PAYMILL API/V2.1 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/v2.1/
```

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/v2.1/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

```
paymill_context = paymill.PaymillContext('<YOUR PRIVATE API KEY>');
```

RUBY

```
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.
50800	Preauthorization failed.

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 ¶

Example

```
{
    "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
}
```

Attributes ¶

id :	string	Unique identifier for this credit card payment
type :	enum(creditcard,debit)	
client :	client object or null	
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.

Payment Object for direct debit payments ¶

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
}
```

Attributes ¶

id :	string	Unique identifier for this direct debit payment
type :	enum(creditcard,debit)	
client :	client object or null	
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.
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) ¶

Example

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

Attributes ¶

id :	string
	Unique identifier for this direct debit payment
type :	enum(creditcard,debit)
client :	client object or null
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 ... ¶

Token

Request

CURL

```
curl https://api.paymill.com/v2.1/payments \
  -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

```
payment_service = paymill_context.get_payment_service();
payment_with_token = payment_service.create(token='098f6bcd4621d373ca
de4e832627b4f6');
```

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);
});
```

Attributes ¶

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

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.

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/v2.1/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

```
payment_service = paymill_context.get_payment_service();
payment_with_token_and_client = payment_service.create(
    token='098f6bcd4621d373cade4e832627b4f6',
    client_id='client_33baaf3ee3251b083420'
);
```

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"        : "<Object>",
        "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 ... ¶

Attributes ¶

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

Creates a direct debit payment from a given token, if

Token

Request

CURL

```
curl https://api.paymill.com/v2.1/payments \
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=12a46bcd462sd3r3care4e8336ssb4f5"
```

PHP

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

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

you're providing the **client**-property, the payment will be created and subsequently be added to the client.

JAVA

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

```
payment_service = paymill_context.get_payment_service();
payment_with_token = payment_service.create(token='12a46bcd462sd3r3ca
re4e8336ssb4f5');
```

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/v2.1/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

```
payment_service = paymill_context.get_payment_service();
payment_with_token_and_client = payment_service.create(
    token='12a46bcd462sd3r3care4e8336ssb4f5',
    client_id='client_33baaf3ee3251b083420'
);
```

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"        : "<Object>",
        "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/v2.1/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);
    }
);
```

Attributes ¶

id : string
Unique identifier for the payment

PYTHON

```
payment_service = paymill_context.get_payment_service();
payment_details = payment_service.detail(payment_with_token);
```

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)>`

Request

CURL

```
curl https://api.paymill.com/v2.1/payments \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

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

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

- `type=creditcard | debit`

Available status for **card_type**:

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

JAVA

```
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

```
payment_service = paymill_context.get_payment_service();
payments_list = payment_service.list();
```

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/v2.1/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

```
payment_service = paymill_context.get_payment_service();
paymentService.remove(payment_with_token);
```

RUBY

```
payment = Paymill::Payment.find( 'pay_ea98515b29437b046207ea45' )
payment.delete
```

.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/v2.1/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";"card_type";"country";"expire_month";"expire_year";"card_
holder";"last4";"updated_at";"created_at";"app_id";"client_id"
"pay_2311e5a076ab0b9c2cdb0399";"creditcard";"visa";"";"2";"2016";"tes
t card holder";"1111";"1342427064";"1342427064";"";"client_33c8f8c13d
759d00b144"
```

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 ¶

Example

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

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.

Sub objects

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

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

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

Token

Request

CURL

```
curl https://api.paymill.com/v2.1/preauthorizations \
  -u <YOUR_PRIVATE_KEY>: \
  -d "token=098f6bcd4621d373cade4e832627b4f6" \
  -d "amount=4200" \
  -d "currency=EUR" \
  -d "description=description example"
```

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)

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();
Preauthorization preauthorization = preauthorizationService.createWit
hToken(
    "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

```
preauthorization_service = paymill_context.get_preauthorization_servi
ce();
preauthorization_with_token = preauthorization_service.create_with_to
ken(
    token='098f6bcd4621d373cade4e832627b4f6',
    amount=4200,
    currency='EUR',
    description='description example'
);
```

RUBY

```
Paymill::Preauthorization.create(
    token: '098f6bcd4621d373cade4e832627b4f6',
    amount: 4200,
    currency: 'EUR',
    description: 'description example'
)
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;

