " + transaction.data.id);
    }
);
```

PYTHON

```
# Not implemented yet
```

RUBY

```
Paymill::Transaction.update_attributes "tran_023d3b5769321c649435",
    description: "My updated transaction description"
# or
transaction = Paymill::Transaction.create amount: 4200, currency: "EU
R",
    token: "098f6bcd4621d373cade4e832627b4f6",
    description: "Test Transaction"
transaction.update_attributes description: "My updated transaction de
scription"
```

.NET

```
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.GetAsync("tran_023d3b576
9321c649435").Result;
transaction.Description = "My updated transaction description";
transactionService.UpdateAsync(transaction).Result;
```

JS

```
pm.transactions.detail("tran_023d3b5769321c649435").then(function(tra
nsaction) {
 transaction.description = "My updated transaction description";
 return pm.transactions.update(transaction);
}).then(function(updatedTransaction) {
 console.log("updated transaction:" + updatedTransaction.description)
;
}, function(error) {
 console.log("couldnt update transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "tran_023d3b5769321c649435",
        "amount" : "4200",
        "origin_amount" : 4200,
        "status" : "closed",
        "description" : "My updated transaction description",
        "livemode" : false,
        "refunds" : null,
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "status" : "closed",
        "is_fraud" : false,
        "short_id" : "0000.1212.3434",
        "fees" : [],
        "invoices" : [],
        "payment" : "<Object>",
        "client" : "<Object>",
        "preauthorization" : null,
```

```
        "app_id" : null,
        "mandate_reference" : null,
        "is_refundable" : true,
        "is_markable_as_fraud" : true
    },
    "mode" : "test"
}
```

List `Transactions` ¶

This function returns a JSON object with a list of transactions. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- **`count`**
- **`offset`**
- **`created_at`**

Available filters:

- **`client=<client id>`**
- **`payment=<payment id>`**
- **`amount=[>|<]<integer>`** e.g. “300” or with prefix: “>300” or “<300”
- **`description=<string>`**
- **`created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`**
- **`updated_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`**
- **`status=<string>`** see list below
- **`last4=<integer>`** last 4 digits of the credit card

Available status for filters:

- **`open`**
- **`closed`**
- **`failed`**
- **`preauth`**
- **`pending`**
- **`refunded`**
- **`partially_refunded`**
- **`chargeback`**

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$transaction = new Paymill\Models\Request\Transaction();

$response = $request->getAll($transaction);
```

JAVA

```
TransactionService transactionService = paymillContext.getTransaction
Service();
PaymillList<Transaction> transactions = transactionService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.transactions.list({},
    function(err, transaction) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("transaction data " + transaction.data);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
transactions = p.gettrans()
```

RUBY

```
Paymill::Transaction.all
```

.NET

```
TransactionService transactionService = paymillContext.TransactionSer
vice;
PaymillList<Transaction> transactions = transactionService.ListAsync(
).Result;
```

JS

```
pm.transactions.list().then(function(pmlist) {
 console.log(pmlist.items.length + " transactions from total of " + p
mlist.count);
}, function(error) {
 console.log("couldnt list transactions:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id" : "tran_03bb8f63d5278f723ced",
            "amount" : "4200",
            "origin_amount" : 4200,
            "status" : "closed",
            "description" : "ShoppingcartID 873242",
            "livemode" : false,
            "refunds" : null,
            "currency" : "EUR",
            "created_at" : 1349946151,
            "updated_at" : 1349946151,
            "response_code" : 20000,
            "short_id" : "0000.1212.3434",
            "is_fraud" : false,
            "invoices" : [],
            "payment" : "<Object>",
            "client" : "<Object>",
            "preauthorization" : null,
            "fees" : [],
            "app_id" : null,
            "mandate_reference" : null,
            "is_refundable" : true,
            "is_markable_as_fraud" : true
        },
        {
            "id" : "tran_5e3105d4c2f34fe9d1f",
            "amount" : "5699",
            "origin_amount" : 5699,
            "status" : "closed",
            "description" : "ShoppingcartID 873243",
            "livemode" : false,
            "refunds" : null,
            "currency" : "EUR",
            "created_at" : 1349953847,
            "updated_at" : 1349953847,
            "response_code" : 20000,
            "short_id" : "0000.1212.3435",
            "is_fraud" : false,
            "invoices" : [],
            "payment" : "<Object>",
            "client" : "<Object>",
            "preauthorization" : null,
            "fees" : [],
            "app_id" : null,
            "mandate_reference" : null,
            "is_refundable" : true,
            "is_markable_as_fraud" : true
        }
    ],
    "data_count" : "2",
    "mode" : "test"
}
```


Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Export `Transactions` List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of transactions. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `amount`
- `created_at`
- `currency`
- `description`
- `status`
- `updated_at`

Available filters:

- `amount`
- `client`
- `created_at`
- `currency`
- `description`
- `last4`
- `payment`
- `status`
- `updated_at`

Request

CURL

```
curl https://api.paymill.com/v2/transactions \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response


```
"id";"amount";"origin_amount";"status";"description";"livemode";"client_id";"client_email";"client_description";"client_app_id";"client_updated_at";"client_created_at";"client_payment_subscription";"client_currency";"client_created_at";"client_updated_at";"client_response_code";"client_short_id";"client_is_fraud";"client_invoices_app_id";"client_invoices_preauthorization";"client_invoices_fees_payment_id";"client_invoices_fees_payment_type";"client_invoices_fees_payment_client";"client_invoices_fees_payment_card_type";"client_invoices_fees_payment_country";"client_invoices_fees_payment_bin";"client_invoices_fees_payment_expire_month";"client_invoices_fees_payment_expire_year";"client_invoices_fees_payment_card_holder";"client_invoices_fees_payment_last4";"client_invoices_fees_payment_updated_at";"client_invoices_fees_payment_created_at";"client_invoices_fees_payment_app_id";"client_invoices_fees_payment_is_recurring";"client_invoices_fees_payment_is_usable_for_preauthorization";"client_invoices_fees_payment_is_refundable";"client_invoices_fees_payment_is_markable_as_fraud"
"tran_51502ed3c9ecadee9bb5790af422";"000";"1200";"refunded";"";"";"client_1d723df9e092c5b06a0e";"";"";"";;;"";"EUR";"1404575620";"1404575620";"20000";"7357.7357.7357";"";"";"";"pay_c0d83abbc2fb92a0bae2c38e";"creditcard";"client_1d723df9e092c5b06a0e";"visa";"DE";"411111";"10";"2014";"";"1111";"1404575620";"1404575620";"";"1";"1";"";"1"
```

Refunds ¶

Refunds are own objects with own calls for existing transactions. The refunded amount will be credited to the account of the client.

Refund Object ¶

Attributes ¶

- id : string
  Unique identifier of this refund.
- transaction : transaction object
- amount : integer (>0)
  The refunded amount.
- status : enum(open, pending, refunded)
  Indicates the current status of this transaction.
- description : string or null
  The description given for this refund.
- livemode : boolean
  Whether this refund happend in test- or in livemode.
- created_at : integer
  Unix-Timestamp for the creation date.
- updated_at : integer
  Unix-Timestamp for the last update.
- app_id : string or null
  App (ID) that created this refund or null if created by yourself.

Example

```
{
    "id" : "refund_87bc404a95d5ce616049",
    "amount" : "4200",
    "status" : "refunded",
    "description" : null,
    "livemode" : false,
    "created_at" : 1349947042,
    "updated_at" : 1349947042,
    "response_code" : 20000,
    "transaction" : "<Object>",
    "app_id": null
}
```

Sub objects

- refund.transaction returns a transaction object

Refund Transaction ¶

This function refunds a transaction that has been created previously and was refunded in parts or wasn't refunded at all. The inserted amount will be refunded to the credit card / direct debit of the original transaction. There will be some fees for the merchant

Request

CURL

```
curl https://api.paymill.com/v2/refunds/tran_023d3b5769321c649435 \
```

for every refund.

Note

- You can refund parts of a transaction until the transaction amount is fully refunded. But be careful there will be a fee for every refund
- There is no need to define a currency for refunds, because they will be in the same currency as the original transaction

Attributes ¶

amount : integer (>0)
Amount (in cents) which will be charged

description : string or null
additional description for this refund

```
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200"
```

PHP

```php
$refund = new Paymill\Models\Request\Refund();
$refund->setId('tran_023d3b5769321c649435')
       ->setAmount(4200) // e.g. "4200" for 42.00 EUR
       ->setDescription('Sample Description');

$response = $request->create($refund);
```

JAVA

```java
TransactionService = paymillContext.getTransactionService();
Transaction transaction = this.transactionService.createWithToken(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "For refund"
);
RefundService = paymillContext.getRefundService();
Refund refund = refundService.refundTransaction(
    transaction,
    4200,
    "Sample Description"
);
```

NODE.JS

```node
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.refunds.refund('tran_023d3b5769321c649435', 4200, "",
    function(err, refund) {
        if (err) {
            console.log("Couldn't create the refund record");
            return;
        }
        console.log("refund id " + refund.data.id);
    }
);
```

PYTHON

```python
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
refund = p.refund(
    tranid='tran_023d3b5769321c649435',
    amount=4200,
    description='Test Refund'
)
```

RUBY

```ruby
Paymill::Refund.create id: "tran_f5bc741dc3809ad3c62fd255e60c",
    amount: 4200
```

.NET

```dotnet
TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.CreateWithTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR",
    "For refund"
).Result;
RefundService refundService = paymillContext.RefundService;
Refund refund = refundService.RefundTransactionAsync(
    transaction,
```

```
    4200,
    "Sample Description"
).Result;
```

JS

```
pm.transactions.refund("result", 4200, "Sample Description").then(fun
ction(refund) {
 console.log("refund:" + refund.id);
}, function(error) {
 console.log("couldnt refund transaction:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "refund_70392dc6a734a8233130",
        "amount" : "4200",
        "status" : "refunded",
        "description" : null,
        "livemode" : false,
        "created_at" : 1365154751,
        "updated_at" : 1365154751,
        "response_code" : 20000,
        "transaction" : "<Object>",
        "app_id" :  null
    },
    "mode" : "test"
}
```


Sub objects

- transaction.refunds returns refund objects
- transaction.payment returns a payment object for credit card
- transaction.client returns a client object
- transaction.preauthorization returns a preauthorization object

Refund Details ¶

Returns detailed informations of a specific refund.

Request

CURL

```
curl https://api.paymill.com/v2/refunds/refund_87bc404a95d5ce616049 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$refund = new Paymill\Models\Request\Refund();
$refund->setId('refund_773ab6f9cd03428953c9');

$response = $request->getOne($refund);
```

JAVA

```
RefundService = paymillContext.getRefundService();
Refund refund = refundService.get("refund_773ab6f9cd03428953c9");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.refunds.details('refund_87bc404a95d5ce616049',
    function(err, refund) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("refund id " + refund.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
refund = p.getrefdetails('refund_773ab6f9cd03428953c9')
```

RUBY

```
Paymill::Refund.find "refund_87bc404a95d5ce616049"
```

.NET

```
RefundService refundService = paymillContext.RefundService();
Refund refund = refundService.GetAsync("refund_773ab6f9cd03428953c9")
.Result;
```

JS

```
pm.refunds.detail("refund_773ab6f9cd03428953c9").then(function(refund
) {
 console.log("refund:" + refund.id);
}, function(error) {
 console.log("couldnt get refund:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "refund_87bc404a95d5ce616049",
        "amount" : "4200",
        "status" : "refunded",
        "description" : null,
        "livemode" : false,
        "created_at" : 1349947042,
        "updated_at" : 1349947042,
        "response_code" : 20000,
        "transaction" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```


Sub objects

- refund.transaction returns a transaction object

List **Refunds** ¶

This function returns a list of existing refunds. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- **count**
- **offset**
- **transaction**
- **client**
- **amount**
- **created_at**

Available filters:

- **client=<client id>**
- **transaction=<transaction id>**
- **amount=[>|<]<integer>** e.g. “300” or with prefix: “>300” or “<300”
- **created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>**

Request

CURL

```
curl https://api.paymill.com/v2/refunds \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$refund = new Paymill\Models\Request\Refund();

$response = $request->getAll($refund);
```

JAVA

```
RefundService = paymillContext.getRefundService();
PaymillList<Refund> refunds = refundService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.refunds.list({},
    function(err, refund) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("refund data " + refund.data);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
refunds = p.getrefs()
```

RUBY

```
Paymill::Refund.all
```

.NET

```
RefundService refundService = paymillContext.RefundService;
PaymillList<Refund> refunds = refundService.ListAsync;
```

JS

```
pm.refunds.list().then(function(pmlist) {
 console.log(pmlist.items.length + " refunds from total of " + pmlist
.count);
}, function(error) {
 console.log("couldnt list transactions:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id" : "refund_87bc404a95d5ce616049",
            "amount" : "4200",
            "status" : "refunded",
            "description" : null,
            "livemode" : false,
            "created_at" : 1349947042,
```

```
            "updated_at" : 1349947042,
            "response_code" : 20000,
            "transaction" : "<Object>",
            "app_id" : null
        }
    ],
    "data_count" : "1",
    "mode" : "test"
}
```


Sub objects

- refund.transaction returns a transaction object

Export **Refunds** List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of refunds. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- **amount**
- **created_at**
- **updated_at**

Available filters:

- **amount**
- **client**
- **created_at**
- **transaction**
- **updated_at**

Request

CURL

```
curl https://api.paymill.com/v2/refunds \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response


```
"id";"amount";"status";"description";"livemode";"created_at";"updated
_at";"response_code";"app_id";"transaction_id";"transaction_amount";"
transaction_origin_amount";"transaction_status";"transaction_descript
ion";"transaction_livemode";"transaction_client_id";"transaction_clie
nt_email";"transaction_client_description";"transaction_client_app_id
";"transaction_client_updated_at";"transaction_client_created_at";"tr
ansaction_client_payment_subscription";"transaction_client_currency";
"transaction_client_created_at";"transaction_client_updated_at";"tran
```

saction_client_response_code";"transaction_client_short_id";"transaction_client_is_fraud";"transaction_client_invoices_app_id";"transaction_client_invoices_preauthorization";"transaction_client_invoices_fees_payment_id";"transaction_client_invoices_fees_payment_type";"transaction_client_invoices_fees_payment_client";"transaction_client_invoices_fees_payment_card_type";"transaction_client_invoices_fees_payment_country";"transaction_client_invoices_fees_payment_bin";"transaction_client_invoices_fees_payment_expire_month";"transaction_client_invoices_fees_payment_expire_year";"transaction_client_invoices_fees_payment_card_holder";"transaction_client_invoices_fees_payment_last4";"transaction_client_invoices_fees_payment_updated_at";"transaction_client_invoices_fees_payment_created_at";"transaction_client_invoices_fees_payment_app_id";"transaction_client_invoices_fees_payment_is_recurring";"transaction_client_invoices_fees_payment_is_usable_for_preauthorization";"transaction_client_invoices_fees_payment_is_refundable";"transaction_client_invoices_fees_payment_is_markable_as_fraud"
"refund_84666650d347013472a3";"1200";"refunded";"";"";"1342427064";"1342427064";"20000";"";"tran_51502ed3c9ecadee9bb5790af422";"000";"1200";"refunded";"";"";"client_1d723df9e092c5b06a0e";"";"";"";;;"";"EUR";"1342427064";"1342427064";"20000";"7357.7357.7357";"";"";"";"pay_c0d83abbc2fb92a0bae2c38e";"creditcard";"client_1d723df9e092c5b06a0e";"visa";"DE";"411111";"10";"2014";"";"1111";"1342427064";"1342427064";"";"1";"1";"";"1"

Clients ¶

The clients object is used to edit, delete, update clients as well as to permit refunds, subscriptions, insert credit card details for a client, edit client details and of course make transactions. Clients can be created individually by you or they will be automatically generated with the transaction if there is no client ID transmitted.

`Client` Object ¶

Attributes ¶

id : string
Unique identifier of this client.

email : string or null
Mail address of this client.

description : string or null
Additional description for this client, perhaps the identifier from your CRM system?

created_at : integer
Unix-Timestamp for the creation date.

Example

```
{
    "id"           :  "client_88a388d9dd48f86c3136",
    "email"        :  "lovely-client@example.com",
    "description"  :  null,
    "created_at"   :  1340199740,
    "updated_at"   :  1340199760,
    "payment"      :  "[ <Object>, ... ] or null",
    "subscription" :  "[ <Object>, ... ] or null",
    "app_id"       :  null
}
```

Sub objects

- client.payment returns payment objects for credit card or direct debit
- client.subscription returns subscription objects or null

updated_at : integer
Unix-Timestamp for the last update.

payment : list
creditcard-object or directdebit-object

subscription : list or null
subscriptions-object

app_id : string or null
App (ID) that created this client or null if created by yourself.

Create new `Client` ¶

This function creates a client object.

Attributes ¶

email : string or null
Mail address of the client, is optional if the transaction creates an user itself

description : string or null
Description for the client

Request

CURL

```
curl https://api.paymill.com/v2/clients \
  -u <YOUR_PRIVATE_KEY>: \
  -d "email=lovely-client@example.com" \
  -d "description=Lovely Client"
```

PHP

```
$client = new Paymill\Models\Request\Client();
$client->setEmail('max.mustermann@example.com')
       ->setDescription('Lovely Client')

$response = $request->create($client);
```

JAVA

```
ClientService clientService = paymillContext.getClientService();
Client client = clientService.createWithEmailAndDescription(
    "lovely-client@example.com",
    "Lovely Client"
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.clients.create(
    {
     email: 'lovely-client@example.com',
        description: 'Lovely Client'
 },
    function(err, client) {
        if (err) {
            console.log("Couldn't create the client record");
            return;
        }
        console.log("client id " + client.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
client = p.newclient(
  email='lovely-client@example.com',
  description='Lovely Client'
```

```
)
```

RUBY

```ruby
client = Paymill::Client.create email: "lovely-client@example.com",
    description: "Lovely Client"
```

.NET

```dotnet
ClientService clientService = paymillContext.ClientService;
Client client = clientService.CreateWithEmailAndDescriptionAsync(
    "lovely-client@example.com",
    "Lovely Client"
).Result;
```

JS

```javascript
pm.clients.create("max.mustermann@example.com", "Lovely Client").then
(function(client) {
    console.log("client:" + client.id);
}, function(error) {
    console.log("couldnt get client:" + error);
});
```

Response

```
{
    "data" : {
        "id"           : "client_88a388d9dd48f86c3136",
        "email"        : "lovely-client@example.com",
        "description"  : "Lovely Client",
        "created_at"   : 1342438695,
        "updated_at"   : 1342438695,
        "payment"      : "[ <Object>, ... ]",
        "subscription" : "<Object>",
        "app_id"       : null
    },
    "mode" : "test"
}
```

Sub objects

- client.payment returns payment objects for credit card or direct debit
- client.subscription returns a subscription object

`Client` Details ¶

To get the details of an existing client you'll need to supply the client ID. The client ID is returned by creating a client.

Request

CURL

```
curl https://api.paymill.com/v2/clients/client_88a388d9dd48f86c3136 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```php
$client = new Paymill\Models\Request\Client();
$client->setId('client_88a388d9dd48f86c3136');

$response = $request->getOne($client);
```

Attributes ¶

id : string

Unique identifier for the client

JAVA