Preauthorization preauthorization = preauthorizationService.CreateWit
hTokenAsync(
        "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":"preauth_e396d56e773f745dfbd3",
        "amount":"4200",
        "currency":"EUR",
        "description":"description example",
        "status":"closed",
        "livemode":false,
        "created_at":1350324120,
        "updated_at":1350324120,
        "app_id":null,
        "payment""<Obejct>",
        "client":"<Obejct>",
        "transaction":"<Obejct>"
    },
    "mode":"test"
}
```

Sub objects

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

Payment

Request

CURL

```
curl https://api.paymill.com/v2.1/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();
Preauthorization preauthorization = preauthorizationService.createWit
hPayment(
    payment,
    4200,
    "EUR",
    "example description"
);
```

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

```
preauthorization_service = paymill_context.get_preauthorization_servi
ce();
preauthorization_with_payment = preauthorization_service.create_with_
payment_id(
    payment_id ='pay_3af44644dd6d25c820a9',
    amount=4200,
    currency='EUR',
    description='description example'
);
```

RUBY

```
payment = Paymill::Payment.create( token: '098f6bcd4621d373cade4e8326
27b4f6' )

Paymill::Preauthorization.create(
    payment: payment.id,
    amount: 4200,
    currency: 'EUR',
    description: 'description example'
)
```

.NET

```
PaymentService paymentService = paymillContext.PaymentService;

Payment payment = paymentService.CreateWithTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6"
).Result;
```

```
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;
Preauthorization preauthorization = preauthorizationService.CreateWit
hPaymentAsync(
    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":"preauth_0b771c503680c341548e",
        "amount":"4200",
        "currency":"EUR",
        "description":"description example",
        "status":"closed",
        "livemode":false,
        "created_at":1349948920,
        "updated_at":1349948920,
        "app_id":null,
        "payment"":"<Obejct>",
        "client":"<Obejct>",
        "transaction":"<Obejct>"
    },
    "mode":"test"
}
```

Sub objects

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

`Preauthorization` Details ¶

Returns data of a specific preauthorization.

Request

CURL

```
curl https://api.paymill.com/v2.1/preauthorizations/preauth_31eb90495
837447f76b7 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

Attributes ¶

id : string
Unique identifier of this preauthorization

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

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

JAVA

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

NODE.JS

```
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

```
preauthorization_service = paymill_context.get_preauthorization_servi
ce();
preauthorization_details = preauthorization_service.detail(preauthori
zation_with_token);
```

RUBY

```
Paymill::Preauthorization.find( 'preauth_31eb90495837447f76b7' )
```

.NET

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

JS

```
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_0b771c503680c341548e",
        "amount":"4200",
        "currency":"EUR",
        "description":"description example",
        "status":"closed",
        "livemode":false,
        "created_at":1349948920,
        "updated_at":1349948920,
        "app_id":null,
        "payment""<Obejct>",
        "client":"<Obejct>",
        "transaction":"<Obejct>"
    },
    "mode":"test"
}
```

Sub objects

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

Remove **`Preauthorizations`** ¶

This function deletes a preauthorization.

Attributes ¶

id : string
Unique identifier for the preauthorization

Request

CURL

```
curl https://api.paymill.com/v2.1/preauthorizations/preauth_31eb90495
837447f76b7 \
  -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.getP
reauthorizationService();
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

```
preauthorization_service = paymill_context.get_preauthorization_servi
ce();
preauthorization_service.remove(preauthorization_with_token);
```

RUBY

```
preauthorization = Paymill::Preauthorization.find( 'preauth_31eb90495
837447f76b7' )
preauthorization.delete
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;
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/v2.1/preauthorizations \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

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

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

JAVA

```
PreauthorizationService preauthorizationService = paymillContext.getP
reauthorizationService();
PaymillList<Preauthorization> preauthorizations = preauthorizationSer
vice.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

```
preauthorization_service = paymill_context.get_preauthorization_servi
ce();
preauthorizations_list = preauthorization_service.list();
```

RUBY

```
Paymill::Preauthorization.all()
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;
PaymillList<Preauthorization> preauthorizations = preauthorizationSer
vice.ListAsync().Result;
```