```
ClientService clientService = paymillContext.getClientService();
Client client = clientService.get("client_88a388d9dd48f86c3136");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.clients.details('client_88a388d9dd48f86c3136',
    function(err, client) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("client id " + client.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
client = p.getclientdetails(cid='client_88a388d9dd48f86c3136')
```

RUBY

```
Paymill::Client.find "client_88a388d9dd48f86c3136"
```

.NET

```
ClientService clientService = paymillContext.ClientService;
Client client = clientService.GetAsync("client_88a388d9dd48f86c3136")
.Result;
```

JS

```
pm.clients.detail("client_88a388d9dd48f86c3136").then(function(client
) {
    console.log("client:" + client.id);
}, function(error) {
    console.log("couldnt get client:" + error);
});
```

Response


```
{
    "data" : {
        "id"           : "client_88a388d9dd48f86c3136",
        "email"        : "client@example.com",
        "description"  : "Lovely Client",
        "created_at"   : 1342438695,
        "updated_at"   : 1342438695,
        "payment"      : "[ <Object>, ... ] or null",
        "subscription" : "<Object>",
        "app_id"       : null
    },
    "mode" : "test"
}
```


Sub objects

- client.payment returns payment objects for credit card or direct debit
- client.subscription returns a subscription object

Update `Client` ¶

This function updates the data of a client. To change only a specific attribute you can set this attribute in the update request. All other attributes that shouldn't be edited aren't inserted. You can only edit the description, email and credit card. The subscription can't be changed by updating the client data. This has to be done in the subscription call.

Attributes ¶

id : string
Unique identifier for the client

email : string or null
mail address of the client.

description : string or null
Description for the client

Request

CURL

```
curl https://api.paymill.com/v2/clients/client_88a388d9dd48f86c3136 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "email=lovely-client@example.com" \
  -d "description=My Lovely Client" \
  -X PUT
```

PHP

```
$client = new Paymill\Models\Request\Client();
$client->setId('client_88a388d9dd48f86c3136')
       ->setEmail('updated-client@example.com')
       ->setDescription('Updated Client');

$response = $request->update($client);
```

JAVA

```
ClientService clientService = paymillContext.getClientService();
Client client = clientService.get("client_88a388d9dd48f86c3136");
client.setDescription("My Lovely Client");
clientService.update( client );
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.clients.update('client_88a388d9dd48f86c3136',
    {
     email: 'lovely-client@example.com',
        description: 'Most awesome client EVAR'
 },
    function(err, client) {
        if (err) {
            console.log("Couldn't update the client record");
            return;
        }
        console.log("client id " + client.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
client = p.updateclient(
    cid=self.client['data']['id'],
    email='lovely-client@example.com',
    description='Most awesome client EVAR'
)
```

RUBY

```
Paymill::Client.update_attributes "client_88a388d9dd48f86c3136",
    description: "My Lovely Client"
# or
client = Paymill::Client.create email: "lovely-client@example.com",
    description: "Lovely Client"
client.update_attributes description: "My Lovely Client"
```

.NET

```
ClientService clientService = paymillContext.ClientService;
Client client = clientService.GetAsync("client_88a388d9dd48f86c3136")
.Result;
client.Description = "My Lovely Client";
clientService.UpdateAsync( client ).Result;
```

JS

```
pm.clients.detail("client_88a388d9dd48f86c3136").then(function(client
) {
 client.description = "My Updated Lovely Client";
 return pm.clients.update(client);
}).then(function(updatedClient) {
 console.log("updated client:" + updatedClient.description);
}, function(error) {
 console.log("couldnt update client:" + error);
});
```

Response

```
{
    "data" : {
        "id"           : "client_88a388d9dd48f86c3136",
        "email"        : "lovely-client@example.com",
        "description"  : "My Lovely Client",
        "created_at"   : 1342438695,
        "updated_at"   : 1342439774,
        "payment"      : "[ <Object>, ... ] or null",
        "subscription" : "<Object>",
        "app_id"       : null
    },
    "mode" : "test"
}
```

Sub objects

- client.payment returns payment objects for credit card or direct debit
- client.subscription returns a subscription object

Remove `client` ¶

This function deletes a client, but your transactions aren't deleted.

Request

CURL

```
curl https://api.paymill.com/v2/clients/client_88a388d9dd48f86c3136 \
  -u <YOUR_PRIVATE_KEY>: \
  -X DELETE
```

PHP

```
$client = new Paymill\Models\Request\Client();
$client->setId('client_88a388d9dd48f86c3136');

$response = $request->delete($client);
```

JAVA

```
ClientService clientService = paymillContext.getClientService();
clientService.delete("client_88a388d9dd48f86c3136");
```

Attributes ¶

id : string
Unique identifier for the client

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.clients.remove('client_88a388d9dd48f86c3136',
    function(err, client) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("deleted the client");
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
response = p.delclient('client_88a388d9dd48f86c3136')
```

RUBY

```
Paymill::Client.delete "client_88a388d9dd48f86c3136"
```

.NET

```
ClientService clientService = paymillContext.ClientService;
clientService.DeleteAsync("client_88a388d9dd48f86c3136").Result;
```

JS

```
pm.clients.remove("client_88a388d9dd48f86c3136").then(function(client
) {
 console.log("deleted client:" + client.id);
}, function(error) {
 console.log("couldnt get transaction:" + error);
});
```

Response


```
{
    "data": {
        "id"           : "client_88a388d9dd48f86c3136",
        "email"        : "client@example.com",
        "description"  : "Lovely Client",
        "created_at"   : 1342438695,
        "updated_at"   : 1342438695,
        "payment"      : "[ <Object>, ... ] or null",
        "subscription" : "<Object>",
        "app_id"       : null
    },
    "mode" : "test"
}
```


Sub objects

- client.payment returns payment objects for credit card or direct debit
- client.subscription returns a subscription object

List `Clients` ¶

This function returns a JSON object with a list of clients. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `count`
- `offset`
- `creditcard`
- `email`
- `created_at`

Available filters:

- `payment=<payment id>`
- `subscription=<subscription id>`
- `offer=<offer id>`
- `description=<string>`
- `email=<email>`
- `created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`
- `updated_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`

Request

CURL

```
curl https://api.paymill.com/v2/clients \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$client = new Paymill\Models\Request\Client();

$response = $request->getAll($client);
```

JAVA

```
 ClientService clientService = paymillContext.getClientService();
 PaymillList<Client> clients = clientService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.clients.list({},
    function(err, client) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("client data " + payments.data);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
clients = p.getclients()
```

RUBY

```
Paymill::Client.all
```

.NET

```
ClientService clientService = paymillContext.ClientService;
PaymillList<Client> clients = clientService.ListAsync().Result;
```

JS

```
pm.clients.list().then(function(pmlist) {
 console.log(pmlist.items.length + " clients from total of " + pmlist
.count);
}, function(error) {
 console.log("couldnt list clients:" + error);
});
```

Response

```
{
    "data" : [
        {
            "id"          : "client_bc798246e32ce7e66dbe",
            "email"       : null,
            "description" : null,
            "created_at"  : 1342427064,
```

```
            "updated_at"   : 1342427064,
            "payment"      : "[ <Object>, ... ] or null",
            "subscription" : "<Object>",
            "app_id"       : null
        }
    ],
    "data_count" : "1",
    "mode" : "test"
}
```

Sub objects

- client.payment returns payment objects for credit card or direct debit
- client.subscription returns a subscription object

Export `Client` List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of clients. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `created_at`
- `description`
- `email`
- `updated_at`

Available filters:

- `created_at`
- `description=<string>`
- `email=<email>`
- `offer=<offer id>`
- `payment=<payment id>`
- `subscription=<subscription id>`
- `updated_at`

Request

CURL

```
curl https://api.paymill.com/v2/clients \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response

```
"id";"email";"description";"app_id";"updated_at";"created_at";"paymen
t";"subscription"
"client_33c8f8c13d759d00b144";"testclient@paymill.de";"test client";"
";"1342427064";"1342427064";"pay_2311e5a076ab0b9c2cdb0399";"sub_c84aa
dd0c1c7529158ee,sub_c36362f70bb78d53e145,sub_11cc72a3a759d5ce7f47"
```

Offers ¶

An offer is a recurring plan which a user can subscribe to. You can create different offers with different plan attributes e.g. a monthly or a yearly based paid offer/plan.

`Offer` Object ¶

Attributes ¶

id : string
Unique identifier of this offer

name : string
Your name for this offer

amount : integer (>0)
Every **`interval`** the specified amount will be charged. Only integer values are allowed (e.g. 42.00 = 4200)

interval : string
Defining how often the client should be charged. Format: number DAY | WEEK | MONTH | YEAR Example: 2 DAY

trial_period_day.. integer or null
Define an optional trial period in number of days

created_at : integer
Unix-Timestamp for the creation Date

updated_at : integer
Unix-Timestamp for the last update

subscription_co... subscription_count
Attributes: (integer) if zero, else (string) **`active`**, (integer) if zero, else (string) **`inactive`**

app_id : string or null
App (ID) that created this offer or null if created by yourself.

Example


```
{
    "id" : "offer_40237e20a7d5a231d99b",
    "name" : "Nerd Special",
    "amount" : 4200,
    "currency": "EUR",
    "interval" : "1 WEEK",
    "trial_period_days" : 0,
    "created_at" : 1341935129,
    "updated_at" : 1341935129,
    "subscription_count": {
        "active": "3",
        "inactive": 0
    },
    "app_id": null
}
```

Create new `Offer` ¶

With this call you can create an offer via the API. You can also create an offer with the Merchant Centre.

Attributes ¶

amount : integer (>0)
Amount (in cents)

currency : string
ISO 4217 formatted currency code

interval : string
Defining how often the client should be charged. Format: number DAY|WEEK|MONTH|YEAR Example: 2 DAY

name : string
Your name for this offer

trial_period_day.. integer or null
Define an optional trial period in number of days

Request

CURL