JS

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

Response

```
{
    "data" : [
        {
            "id":"preauth_0b771c503680c341548e",
            "amount":"4200",
            "currency":"EUR",
            "description":"description example",
            "status":"closed",
            "livemode":false,
            "created_at":1349948920,
            "updated_at":1349948920,
            "app_id":null,
            "payment""<Obejct>",
            "client":"<Obejct>",
            "transaction":"<Obejct>"
        }
    ],
    "data_count" : "1",
    "mode" : "test"
}
```

Sub objects

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

Request

CURL

following parameters can be used:

- `amount`
- `created_at`
- `updated_at`

Available filters:

- `amount`
- `client`
- `created_at`
- `payment`
- `updated_at`

```
curl https://api.paymill.com/v2.1/preauthorizations \
  -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";"currency";"description";"status";"livemode";"created_a
t";"updated_at";"app_id";"payment_id";"client_id";"transaction_id"
"preauth_595d96437ad81d5ca965";"499";"EUR";"Subscription#sub_5dd7af6f
a6d58c60a4e9";"preauth_subscription";"";"1342427064";"1342427064";"";
"pay_2311e5a076ab0b9c2cdb0399";"client_33c8f8c13d759d00b144";""
```

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 ¶

Example

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

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).
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.
refunds :	list refund objects or null
client :	clients-object or null
currency :	string ISO 4217 formatted currency code.
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.
is_fraud :	boolean The transaction is marked as fraud or not.
invoices :	list PAYMILL invoice where the transaction fees are charged or null.
app_id :	string or null App (ID) that created this transaction or null if created by yourself.
preauthorization :	preauthorizations-object or null
fees :	list App fees or null.
payment :	creditcard-object or directdebit-object or null
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: ‘ , . : + - / ( ) ?
is_refundable :	boolean
	The transaction is refundable or not.
is_markable_as…	boolean
	The transaction is markable is fraud or not.

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.

Payment

Request

CURL

```
curl https://api.paymill.com/v2.1/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"
);
```

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 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.

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

```
transaction_service = paymill_context.get_transaction_service();
transaction_with_payment = transaction_service.create_with_payment_id
(
    payment_id='pay_3af44644dd6d25c820a9',
    amount=4200,
    currency='EUR',
    description='Test Transaction'
);
```

RUBY

```
payment = Paymill::Payment.create( token: '098f6bcd4621d373cade4e8326
27b4f6' )
transaction = Paymill::Transaction.create(
    payment: payment,
    amount: 4200,
    currency: 'EUR',
    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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "app_id" : null,
        "preauthorization" : null,
        "fees" : [],
        "payment" : "<Object>",
        "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/v2.1/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

```
transaction_service = paymill_context.get_transaction_service();
transaction_with_token = transaction_service.create_with_token(
    token='098f6bcd4621d373cade4e832627b4f6',
    amount=4200,
    currency='EUR',
    description='Test Transaction'
);
```

RUBY

```
Paymill::Transaction.create(
    token: '098f6bcd4621d373cade4e832627b4f6',
    amount: 4200,
    currency: 'EUR',
    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,
        "client" : "<Object>",
        "currency" : "EUR",
            "created_at" : 1349946151,
            "updated_at" : 1349946151,
            "response_code" : 20000,
            "short_id" : "0000.1212.3434",
            "is_fraud" : false,
            "invoices" : [],
            "app_id" : null,
            "preauthorization" : null,
            "fees" : [],
            "payment" : "<Object>",
            "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/v2.1/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

```
transaction_service = paymill_context.get_transaction_service();
transaction_with_client_and_payment = transaction_service.create_with
_payment_id(
    payment_id='pay_3af44644dd6d25c820a9',
    amount=4200, currency='EUR',
    description='Test Transaction',
    client_id='client_33baaf3ee3251b083420'
);
```

RUBY

```
client = Paymill::Client.create( email: 'mail@example.com' )
payment = Paymill::Payment.create(
    token: '098f6bcd4621d373cade4e832627b4f6',
    client: client
)

Paymill::Transaction.create(
    payment: payment,
    client: client,
    amount: 4200,
    currency: 'EUR'
)
```

.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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "app_id" : null,
        "preauthorization" : null,
        "fees" : [],
        "payment" : "<Object>",
        "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/v2.1/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

```
transaction_service = paymill_context.get_transaction_service();
transaction_with_preauthorization = transaction_service.create_with_p
reauthorization_id(
    preauthorization_id='preauth_ec54f67e52e92051bd65',
    amount=4200, currency='EUR',
    description='Test Transaction'
);
```

RUBY

```
preauthorization = Paymill::Preauthorization.create(
    token: '098f6bcd4621d373cade4e832627b4f6',
    amount: 4200,
    currency: 'EUR'
)
transaction = Transaction.create(
    preauthorization: preauthorization,
    amount: 4200,
    currency: 'EUR'
)
```

.NET

```
PreauthorizationService preauthorizationService = paymillContext.Prea
uthorizationService;
Preauthorization preauthorization = preauthorizationService.CreateWit
hTokenAsync(
    "098f6bcd4621d373cade4e832627b4f6",
    4200,
    "EUR"
).Result;

TransactionService transactionService = paymillContext.TransactionSer
vice;
Transaction transaction = transactionService.CreateWithPreauthorizati
onAsync(
    preauthorization,
    4200,
    "EUR",
    "Test Transaction"
).Result;
```

JS

```
pm.transactions.createWithPreauthorization("preauth_ec54f67e52e92051b
d65", 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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "app_id" : null,
        "preauthorization" : null,
        "fees" : [],
        "payment" : "<Object>",
        "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/v2.1/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

```
transaction_service = paymill_context.get_transaction_service();
transaction_with_token = transaction_service.create_with_token(
    token='098f6bcd4621d373cade4e832627b4f6',
    amount=4200, currency='EUR',
    description='Test Transaction',
    fee_amount=4200,
    fee_payment_id='pay_3af44644dd6d25c820a8',
    fee_currency='EUR'
);
```

RUBY

```
Paymill::Transaction.create(
    token: '098f6bcd4621d373cade4e832627b4f6',
    amount: 4200, currency: 'EUR',
    description: 'Test Transaction',
    fee_amount: 4200,
    fee_payment: 'pay_a818b847db6ce5ff636f',
    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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "app_id" : null,
        "preauthorization" : null,
        "fees" : [
          {
            "type" : "application",
            "application" : "app_1d70acbf80c8c35ce83680715c06be0",
            "payment" : "pay_3af44644dd6d25c820a8",
            "amount" : 420,
            "currency" :  "EUR",
            "billed_at" :  null
          }
        ],
        "payment" : "<Object>",
        "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.

Transaction Id

Request

CURL

```
curl https://api.paymill.com/v2.1/transactions/tran_023d3b5769321c649
435 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

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

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

Attributes ¶

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

```
transaction_service = paymill_context.get_transaction_service();
transaction_details = transaction_service.detail(transaction_with_tok
en);
```

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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
            "short_id" : "0000.1212.3434",
            "is_fraud" : false,
            "invoices" : [],
            "app_id" : null,
            "preauthorization" : null,
            "fees" : [],
            "payment" : "<Object>",
            "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/v2.1/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

```
transaction_service = paymill_context.get_transaction_service();
transaction_details = transaction_service.detail(transaction_with_tok
en);
```

RUBY

```
Paymill::Transaction.find( 'tran_023d3b5769321c649435' )
```

.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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "app_id" : null,
        "preauthorization" : null,
        "fees" : [],
        "payment" : "<Object>",
        "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.

Request

CURL

```
curl https://api.paymill.com/v2.1/transactions/tran_023d3b5769321c649
435 \
  -u <YOUR_PRIVATE_KEY>:  \
  -d "description=My updated transaction description" \
  -X PUT
```

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.

PHP

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

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

JAVA

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

```
transaction_service = paymill_context.get_transaction_service();
transaction_with_token.amount = 3200;
transaction_with_token.currency = 'USD';
transaction_with_token.description = 'My updated transaction descript
ion'
transaction_service.update(transaction_with_token);
```

RUBY

```
transaction =  Paymill::Transaction.find( 'tran_023d3b5769321c649435'
)
transaction.description = 'My updated transaction description'
transaction.update
```

.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,
        "client" : "<Object>",
        "currency" : "EUR",
        "created_at" : 1349946151,
        "updated_at" : 1349946151,
        "response_code" : 20000,
        "short_id" : "0000.1212.3434",
        "is_fraud" : false,
        "invoices" : [],
        "app_id" : null,
        "preauthorization" : null,
        "fees" : [],
        "payment" : "<Object>",
        "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`**

Request

CURL

```
curl https://api.paymill.com/v2.1/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;
```

- closed
- failed
- preauth
- pending
- refunded
- partially_refunded
- chargeback