```
curl https://api.paymill.com/v2/offers \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "interval=1 WEEK" \
  -d "name=Nerd Special"
```

PHP

```
$offer = new Paymill\Models\Request\Offer();
$offer->setAmount(4200)
      ->setCurrency('EUR')
      ->setInterval('1 WEEK')
      ->setName('Nerd Special');

$response = $request->create($offer);
```

JAVA

```
OfferService offerService = paymillContext.getOfferService();
Offer offer = offerService.create("4200", "EUR", "1 WEEK", "Nerd Spec
ial", 0);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.offers.create(
    {
        amount: 4200,
        currency: 'EUR',
        interval: 'week',
        name: 'Nerd Special'
    },
    function(err, offer) {
        if (err) {
            console.log("Couldn't create the offer record");
            return;
        }
        console.log("offer id " + offer.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
offer = p.newoffer(
    amount=4200,
    interval='week',
    currency='EUR',
    name='Nerd Special'
)
```

RUBY

```
Paymill::Offer.create amount: 4200, currency: "EUR",
    interval: "1 WEEK", name: "Nerd Special"
```

.NET

```
OfferService offerService = paymillContext.OfferService;
Offer offer = offerService.CreateAsync("4200", "EUR", "1 WEEK", "Nerd
```

```
Special", 0).Result;
```

JS

```
pm.offers.create(4200, "EUR", new pm.OfferInterval(1, pm.OfferInterva
l.Period.WEEK), "Nerd Special").then(function(offer) {
    console.log("offer:" + offer.id);
}, function(error) {
    console.log("couldnt get client:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "offer_40237e20a7d5a231d99b",
        "name" : "Nerd Special",
        "amount" : "4200",
        "currency": "EUR",
        "interval" : "1 WEEK",
        "trial_period_days" : 0,
        "created_at" : 1341935129,
        "updated_at" : 1341935129,
        "subscription_count": {
            "active": "3",
            "inactive": 0
        },
        "app_id": null
    },
    "mode" : "test"
}
```


`Offer` Details ¶

Getting detailed information about an offer requested with the offer ID.

Request

CURL

```
curl https://api.paymill.com/v2/offers/offer_40237e20a7d5a231d99b \
  -u <YOUR_PRIVATE_KEY>: \
```

PHP

```
$offer = new Paymill\Models\Request\Offer();
$offer->setId('offer_40237e20a7d5a231d99b');

$response = $request->getOne($offer);
```

JAVA

```
OfferService offerService = paymillContext.getOfferService();
Offer offer = offerService.get("offer_40237e20a7d5a231d99b");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.offers.details('offer_40237e20a7d5a231d99b',
    function(err, offer) {
        if (err) {
            console.log("Error :(");
```

Attributes ¶

id : string
Unique identifier for the offer

```
            return;
        }
        console.log("offer id " + offer.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
client = p.getofferdetails(oid='offer_40237e20a7d5a231d99b')
```

RUBY

```
Paymill::Offer.find "offer_40237e20a7d5a231d99b"
```

.NET

```
OfferService offerService = paymillContext.OfferService;
Offer offer = offerService.GetAsync("offer_40237e20a7d5a231d99b").Res
ult;
```

JS

```
pm.offers.detail("offer_40237e20a7d5a231d99b").then(function(offer) {
 console.log("offers:" + offer.id);
}, function(error) {
 console.log("couldnt get offer:" + error);
});
```

Response


```
{
    "data": {
        "id" : "offer_40237e20a7d5a231d99b",
        "name" : "Nerd Special",
        "amount" : 4200,
        "currency": "EUR",
        "interval" : "1 WEEK",
        "trial_period_days" : 0,
        "created_at" : 1341935129,
        "updated_at" : 1341935129,
        "subscription_count": {
            "active": 3,
            "inactive": 0
        },
        "app_id": null
    },
    "mode" : "test"
}
```


Update `offer` ¶

Updates the offer. If the offer is assigned to a client only the name can be changed all other attributes cannot be edited.

Request

CURL

```
curl https://api.paymill.com/v2/offers/offer_40237e20a7d5a231d99b \
  -u <YOUR_PRIVATE_KEY>: \
  -d "name=Extended Special" \
  -X PUT
```

Attributes ¶

id : string
Unique identifier for the offer

name : string
Your name for this offer

PHP

```php
$offer = new Paymill\Models\Request\Offer();
$offer->setId('offer_40237e20a7d5a231d99b')
      ->setName('Extended Special');

$response = $request->update($offer);
```

JAVA

```java
OfferService offerService = paymillContext.getOfferService();
Offer offer = offerService.get("offer_40237e20a7d5a231d99b");
offer.setName("Extended Special");
offerService.update( offer );
```

NODE.JS

```node
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.offers.update('offer_40237e20a7d5a231d99b',
    {
        name: 'Extended Special'
    },
    function(err, offer) {
        if (err) {
            console.log("Couldn't update the offer record");
            return;
        }
        console.log("offer id " + offer.data.id);
    }
);
```

PYTHON

```python
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
offer = p.updateoffer(
    oid='offer_40237e20a7d5a231d99b',
    name='Extended Special'
)
```

RUBY

```ruby
Paymill::Offer.update_attributes "offer_40237e20a7d5a231d99b",
    name: "Extended Special"
# or
offer = Paymill::Offer.create amount: 4200, currency: "EUR",
    interval: "1 WEEK", name: "Nerd Special"
offer.update_attributes name: "Extended Special"
```

.NET

```dotnet
OfferService offerService = paymillContext.OfferService;
Offer offer = offerService.GetAsync("offer_40237e20a7d5a231d99b").Res
ult;
offer.Name = "Extended Special";
offerService.UpdateAsync( offer ).Result;
```

JS

```javascript
pm.offers.detail("offer_40237e20a7d5a231d99b").then(function(offer) {
    offer.name = "Extended Special";
    return pm.offers.update(offer);
}).then(function(updatedOffer) {
    console.log("updated offer:" + updatedOffer.description);
}, function(error) {
    console.log("couldnt update offer:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "offer_40237e20a7d5a231d99b",
        "name" : "Extended Special",
        "amount" : 4200,
        "currency": "EUR",
        "interval" : "1 WEEK",
        "trial_period_days" : 0,
        "created_at" : 1341935129,
        "updated_at" : 1341935129,
        "subscription_count": {
            "active": "3",
            "inactive": 0
        },
        "app_id": null
    },
    "mode" : "test"
}
```


Remove `Offer` ¶

You only can delete an offer if no client is subscribed to this offer.

Request

CURL

```
curl https://api.paymill.com/v2/offers/offer_40237e20a7d5a231d99b \
  -u <YOUR_PRIVATE_KEY>: \
  -X DELETE
```

PHP

```
$offer = new Paymill\Models\Request\Offer();
$offer->setId('offer_40237e20a7d5a231d99b');

$response = $request->delete($offer);
```

JAVA

```
OfferService offerService = paymillContext.getOfferService();
offerService.delete("offer_40237e20a7d5a231d99b");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.offers.remove('offer_88a388d9dd48f86c3136',
    function(err, offer) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("deleted the offer");
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
p.deloffer(oid='offer_40237e20a7d5a231d99b')
```

Attributes ¶

id : string
Unique identifier for the offer

RUBY

```ruby
Paymill::Offer.delete "offer_40237e20a7d5a231d99b"
```

.NET

```dotnet
OfferService offerService = paymillContext.OfferService;
offerService.DeleteAsync("offer_40237e20a7d5a231d99b").Result;
```

JS

```javascript
pm.offers.remove("offer_40237e20a7d5a231d99b").then(function(offer) {
 console.log("deleted offer:" + offer.id);
}, function(error) {
 console.log("couldnt get offer:" + error);
});
```

Response

```
{
    "data":[

    ],
    "mode" : "test"
}
```

List `Offers` ¶

This function returns a JSON object with a list of offers. In which order this list are returned depends on the optional parameter order. The following parameters can be used:

- **`count`**
- **`offset`**
- **`interval`**
- **`amount`**
- **`created_at`**
- **`trial_period_days`**

Available filters:

- **`name=<name>`**
- **`trial_period_days=<integer>`**
- **`amount=[>|<]<integer>`** e.g. “300” or with prefix: “>300” or “<300”
- **`created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`**
- **`updated_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`**

Request

CURL

```curl
curl https://api.paymill.com/v2/offers \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```php
$offer = new Paymill\Models\Request\Offer();

$response = $request->getAll($offer);
```

JAVA

```java
OfferService offerService = paymillContext.getOfferService();
PaymillList<Offer> offers = offerService.list();
```

NODE.JS

```node
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.offers.list({},
    function(err, offer) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("offer data " + offer.data);
    }
```

```
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
offers = p.getoffers()
```

RUBY

```
Paymill::Offer.all
```

.NET

```
OfferService offerService = paymillContext.OfferService;
PaymillList<Offer> offers = offerService.ListAsync().Result;
```

JS

```
pm.offers.list().then(function(pmlist) {
 console.log(pmlist.items.length + " offers from total of " + pmlist.
count);
}, function(error) {
 console.log("couldnt list offers:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id" : "offer_40237e20a7d5a231d99b",
            "name" : "Nerd Special",
            "amount" : 4200,
            "currency": "EUR",
            "interval" : "1 WEEK",
            "trial_period_days" : 0,
            "created_at" : 1341935129,
            "updated_at" : 1341935129,
            "subscription_count": {
                "active": "3",
                "inactive": 0
            },
            "app_id": null
        }
    ],
    "data_count" : "1",
    "mode" : "test",
}
```


Export `Offers` List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of offers. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `amount`
- `created_at`
- `currency`

Request

CURL

```
curl https://api.paymill.com/v2/offers \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

- **interval**
- **name**
- **trial_period_days**
- **updated_at**

Available filters:

- **amount**
- **created_at**
- **currency**
- **interval**
- **name**
- **trial_period_days**
- **updated_at**

Subscriptions ¶

Subscriptions allow you to charge recurring payments on a client's credit card / to a client's direct debit. A subscription connects a client to the offers-object. A client can have several subscriptions to different offers, but only one subscription to the same offer.