```
        }
        console.log("transaction data " + transaction.data);
    }
);
```

PYTHON

```
transaction_service = paymill_context.get_transaction_service();
transactions_list = transaction_service.list();
```

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,
            "client" : "<Object>",
            "currency" : "EUR",
            "created_at" : 1349946151,
            "updated_at" : 1349946151,
            "response_code" : 20000,
            "short_id" : "0000.1212.3434",
            "is_fraud" : false,
            "invoices" : [],
            "app_id" : null,
            "preauthorization" : null,
            "fees" : [],
            "payment" : "<Object>",
            "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,
            "client" : "<Object>",
            "currency" : "EUR",
            "created_at" : 1349953847,
            "updated_at" : 1349953847,
            "response_code" : 20000,
            "short_id" : "0000.1212.3434",
            "is_fraud" : false,
            "invoices" : [],
            "app_id" : null,
            "preauthorization" : null,
            "fees" : [],
            "payment" : "<Object>",
            "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/v2.1/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";"curr
ency";"created_at";"updated_at";"response_code";"short_id";"is_fraud"
;"app_id";"client_id";"payment_id";"preauthorization_id";"invoices";"
fees"
"tran_494d384289fbaa1aa342a35723f7";"599";"599";"closed";"Test Transa
ction";"";"EUR";"1342427064";"1342427064";"20000";"7357.7357.7357";""
;"";"client_53396385b7438a6a5cc2";"pay_2bbe85119a00f22d061eb752";;"";
""
```

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 ¶

Example

```
{
    "id" : "refund_87bc404a95d5ce616049",
    "amount" : "042",
    "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

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.

`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 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

Request

CURL

```
curl https://api.paymill.com/v2.1/refunds/tran_023d3b5769321c649435 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=4200"
```

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

```
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

```
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

```
refund_service = paymill_context.get_refund_service();
refund_transaction = refund_service.refund_transaction(
    transaction_id='tran_ca3e7d41fb16d0157a99',
    amount=4200
);
```

RUBY

```
transaction = Paymill::Transaction.create(
    token: '098f6bcd4621d373cade4e832627b4f6',
    amount: 4200,
    currency: 'USD'
)

Paymill::Refund.create(
    transaction,
    amount: 4200,
```

Attributes ¶

amount :	integer (>0) Amount (in cents) which will be charged
description :	string or null additional description for this refund

```
    description: 'Sample Description'
)
```

.NET

```
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" : "010",
        "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/v2.1/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

```
refund_service = paymill_context.get_refund_service();
refund_details = refund_service.detail(refund_transaction)
```

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" : "042",
        "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/v2.1/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

```
refund_service = paymill_context.get_refund_service();
refunds_list = refund_service.list();
```

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" : "042",
            "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/v2.1/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"
"refund_a7c4a0b9d09d9833a5d5";"2222";"refunded";"";"";"1342427064";"1
342427064";"20000";"";"tran_27a814bfbc7f3af580143713f80e"
```

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 ¶

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

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.
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/v2.1/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

```
client_service = paymill_context.get_client_service();
client = client_service.create(email='lovely-client@example.com');
```

RUBY

```
Paymill::Client.create(
    email: 'lovely-client@example.com',
    description: 'Lovely Client'
)
```

.NET

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

JS

```
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/v2.1/clients/client_88a388d9dd48f86c3136
\
  -u <YOUR_PRIVATE_KEY>:
```

PHP

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

$response = $request->getOne($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) {
```

Attributes ¶

id : string
Unique identifier for the client

```
            console.log("Error :(");
            return;
        }
        console.log("client id " + client.data.id);
    }
);
```

PYTHON

```
client_service = paymill_context.get_client_service();
client_details = client_service.detail(client)
```

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.