Subscription Object ¶

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response


```
"id";"name";"amount";"currency";"interval";"trial_period_days";"created_at";"updated_at";"subscription_count_active";"subscription_count_inactive";"subscription_count_app_id"
"offer_1a5d80dc75db9b5c0c64";"Example Offer";"499";"EUR";"3 WEEK";"22";"1342427064";"1342427064";"1";"8";""
```


Example


```
{
    "id" : "sub_012db05186ccfe22d86c",
    "offer" : "<Object>",
    "livemode" : false,
    "cancel_at_period_end" : false,
    "trial_start" : null,
```

Attributes ¶

id :	string Unique identifier of this subscription.
offer :	offer object
livemode :	boolean Whether this subscription was issued while being in live mode or not.
cancel_at_perio...	boolean Cancel this subscription immediately or at the end of the current period?
trial_start :	integer or null Unix-Timestamp for the trial period start
trial_end :	integer or null Unix-Timestamp for the trial period end.
next_capture_at :	integer Unix-Timestamp for the next charge.
created_at :	integer Unix-Timestamp for the creation Date.
updated_at :	integer Unix-Timestamp for the last update.
canceled_at :	integer or null Unix-Timestamp for the cancel date.
payment :	payment object for credit card or payment object for direct debit
mandate_refere..	string or null SEPA mandate reference, can be optionally specified for direct debit transactions. If specified for other payment methods, it has no effect but must still be valid. If specified, the string must not be empty, can be up to 35 characters long and may contain digits 0-9 letters a-z A-Z allowed special characters: ' , . : + - / ( ) ?
client :	client object
app_id :	string or null App (ID) that created this subscription or null if created by yourself.

This function connects the offer with a client.

```
    "trial_end" : null,
    "next_capture_at" : 1369563095,
    "created_at" : 1341935490,
    "updated_at" : 1341935490,
    "canceled_at" : null,
    "payment" : "<Object>",
    "mandate_reference" : null,
    "client" : "<Object>",
    "app_id" : null
}
```

Sub objects

- subscription.offer returns an offer object
- subscription.payment returns a payment object for credit card or a payment object for direct debit
- subscription.client returns a client object

Create new `Subscription` ¶

This function creates a subscription between a client and an offer. A client can have several subscriptions to different offers, but only one subscription to the same offer. The clients is charged for each billing interval entered.

Request

CURL

```
curl https://api.paymill.com/v2/subscriptions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "client=client_64b025ee5955abd5af66" \
  -d "offer=offer_40237e20a7d5a231d99b" \
  -d "payment=pay_95ba26ba2c613ebb0ca8"
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setClient('client_88a388d9dd48f86c3136')
             ->setOffer('offer_40237e20a7d5a231d99b')
             ->setPayment('pay_95ba26ba2c613ebb0ca8');
```

Attributes ¶

offer :	string

Unique offer identifier

payment: string
Unique payment identifier

client: string
Unique client identifier. If not provided the client from the payment is being used.

start_at: integer or null
Unix-Timestamp for the subscription start date

mandate_refere.. string or null
SEPA mandate reference, can be optionally specified for direct debit transactions. If specified for other payment methods, it has no effect but must still be valid. If specified, the string must not be empty, can be up to 35 characters long and may contain
digits 0-9
letters a-z A-Z
allowed special characters: ‘ , . : + - / ( ) ?

```
$response = $request->create($subscription);
```

JAVA

```
ClientService clientService = paymillContext.getClientService();
Client client = clientService.get("client_64b025ee5955abd5af66");
OfferService offerService = paymillContext.getOfferService();
Offer offer = offerService.get("offer_40237e20a7d5a231d99b");
PaymentService paymentService = paymillContext.getPaymentService();
Payment payment = paymentService.get("pay_95ba26ba2c613ebb0ca8");
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
Subscription subscription = subscriptionService.createWithOfferPaymen
tAndClient(
    offer,
    payment,
    client
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.subscriptions.create(
    {
     client: 'client_64b025ee5955abd5af66',
     offer: 'offer_40237e20a7d5a231d99b',
     payment: 'pay_95ba26ba2c613ebb0ca8'
 },
    function(err, subscription) {
        if (err) {
            console.log("Couldn't create the subscription record");
            return;
        }
        console.log("subscription id " + subscription.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
subscription = p.newsub(
    client='client_88a388d9dd48f86c3136',
    offer='offer_40237e20a7d5a231d99b',
    payment='pay_95ba26ba2c613ebb0ca8'
)
```

RUBY

```
Paymill::Subscription.create client: "client_64b025ee5955abd5af66",
    offer: "offer_40237e20a7d5a231d99b",
    payment: "pay_95ba26ba2c613ebb0ca8"
```

.NET

```
ClientService clientService = paymillContext.ClientService;
Client client = clientService.GetAsync("client_64b025ee5955abd5af66")
.Result;
OfferService offerService = paymillContext.OfferService;
Offer offer = offerService.GetAsync("offer_40237e20a7d5a231d99b").Res
ult;
PaymentService paymentService = paymillContext.PaymentService;
Payment payment = paymentService.GetAsync("pay_95ba26ba2c613ebb0ca8")
.Result;
SubscriptionService subscriptionService = paymillContext.Subscription
Service;
Subscription subscription = subscriptionService.CreateWithOfferPaymen
tAndClientAsync(
    offer,
    payment,
```

```
    client
).Result;
```

JS

```
pm.subscriptions.create("offer_40237e20a7d5a231d99b", "pay_95ba26ba2c
613ebb0ca8", "client_64b025ee5955abd5af66").then(function(subscriptio
n) {
 console.log("created subscription:" + subscription.id);
}, function(error) {
 console.log("couldnt get subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_012db05186ccfe22d86c",
        "offer" : "<Object>",
        "livemode" : false,
        "cancel_at_period_end" : false,
        "trial_start" : null,
        "trial_end" : null,
        "next_capture_at" : 1369563095,
        "created_at" : 1341935490,
        "updated_at" : 1341935490,
        "canceled_at" : null,
        "payment" : "<Object>",
        "mandate_reference" : null,
        "client" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```

Sub objects

- subscription.offer returns an offer object
- subscription.payment returns a payment object for credit card or a payment object for direct debit
- subscription.client returns a client object

`Subscription` Details ¶

This function returns the detailed information of the concrete requested subscription.

Request

CURL

```
curl https://api.paymill.com/v2/subscriptions/sub_dc180b755d10da32486
4 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dc180b755d10da324864');

$response = $request->getOne($subscription);
```

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
```

Attributes ¶

id : string
Unique identifier for the subscription

```
ionService();
Subscription subscription = subscriptionService.get("sub_dc180b755d10
da324864");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.subscriptions.details('sub_dc180b755d10da324864',
    function(err, subscription) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("subscription id " + subscription.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
subscription = p.getsubdetails(sid='sub_dc180b755d10da324864')
```

RUBY

```
Paymill::Subscription.find "sub_dc180b755d10da324864"
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;
Subscription subscription = subscriptionService.GetAsync("sub_dc180b7
55d10da324864").Result;
```

JS

```
pm.subscriptions.detail("sub_dc180b755d10da324864").then(function(sub
scription) {
 console.log("subscription:" + subscription.id);
}, function(error) {
 console.log("couldnt get subscription:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "sub_dc180b755d10da324864",
        "offer" : "<Object>",
        "livemode" : false,
        "cancel_at_period_end" : false,
        "trial_start" : null,
        "trial_end" : null,
        "next_capture_at" : 1369563095,
        "created_at" : 1341935490,
        "updated_at" : 1341935490,
        "canceled_at" : null,
        "payment" : "<Object>",
        "mandate_reference" : null,
        "client" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```


Sub objects

- subscription.offer returns an offer object
- subscription.payment returns a payment object for credit card or a payment object for direct debit
- subscription.client returns a client object

Update `Subscription` ¶

This function updates the subscription of a client. You can change e.g. the cancel_at_period_end attribute to terminate a subscription either immediately or after the next payment. Or you can assign the subscription to another offer (offer=<new_offer_id>).

Attributes ¶

id :	string Unique identifier for the subscription
cancel_at_perio...	boolean To terminate a subscription either immediately or after the next payment.
offer :	string Unique identifier describing the offer which is subscribed to the client (optional)
payment :	string Unique identifier describing a payment of the client
mandate_refere...	string or null SEPA mandate reference, can be optionally specified for direct debit transactions. If specified for other payment methods, it has no effect but must still be valid. If specified, the string must not be empty, can be up to 35 characters long and may contain digits 0-9 letters a-z A-Z allowed special characters: ‘ , . : + - / ( ) ?

Request

CURL

```
curl https://api.paymill.com/v2/subscriptions/sub_dc180b755d10da32486
4 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "cancel_at_period_end=true" \
  -d "offer=offer_40237e20a7d5a231d99b" \
  -d "payment=pay_95ba26ba2c613ebb0ca8" \
  -X PUT
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dc180b755d10da324864')
             ->setOffer('offer_40237e20a7d5a231d99b')
             ->setPayment('pay_95ba26ba2c613ebb0ca8')
             ->setCancelAtPeriodEnd(false);

$response = $request->update($subscription);
```

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
Subscription subscription = subscriptionService.get(
    "sub_dc180b755d10da324864"
);
subscription.setCancelAtPeriodEnd( true );
subscriptionService.update( subscription );
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.subscriptions.update('sub_dc180b755d10da324864',
    {
     cancel_at_period_end: true,
     offer: 'offer_40237e20a7d5a231d99b'
 },
    function(err, subscription) {
        if (err) {
            console.log("Couldn't update the subscription record");
            return;
        }
        console.log("subscription id " + subscription.data.id);
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
response = p.cancelsubafter(
```

```
    sid='sub_dc180b755d10da324864',
    cancel=True
)
```

RUBY

```
Paymill::Subscription.update_attributes "sub_6d2d5aa2c9da6cf096b7",
    offer: "offer_1b254ef5bdb90a5807d5"
# or
subscription = Paymill::Subscription.create client: "client_64b025ee5
955abd5af66",
    offer: "offer_40237e20a7d5a231d99b",
    payment: "pay_95ba26ba2c613ebb0ca8"
subscription.update_attributes offer: "offer_1b254ef5bdb90a5807d5"
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;
Subscription subscription = subscriptionService.GetAsync(
    "sub_dc180b755d10da324864"
).Result;
subscription.CancelAtPeriodEnd = true ;
subscriptionService.UpdateAsync( subscription ).Result;
```

JS

```
pm.subscriptions.detail("sub_dc180b755d10da324864").then(function(sub
scription) {
 subscription.payment = "pay_917018675b21ca03c4fb";
 return pm.subscriptions.update(subscription);
}).then(function(updatedSubscription) {
 console.log("updated subscription:" + updatedSubscription.descriptio
n);
}, function(error) {
 console.log("couldnt update subscription:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "sub_dc180b755d10da324864",
        "offer" : "<Object>",
        "livemode" : false,
        "cancel_at_period_end" : true,
        "trial_start" : null,
        "trial_end" : null,
        "next_capture_at" : 1369563095,
        "created_at" : 1341935490,
        "updated_at" : 1341935490,
        "canceled_at" : null,
        "payment" : "<Object>",
        "mandate_reference" : null,
        "client" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```


Sub objects

- subscription.offer returns an offer object
- subscription.payment returns a payment object for credit card or a payment object for direct debit
- subscription.client returns a client object

Remove `Subscription` ¶

This function removes an existing subscription. The subscription will be directly terminated but pending transactions will still be charged.

Attributes ¶

id : string
Unique identifier for the subscription

Request

CURL

```
curl https://api.paymill.com/v2/subscriptions/sub_dc180b755d10da32486
4 \
  -u <YOUR_PRIVATE_KEY>: \
  -X DELETE
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dc180b755d10da324864');

$response = $request->delete($subscription);
```

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.delete("sub_dc180b755d10da324864");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.subscriptions.remove('sub_dc180b755d10da324864',
    function(err, subscription) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("deleted the subscription");
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
response = p.cancelsubnow(sid='sub_012db05186ccfe22d86c')
```

RUBY

```
Paymill::Subscription.delete "sub_dc180b755d10da324864"
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;
subscriptionService.DeleteAsync("sub_dc180b755d10da324864").Result;
```

JS

```
pm.subscriptions.detail("sub_dc180b755d10da324864").then(function(sub
scription) {
 console.log("deleted subscription:" + subscription.id);
}, function(error) {
 console.log("couldnt get subscription:" + error);
});
```

Response


```
{
    "data" : {
        "id" : "sub_dc180b755d10da324864",
        "offer" : "<Object>",
        "livemode" : false,
        "cancel_at_period_end" : false,
        "trial_start" : null,
        "trial_end" : null,
        "next_capture_at" : 1369563095,
        "created_at" : 1341935490,
        "updated_at" : 1349948303,
        "canceled_at" : 1349948303,
        "payment" : "<Object>",
        "mandate_reference" : null,
        "client" : "<Object>",
        "app_id" : null
    },
    "mode" : "test"
}
```


Sub objects

- subscription.offer returns an offer object
- subscription.payment returns a payment object for credit card or a payment object for direct debit
- subscription.client returns a client object

List **Subscriptions** ¶

This function returns a JSON object with a list of subscriptions. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- **count**
- **offset**
- **offer**
- **canceled_at**
- **created_at**

Available filters:

- **offer=<offer id>**
- **created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>**

Request

CURL

```
curl https://api.paymill.com/v2/subscriptions \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();

$response = $request->getAll($subscription);
```

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
PaymillList<Subscription> subscriptions = subscriptionService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.subscriptions.list({},
    function(err, subscription) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("subscription data " + subscription.data);
```

```
    }
);
```

PYTHON

```
private_key = '<YOUR_PRIVATE_KEY>'
p = pymill.Pymill(private_key)
subscriptions = p.getsubs()
```

RUBY

```
Paymill::Subscription.all
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;
PaymillList<Subscription> subscriptions = subscriptionService.ListAsy
nc().Result;
```

JS

```
pm.subscriptions.list().then(function(pmlist) {
 console.log(pmlist.items.length + " offers from total of " + pmlist.
count);
}, function(error) {
 console.log("couldnt list subscriptions:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id" : "sub_dc180b755d10da324864",
            "offer" : "<Object>",
            "livemode" : false,
            "cancel_at_period_end" : false,
            "trial_start" : null,
            "trial_end" : null,
            "next_capture_at" : 1369563095,
            "created_at" : 1341935490,
            "updated_at" : 1349948303,
            "canceled_at" : 1349948303,
            "payment" : "<Object>",
            "mandate_reference" : null,
            "client" : "<Object>",
            "app_id" : null
        }
    ],
    "data_count" : "1",
    "mode" : "test"
}
```


Sub objects

- subscription.offer returns an offer object
- subscription.payment returns a payment object for credit card or a payment object for direct debit
- subscription.client returns a client object

Export `Subscriptions` List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of subscriptions. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

> - **`created_at`**
> - **`updated_at`**

Available filters:

> - **`offer`**
> - **`currency`**
> - **`created_at`**
> - **`canceled_at`**
> - **`updated_at`**

Request

CURL

```
curl https://api.paymill.com/v2/subscriptions \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response

```
"id";"offer_id";"offer_name";"offer_amount";"offer_currency";"offer_interval";"offer_trial_period_days";"offer_created_at";"offer_updated_at";"offer_subscription_count_active";"offer_subscription_count_inactive";"offer_subscription_count_app_id";"offer_livemode";"offer_cancel_at_period_end";"offer_trial_start";"offer_trial_end";"offer_next_capture_at";"offer_created_at";"offer_updated_at";"offer_canceled_at";"offer_app_id";"offer_payment_id";"offer_payment_type";"offer_payment_client";"offer_payment_card_type";"offer_payment_country";"offer_payment_bin";"offer_payment_expire_month";"offer_payment_expire_year";"offer_payment_card_holder";"offer_payment_last4";"offer_payment_updated_at";"offer_payment_created_at";"offer_payment_app_id";"offer_payment_is_recurring";"offer_payment_is_usable_for_preauthorization";"offer_payment_client_id";"offer_payment_client_email";"offer_payment_client_description";"offer_payment_client_app_id";"offer_payment_client_updated_at";"offer_payment_client_created_at"
"sub_c84aadd0c1c7529158ee";"offer_1a5d80dc75db9b5c0c64";"Example Subscription";"499";"EUR";"3 WEEK";"22";;;"1";"8";"";"";"";"1401983620";"1404575620";"";"1342427064";"1342427064";"1402640050";"";"pay_2311e5a076ab0b9c2cdb0399";"creditcard";"client_33c8f8c13d759d00b144";"visa";"DE";"411111";"2";"2016";"test card holder";"1111";"1342427064";"1342427064";"";"1";"1";"client_33c8f8c13d759d00b144";"testclient@example.com";"test client";"";"1342427064";"1342427064"
```

Webhooks ¶

With webhooks we give you the possibility to react automatically to certain events which happen within our system. A webhook is basically a URL where we send an HTTP POST request to, every time one of the events attached to that webhook is triggered. Alternatively you can define an email address where we send the event's information to You can manage your webhooks via the API as explained below or you can use the web interface inside our Merchant Centre.

Our call to the webhook / email includes a JSON encoded event object with detailed information about the event in it's POST body.

Webhooks Details

- we expect a http status code of 200 in the response of our webhook call.
- every content in the body will be discarded, so you might just leave that blank.
- if we receive another code or a timeout, we will retry to call the same webhook every hour up to five times. emails will be sent only once.
- if the webhook call to one webhook fails 5 times, we automatically deactivate the webhook. You can still see them in your settings.
- the webhook will be called asynchronously within a few minutes after the actual event has happened.

Events ¶

There are a number of events you can react to. Each webhook can be configured to catch any kind of event individually, so you can create different webhooks for different events. Each Webhook needs to be attached to at least one event.

For example the event subscription.succeeded is triggered every time a successful transaction has been made in our system that is based on a subscription. Shortly after that has been triggered, we will call every webhook you defined for this event and send detailed information to it.