Request

CURL

```
curl https://api.paymill.com/v2.1/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

```
client_service = paymill_context.get_client_service();
client.email = 'lovely-client-updated-email@example.com';
client_service.update(client);
```

RUBY

```
client = Paymill::Client.find( 'client_88a388d9dd48f86c3136' )
client.email = 'lovely-client@example.com'
client.description = 'My Lovely Client'
client.update
```

.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);
```

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

```
});
```

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/v2.1/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");
```

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

Attributes ¶

id : string
Unique identifier for the client

```
client_service = paymill_context.get_client_service();
client_service.remove(client);
```

RUBY

```
client = Paymill::Client.find( 'client_88a388d9dd48f86c3136' )
client.delete
```

.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": 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`

Request

CURL

```
curl https://api.paymill.com/v2.1/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);
```

```
(from)>-<timestamp (to)>
```

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

PYTHON

```
client_service = paymill_context.get_client_service();
clients_list = client_service.list();
```

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

Request

CURL

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`

```
curl https://api.paymill.com/v2.1/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 ¶

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
}
```

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.

Create new **`Offer`** ¶

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

Request

CURL

```
curl https://api.paymill.com/v2.1/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();
```

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

```
Offer offer = offerService.create(
    4200,
    "EUR",
    "1 WEEK",
    "Nerd Special",
    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

```
offer_service = paymill_context.get_offer_service();
offer = offer_service.create(
    amount=4200,
    currency='EUR',
    interval='1 WEEK',
    'Nerd Special',
    0
);
```

RUBY

```
Paymill::Offer.create(
    amount: 4200,
    currency: 'EUR',
    interval: '1 MONTH',
    name: 'Nerd Special',
    trial_period_days: 0
)
```

.NET

```
OfferService offerService = paymillContext.OfferService;

Offer offer = offerService.CreateAsync(
    4200,
    "EUR",
    Interval.period(1, Interval.TypeUnit.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/v2.1/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 :(");
            return;
        }
        console.log("offer id " + offer.data.id);
    }
);
```

PYTHON

```
offer_service = paymill_context.get_offer_service();
offer_details = offer_service.detail(offer);
```

Attributes ¶

id : string
  Unique identifier for the offer

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. With the update_subscriptions attribute all related subscriptions could be updated too.

Request

CURL

```
curl https://api.paymill.com/v2.1/offers/offer_40237e20a7d5a231d99b \
  -u <YOUR_PRIVATE_KEY>: \
  -d "name=Extended Special" \
  -d "interval=1 MONTH" \
  -d "amount=3333" \
  -d "currency=USD" \
  -d "trial_period_days=33" \
  -d "update_subscriptions=true" \
  -X PUT
```

PHP

```
$offer = new Paymill\Models\Request\Offer();
$offer->setId('offer_40237e20a7d5a231d99b')
      ->setName('Extended Special')
      ->setInterval('1 MONTH')
      ->setAmount(3333)
```

Attributes ¶

id :	string Unique identifier for the offer
name :	string Your name for this offer (optional)
interval :	string Defining how often the client should be charged. Format: number

	DAY|WEEK|MONTH|YEAR (optional)
amount:	string
	Your amount of the offer in cents (optional)
currency:	string
	ISO 4217 formatted currency code (optional)
trial_period_day..	int
	Your trial period in number of days (optional)
update_subscri...	boolean
	Definition, if all related subscriptions also should be updated.

```
      ->setCurrency('USD')
      ->setTrialPeriodDays(33)
      ->updateSubscriptions(true);

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

JAVA

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

offer.setName("Extended Special");
offer.setInterval("1 MONTH")
offer.setAmount(3333);
offer.setCurrency("USD");
offer.setTrialPeriodDays(33);

boolean updateSubscriptions=true;
offerService.update( offer,updateSubscriptions );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
offer_service = paymill_context.get_offer_service();
offer.name = 'Extended Special';
offer.interval = '1 MONTH';
offer.amount = 3333;
offer.currency = 'USD';
offer.trial_period_days = '33';

offer_service.update(offer, true);
```

RUBY

```
offer = Paymill::Offer.find( 'offer_40237e20a7d5a231d99b' )

offer.name = 'Extended Special'
offer.interval = '1 MONTH'
offer.currency = 'USD'
offer.amount = '3333'

offer.update( update_subscriptions: true )
```

.NET

```
OfferService offerService = paymillContext.OfferService;

Offer offer = offerService.GetAsync( "offer_40237e20a7d5a231d99b" ).R
esult;

offer.Name = "Extended Special";
offer.Interval = Interval.period(1, Interval.TypeUnit.MONTH);
offer.Amount = 3333;
offer.Currency = "USD";
offer.TrialPeriodDays  = 33;

boolean updateSubscriptions = true;

Offer updatedOffer = offerService.UpdateAsync(offer, updateSubscripti
ons ).Result;
```

JS

```
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" : 3333,
        "currency": "USD",
        "interval" : "1 MONTH",
        "trial_period_days" : 33,
        "created_at" : 1341935129,
        "updated_at" : 1341938129,
        "subscription_count": {
            "active": "3",
            "inactive": 0
        },
        "app_id": null
    },
    "mode" : "test"
}
```

Remove `offer` ¶

You only can delete an offer and decide, if all related subscriptions also should be deleted or not.

Attributes ¶

id :	string Unique identifier for the offer
remove_with_su.	boolean Definition if all related subscriptions also should be deleted.

Request

CURL

```
curl https://api.paymill.com/v2.1/offers/offer_40237e20a7d5a231d99b \
  -u <YOUR_PRIVATE_KEY>: \
  -d "remove_with_subscriptions=false" \
  -X DELETE
```

PHP

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

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

JAVA

```
OfferService offerService = paymillContext.getOfferService();
boolean removeWithSubscriptions = true;
offerService.delete( "offer_40237e20a7d5a231d99b", removeWithSubscrip
tions );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
offer_service = paymill_context.get_offer_service();
```

```
offer_service.remove(offer);
```

RUBY

```
# with subscriptions:
offer = Paymill::Offer.find( 'offer_40237e20a7d5a231d99b' )
offer.delete_with_subscriptions

# without subscriptions:
offer = Paymill::Offer.find( 'offer_40237e20a7d5a231d99b' )
offer.delete_without_subscriptions()
```

.NET

```
OfferService offerService = paymillContext.OfferService;
boolean removeWithSubscriptions = true;
Boolean result = offerService.DeleteAsync(
    "offer_40237e20a7d5a231d99b",
    removeWithSubscriptions
).Result;
```

JS

```
pm.offers.remove("offer_40237e20a7d5a231d99b",false).then(function(of
fer) {
    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`

Request

CURL

```
curl https://api.paymill.com/v2.1/offers \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

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

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

JAVA

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

NODE.JS

(from)>-<timestamp (to)>

- updated_at=<timestamp> | <timestamp (from)>-<timestamp (to)>

```
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

```
offer_service = paymill_context.get_offer_service();
offers_list = offer_service.list();
```

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`
- `interval`
- `name`
- `trial_period_days`
- `updated_at`

Available filters:

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

Request

CURL

```
curl https://api.paymill.com/v2.1/offers \
  -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";"name";"amount";"currency";"interval";"trial_period_days";"creat
ed_at";"updated_at";"subscription_count_active";"subscription_count_i
nactive";"app_id"
"offer_1a5d80dc75db9b5c0c64";"Example Offer";"499";"EUR";"3 WEEK";"22
";"1342427064";"1342427064";"1";"8";""
```

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 ¶

Attributes ¶

id :	string Unique identifier of this subscription.
livemode :	boolean Whether this subscription was issued while being in live mode or not.
offer :	offer object
amount :	integer the amount of the subscription in cents
temp_amount :	integer or null a one-time amount in cents, will charge once only
curreny :	string ISO 4217 formatted currency code
interval :	string Defining how often the client should be charged. Format: number DAY|WEEK|MONTH|YEAR [, WEEKDAY] Example: 2 DAYS, MONDAY
name :	string or null name of the subscription
trial_start :	integer or null Unix-Timestamp for the trial period start
trial_end :	integer or null Unix-Timestamp for the trial period end.
period_of_valid…	string or null limit the validity of the subscription, format: integer MONTH|YEAR|WEEK|DAY
end_of_period :	Unix-Timestamp or null expiring date of the subscription
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.
is_canceled :	boolean subscription is marked as canceled or not
is_deleted :	boolean

Example