Available Events

- chargeback.executed: returns a transaction-object with state set to chargeback
- client.updated: returns a client-object if a client was updated
- transaction.created: returns a transaction-object
- transaction.succeeded: returns a transaction-object
- transaction.failed: returns a transaction-object
- client.updated: returns a client-object if a client was updated
- subscription.created: returns a subscription-object
- subscription.updated: returns a subscription-object
- subscription.deleted: returns a subscription-object
- subscription.succeeded: returns a transaction-object and a subscription-object
- subscription.failed: returns a transaction-object and a subscription-object
- refund.created: returns a refund-object
- refund.succeeded: returns a refunds-object
- refund.failed: returns a refunds-object
- payout.transferred: returns an invoice-object with the payout sum for the invoice period
- invoice.available: returns an invoice-object with the fees sum for the invoice period
- app.merchant.activated: returns a merchant-object if a connected merchant was activated
- app.merchant.deactivated: returns a merchant-object if a connected merchant was deactivated
- app.merchant.rejected: returns a merchant-object if a connected merchant was rejected
- app.merchant.live_requests_allowed: returns a merchant-object if a connected merchant allows live requests
- app.merchant.live_requests_not_allowed: returns a merchant-object if a connected merchant denys live requests
- app.merchant.pm.updated: returns a list of currently active payment methods if they have changed
- app.merchant.app.disabled: returns a merchant object if a connected merchant disabled your app
- payment.expired: returns a payment-object if a creditcard is going to expire next month

Example event

```
{
    "event": {
        "event_type": "subscription.succeeded",
        "event_resource": {
            "subscription": "<Object>",
```

```
            "transaction": "<Object>"
        },
        "created_at": "1358027174",
        "app_id": null
    }
}
```

PHP

```
$body = @file_get_contents('php://input');
$event_json = json_decode($body, true);
```

Webhook Object ¶

Example URL webhook

```
{
    "id":"hook_40237e20a7d5a231d99b",
    "url":"<your-webhook-url>",
    "livemode":false,
    "event_types":[
        "transaction.succeeded",
        "transaction.failed"
    ],
    "created_at":1358982000,
    "updated_at":1358982000,
    "active":true,
    "app_id":null
}
```

Attributes ¶

id : string
Unique identifier of this webhook

url : string
the url of the webhook

email : string
either the email OR the url have to be set and will be returned

livemode : you can create webhooks for livemode and testmode

event_types : array of event_types

active : boolean
if false, no events will be dispatched to this webhook anymore

app_id : string or null
App (ID) that created this webhook or null if created by yourself.

Example e-mail webhook

```
{
    "id":"hook_40237e20a7d5a231d99b",
    "email":"<your-webhook-email>",
    "livemode":false,
    "event_types":[
        "transaction.succeeded",
        "transaction.failed"
    ],
    "created_at":1358982000,
    "updated_at":1358982000,
    "active":true,
    "app_id":null
}
```

Create new **URL Webhook** ¶

With this call you can create a webhook to a url via the API.

Request

CURL

```
curl https://api.paymill.com/v2/webhooks \
-u <YOUR_PRIVATE_KEY>: \
-d "url=<your-webhook-url>" \
```

Attributes ¶

url : string
the url of the webhook

event_types : array
includes a set of webhook event types as strings

active : true|false
can be used to create an inactive webhook in the beginning

```
-d "event_types[]=transaction.succeeded" \
-d "event_types[]=transaction.failed"
```

PHP

```
$webhook = new Paymill\Models\Request\Webhook();
$webhook->setUrl('<your-webhook-url>')
        ->setEventTypes(array(
            'transaction.succeeded',
            'transaction.failed'
        ));

$response = $request->create($webhook);
```

JAVA

```
WebhookService webhookService = paymillContext.getWebhookService();
EventType[] eventTypes = new EventType[] {
    EventType.TRANSACTION_SUCCEEDED,
    EventType.TRANSACTION_FAILED
};
Webhook webhook = webhookService.createUrlWebhook(
    "<your-webhook-url>",
    eventTypes
);
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.webhooks.create(
    {
        url: '<your-webhook-url>',
        event_types: ['subscription.succeeded"', 'subscription.failed
']
    },
    function(err, webhook) {
        if (err) {
            console.log("Couldn't create the webhook record");
            return;
        }
        console.log(webhook.data);
    }
);
```

PYTHON

```
# Not implemented yet
```

RUBY

```
Paymill::Webhook.create url: "<your-webhook-url>",
    event_types: ["transaction.succeeded", "transaction.failed"]
```

.NET

```
WebhookService webhookService = paymillContext.WebhookService;
EventType[] eventTypes = new EventType[] {
    EventType.TRANSACTION_SUCCEEDED,
    EventType.TRANSACTION_FAILED
}.Result;
Webhook webhook = webhookService.CreateUrlWebhookAsync(
    "<your-webhook-url>",
    eventTypes
).Result;
```

JS

```
pm.webhooks.createUrl("<your-webhook-url>",[
```

```
    pm.Webhook.EventType.TRANSACTION_SUCCEDED,
    pm.Webhook.EventType.TRANSACTION_FAILED
]).then(function(webhook) {
    console.log("created webhook:" + webhook.id);
}, function(error) {
    console.log("couldnt get webhook:" + error);
});
```

Response

```
{
    "data" : {
            "id":"hook_40237e20a7d5a231d99b",
            "url":"<your-webhook-url>",
            "livemode":false,
            "event_types":[
                "transaction.succeeded",
                "transaction.failed"
            ],
            "created_at":1358982000,
            "updated_at":1358982000,
            "active" : true,
            "app_id" : null
        },
        "mode" : "test"
}
```

Create new `E-Mail Webhook` ¶

Instead of setting the url parameter you can set the email parameter to create a webhook, where we send mails to in case of an event.

Attributes ¶

email :	string the webhooks email. must be a valid mail address
event_types :	array includes a set of webhook event types as strings
active :	true|false can be used to create an inactive webhook in the beginning

Request

CURL

```
curl https://api.paymill.com/v2/webhooks \
-u <YOUR_PRIVATE_KEY>: \
-d "email=<your-webhook-email>" \
-d "event_types[]=subscription.succeeded" \
-d "event_types[]=subscription.failed"
```

PHP

```
$webhook = new Paymill\Models\Request\Webhook();
$webhook->setEmail('<your-webhook-email>')
        ->setEventTypes(array(
            'transaction.succeeded',
            'transaction.failed'
        ));

$response = $request->create($webhook);
```

JAVA

```
WebhookService webhookService = paymillContext.getWebhookService();
EventType[] eventTypes = new EventType[] {
    EventType.TRANSACTION_SUCCEEDED,
    EventType.TRANSACTION_FAILED
};
Webhook webhook = webhookService.createEmailWebhook(
    "<your-webhook-email>",
    eventTypes
);
```

NODE.JS

```node
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.webhooks.create(
    {
        email: '<your-webhook-email>',
        event_types: ['transaction.succeeded', 'transaction.failed']
    },
    function(err, webhook) {
        if (err) {
            console.log("Couldn't create the webhook record");
            return;
        }
        console.log(webhook.data);
    }
);
```

PYTHON

```python
# Not implemented yet
```

RUBY

```ruby
Paymill::Webhook.create email: "<your-webhook-email>",
    event_types: ["transaction.succeeded", "transaction.failed"]
```

.NET

```dotnet
WebhookService webhookService = paymillContext.WebhookService;
EventType[] eventTypes = new EventType[] {
    EventType.TRANSACTION_SUCCEEDED,
    EventType.TRANSACTION_FAILED
};
Webhook webhook = webhookService.CreateEmailWebhookAsync(
    "<your-webhook-email>",
    eventTypes
).Result;
```

JS

```node
pm.webhooks.createEmail("<your-webhook-email>",[
    pm.Webhook.EventType.TRANSACTION_SUCCEEDED,
    pm.Webhook.EventType.TRANSACTION_FAILED
]).then(function(webhook) {
    console.log("created webhook:" + webhook.id);
}, function(error) {
    console.log("couldnt get webhook:" + error);
});
```

Response


```
{
    "data" : {
            "id" : "hook_40237e20a7d5a231d99b",
            "email" : "<your-webhook-email>",
            "livemode" : false,
            "event_types" : [
                "transaction.succeeded",
                "transaction.failed"
            ],
            "created_at" : 1358982000,
            "updated_at" : 1358982000,
            "active" : true,
            "app_id" : null
        },
        "mode" : "test"
}
```

Webhook Details ¶

Getting detailed information about a webhook requested with the webhook id.

Request

CURL

```
curl https://api.paymill.com/v2/webhooks/hook_40237e20a7d5a231d99b \
-u <YOUR_PRIVATE_KEY>:
```

PHP

```
$webhook = new Paymill\Models\Request\Webhook();
$webhook->setId('hook_40237e20a7d5a231d99b');

$response = $request->getOne($webhook);
```

JAVA

```
WebhookService webhookService = paymillContext.getWebhookService();
Webhook webhook = webhookService.get("hook_40237e20a7d5a231d99b");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.webhooks.details('hook_40237e20a7d5a231d99b',
    function(err, webhook) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("webhook id " + webhook.data.id);
    }
);
```

RUBY

```
Paymill::Webhook.find "hook_40237e20a7d5a231d99b"
```

.NET

```
WebhookService webhookService = paymillContext.WebhookService;
Webhook webhook = webhookService.GetAsync("hook_40237e20a7d5a231d99b"
).Result;
```

JS

```
pm.webhooks.detail("hook_40237e20a7d5a231d99b").then(function(webhook
) {
 console.log("webhook:" + webhook.id);
}, function(error) {
 console.log("couldnt get webhook:" + error);
});
```

Response


```
{
   "data" : {
       "id":"hook_40237e20a7d5a231d99b",
       "url":"<your-webhook-url>",
       "livemode":false,
       "event_types":[
```

```
            "transaction.succeeded",
            "transaction.failed"
        ],
        "created_at":1358982000,
        "updated_at":1358982000,
        "active" : true,
        "app_id" : null
    },
    "mode" : "test"
}
```

Response of an e-mail webhook

```
{
    "data" : {
        "id":"hook_40237e20a7d5a231d99b",
        "email":"<your-webhook-email>",
        "livemode":false,
        "event_types":[
            "transaction.succeeded",
            "transaction.failed"
        ],
        "created_at":1358982000,
        "updated_at":1358982000,
        "active" : true,
        "app_id" : null
    },
    "mode" : "test"
}
```

Update **Webhook** ¶

Updates the webhook. You can change the url/email, the event types and the active state.

Attributes ¶

- id : string — the unique identifier of this webhook
- url : string — the url of the webhook
- email : string — the email for the webhook
- event_types : array of event_types
- active : true|false — activate / deactivate webhook

Request

CURL

```
curl https://api.paymill.com/v2/webhooks/hook_40237e20a7d5a231d99b \
   -u <YOUR_PRIVATE_KEY>: \
   -d "url=<new-webhook-url>" \
   -X PUT
```

PHP

```
$webhook = new Paymill\Models\Request\Webhook();
$webhook->setId('hook_40237e20a7d5a231d99b')
        ->setUrl('<your-webhook-url>')
        ->setEventTypes(array(
            'transaction.failed',
            'subscription.failed'
        ));

$response = $request->update($webhook);
```

JAVA

```
WebhookService webhookService = paymillContext.getWebhookService();
Webhook webhook = webhookService.get("hook_40237e20a7d5a231d99b");
webhook.setUrl("http://www.example.org");
webhookService.update( webhook );
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
```

```
var paymill = require('paymill-node')(api_key);

paymill.webhooks.update('hook_40237e20a7d5a231d99b', {
        url: '<your-webhook-url>'
  }, function(err, webhook) {
        if (err) {
            console.log("Couldn't update the webhook record");
            return;
        }
        console.log("webhook id " + webhook.data.id);
    }
);
```

RUBY

```
Paymill::Webhook.update_attributes "hook_40237e20a7d5a231d99b",
    url: "<new-webhook-url>"
# or
webhook = Paymill::Webhook.create url: "<your-webhook-url>",
    event_types: ["transaction.succeeded", "transaction.failed"]
webhook.update_attributes url: "<new-webhook-url>"
```

.NET

```
WebhookService webhookService = paymillContext.WebhookService;
Webhook webhook = webhookService.GetAsync("hook_40237e20a7d5a231d99b"
).Result;
webhook.Email = "test1@mail.com";
webhookService.UpdateAsync( webhook ).Wait();
```

JS

```
pm.webhooks.detail("hook_40237e20a7d5a231d99b").then(function(webhook
) {
 webhook.email = "<your-udpated-webhook-email>";
 return pm.webhooks.update(webhook);
}).then(function(updatedWebhook) {
 console.log("updated webhook:" + updatedWebhook.description);
}, function(error) {
 console.log("couldnt update webhook:" + error);
});
```

Response


```
{
    "data" : {
        "id":"hook_40237e20a7d5a231d99b",
        "url":"<your-webhook-url>",
        "livemode":false,
        "event_types":[
            "transaction.succeeded",
            "transaction.failed"
        ],
        "created_at":1358982000,
        "updated_at":1358982000,
        "active" : true,
        "app_id" : null
    },
    "mode" : "test"
}
```


Remove **Webhook** ¶

All pending calls to a webhook are deleted as well, as soon as you delete the webhook itself.

Request

CURL

```
curl https://api.paymill.com/v2/webhooks/hook_40237e20a7d5a231d99b \
-u <YOUR_PRIVATE_KEY>: \
-X DELETE
```

PHP

```
$webhook = new Paymill\Models\Request\Webhook();
$webhook->setId('hook_40237e20a7d5a231d99b');

$response = $request->delete($webhook);
```

JAVA

```
WebhookService webhookService = paymillContext.getWebhookService();
Webhook webhook = webhookService.delete("hook_40237e20a7d5a231d99b");
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.webhooks.remove('hook_88a388d9dd48f86c3136',
    function(err, webhook) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("deleted the webhook");
    }
);
```

RUBY

```
Paymill::Webhook.delete "hook_40237e20a7d5a231d99b"
```

.NET

```
WebhookService webhookService = paymillContext.WebhookService;
Webhook webhook = webhookService.DeleteAsync("hook_40237e20a7d5a231d9
9b").Result;
```

JS

```
pm.webhooks.remove("hook_40237e20a7d5a231d99b").then(function(webhook
) {
 console.log("deleted webhook:" + webhook.id);
}, function(error) {
 console.log("couldnt get webhook:" + error);
});
```

Response

```
{
    "data":[

    ],
    "mode" : "test"
}
```

List `Webhooks` ¶

This function returns a JSON object with a list of webhooks. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `count`
- `offset`
- `url`
- `email`
- `created_at`

Available filters:

- `email=<email>`
- `url=<url>`
- `created_at=<timestamp> | <timestamp (from)>-<timestamp (to)>`

Request

CURL

```
curl https://api.paymill.com/v2/webhooks/ \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

```
$webhook = new Paymill\Models\Request\Webhook();
$webhook->setFilter(array(
    'count' => 2,
    'offset' => 0
));

$response = $request->getAll($webhook);
```

JAVA

```
WebhookService webhookService = paymillContext.getWebhookService();
PaymillList<Webhook> webhooks = webhookService.list();
```

NODE.JS

```
var api_key = '<YOUR_PRIVATE_KEY>';  // secret paymill API key
var paymill = require('paymill-node')(api_key);

paymill.webhooks.list({},
    function(err, webhook) {
        if (err) {
            console.log("Error :(");
            return;
        }
        console.log("webhook data " + webhook.data);
    }
);
```

RUBY

```
Paymill::Webhook.all
```

.NET

```
WebhookService webhookService = paymillContext.WebhookService;
PaymillList<Webhook> webhooks = webhookService.ListAsync();
```

JS

```
pm.webhooks.list().then(function(pmlist) {
 console.log(pmlist.items.length + " webhooks from total of " + pmlis
t.count);
}, function(error) {
 console.log("couldnt list webhooks:" + error);
});
```

Response


```
{
    "data" : [
        {
            "id":"hook_40237e20a7d5a231d99b",
            "url":"<your-webhook-url>",
            "livemode":false,
            "event_types":[
                "transaction.succeeded",
```

```
                "transaction.failed"
            ],
            "created_at":1358982000,
            "updated_at":1358982000,
            "active" : true,
            "app_id" : null
        },
        {
            "id":"hook_40237e20a7d5skt6d99b",
            "email":"<your-webhook-email>",
            "livemode":false,
            "event_types":[
                "subscription.succeeded",
                "subscription.failed"
            ],
            "created_at":1358911000,
            "updated_at":1358913000,
            "active" : true,
            "app_id" : null
        }
    ],
    "data_count" : "2",
    "mode" : "test"
}
```


Export `Webhooks` List ¶

This function returns CSV separated by semicolons, encapsulated by double quotes, with a list of webhooks. In which order this list is returned depends on the optional parameter order. The following parameters can be used:

- `created_at`
- `email`
- `updated_at`
- `url`

Available filters:

- `email`
- `url`
- `created_at`

Request

CURL

```
curl https://api.paymill.com/v2/webhooks \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

```
/* Not implemented yet */
```

JAVA

```
/* Not implemented yet */
```

PYTHON

```
# Not implemented yet
```

RUBY

```
# Not implemented yet
```

.NET

```
/* Not implemented yet */
```

JS

```
/* Not implemented yet */
```

Response

```
"id";"livemode";"event_types_created_at";"event_types_updated_at";"event_types_active";"event_types_app_id";"event_types_version";"event_types_email"
"hook_b7542f01b384b2539789";"";"1342427064";"1342427064";"1";"";"2.1";"webhooks@example.de"
```

Internal Objects ¶

Here you find the internal objects which do not have a public API endpoint yet.

Fee Object ¶

To find out how collecting application fees click here.

Example

```
{
    "type": "application",
    "application": "app_1d70acbf80c8c35ce83680715c06be0d15c06be0d",
    "payment": "pay_917018675b21ca03c4fb",
    "amount": 420,
    "currency": "EUR",
    "billed_at": null
}
```

Attributes ¶

type : string
Recipient of the fee

application : string
If App fee, app object ID (optional)

payment : string
Payment object ID from which the fee gets paid

amount : integer
Formatted fee amount

currency : string
ISO 4217 formatted currency code

billed_at : integer
Unix-Timestamp for the creation date

Invoice Object ¶

Example

Attributes ¶

invoice_nr : string
invoice number

netto : integer
Formatted netto amount

brutto : integer
Formatted brutto amount

status : string
Invoice status (e.g. sent, trx_ok, trx_failed, invalid_payment, success, 1st_reminder, 2nd_reminder, 3rd_reminder, suspend, canceled, transferred)

period_from : integer
Unix-Timestamp for the start of this invoice period

period_until : integer
Unix-Timestamp for the end of this invoice period

currency : string
ISO 4217 formatted currency code.

vat_rate : integer
VAT rate of the brutto amount

billing_date : integer
Unix-Timestamp for the billing date

invoice_type : enum(paymill, wirecard, acceptance etc.)
Indicates if it"s a PAYMILL invoice or an acquirer payout.

last_reminder_d.. integer
Unix-Timestamp for last payment reminder

```
{
    "invoice_nr": "1293724",
    "netto": 12399,
    "brutto": 14755,
    "status": "sent",
    "period_from": 1349946151,
    "period_until": 1352538151,
    "currency": "EUR",
    "vat_rate": 19,
    "billing_date": 1353142951,
    "invoice_type": "paymill",
    "last_reminder_date": null
}
```

Merchant Object ¶

Attributes ¶

identifier_key : string
Unique identifier of this merchant.

email : string
email address

locale : string
culture setting

country : string or null
country code

currencies : List of activated currencies (ISO 4217 formatted)
Deprecated. This information is now part of payment_methods

methods : List of activated card brands

Example

```
{
    "identifier_key": "mer_123456789",
    "email": "mail@example.com",
    "locale": "de_DE",
    "country": "DEU",
    "currencies": ["EUR", "GPB"],
    "methods": ["visa", "mastercard"]
}
```

`Payment method` Object ¶

Attributes ¶

type : string
Card brand (e.g. visa, mastercard, amex, elv, sepa etc.)

currency : string
ISO 4217 formatted currency code.

acquirer : string
Acquiring bank enum(wirecard, acceptance, none)

Example

```
{
    "type": "visa",
    "currency": "EUR",
    "acquirer": "wirecard"
}
```