```
{
    "id" : "sub_09a1944830b7e37e2005",
    "offer" : "<Object>",
    "livemode" : false,
    "amount" : 299,
    "temp_amount" : null,
    "currency" : "USD",
    "name" : "Testing",
    "interval" : "1 DAY",
    "trial_start" : 1400555454,
    "trial_end" : null,
    "period_of_validity" : null,
    "end_of_period" : null,
    "next_capture_at" : 1400642826,
    "created_at" : 1400555454,
    "updated_at" : 1400556426,
    "canceled_at" : null,
    "payment" : "<Object>",
    "app_id" : null,
    "is_canceled" : false,
    "is_deleted" : false,
    "status" : "failed",
    "mandate_reference" : null,
    "client" : "<Object>"
}
```

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 is marked as deleted or not
status :	string
	shows, if subscription is "active", "inactive", "expired" or "failed"

This function connects the offer with a client.

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.

Attributes ¶

offer :	string
	Unique offer identifier (if no offer is given, amount, currency and interval are required)
payment :	string
	Unique payment identifier
client :	string
	Unique client identifier. If not provided the client from the payment is being used.
amount :	integer (>0)
	the amount of the subscription in cents (is required if no offer id is given)
currency :	string
	ISO 4217 formatted currency code (is required if no offer id is given)
interval :	string
	Defining how often the client should be charged. Format: number DAY|WEEK|MONTH|YEAR [, WEEKDAY] Example: 2 DAYS, MONDAY (is required if no offer id is given)
name :	string or null
	name of the subscription (optional)
period_of_valid…	string or null
	limit the validity of the subscription, format: integer MONTH|YEAR|WEEK|DAY (optional)
start_at :	integer or null
	Unix-Timestamp for the subscription start date, if trial_end > start_at, the trial_end will be set to start_at (optional)
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

Without an offer

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "client=client_81c8ab98a8ac5d69f749" \
  -d "payment=pay_5e078197cde8a39e4908f8aa" \
  -d "amount=3000" \
  -d "currency=EUR" \
  -d "interval=1 week,monday" \
  -d "name=Example Subscription" \
  -d "period_of_validity=2 YEAR" \
  -d "start_at=1400575533"
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setClient('client_81c8ab98a8ac5d69f749')
             ->setAmount(3000);
             ->setPayment('pay_5e078197cde8a39e4908f8aa');
             ->setCurrency('EUR');
             ->setInterval('1 week,monday');
             ->setName('Example Subscription');
             ->setPeriodOfValidity('2 YEAR');
             ->setStartAt('1400575533');

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.create( Subscription.create(
    "pay_5e078197cde8a39e4908f8aa",
    3000,
    "EUR",
    Interval.periodWithChargeDay( 1, Unit.WEEK, Weekday.MONDAY )
    )
    .withName( "Example Subscription" )
    .withPeriodOfValidity( Interval.period( 2, Unit.YEAR ) )
    .withStartDate( new Date( 1400575533 ) )
);
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_without_offer = subscription_service.create_with_amount(
```

letters a-z A-Z
allowed special characters: ‘ , . : + - / ( ) ?

```
    payment_id='pay_3af44644dd6d25c820a9',
    amount=4200,
    currency='EUR',
    interval='2 DAYS,MONDAY'
);
```

RUBY

```
client = Paymill::Client.create( email: 'client@example.com' )
payment = Paymill::Payment.create( token: '098f6bcd4621d373cade4e8326
27b4f6', client: client )

Paymill::Subscription.create(
    payment: payment,
    client: client,
    name: 'Example Subscription',
    amount: 3000,
    currency: 'EUR',
    interval: '1 week,monday',
    period_of_validity: '2 YEAR',
    start_at: 2.days.from_now
)
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

subscriptionService.CreateAsync( Subscription.Create(
    "pay_5e078197cde8a39e4908f8aa",
    3000,
    "EUR",
    Interval.periodWithChargeDay( 1, Interval.TypeUnit.WEEK, Interval
.Weekday.MONDAY ) )
.WithName( "Example Subscription" )
.WithPeriodOfValidity( Interval.period( 2, Interval.TypeUnit.YEAR ) )
.WithStartDate(DateTime.Now.AddDays(5)
);
```

JS

```
pm.subscriptions.fromParams("pay_5e078197cde8a39e4908f8aa",3000,"EUR"
,"1 week,monday")
.withClient( "client_81c8ab98a8ac5d69f749" )
.withName( "Example Subscription" )
.withPeriodOfValidity( "2 YEAR" )
.withStartDate( new Date(1400575533))
.create().then( function(subscription) {
    console.log("created subscription:" + subscription.id);
}, function(error) {
    console.log("couldnt create subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : {<Object>},
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1399908040,
        "trial_end" : 1400575532,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1400575532,
        "created_at" : 1398271207,
        "updated_at" : 1398271207,
            "canceled_at" : null,
            "payment" : {<Object>},
            "app_id" : null,
            "is_canceled" : false,
            "is_deleted" : false,
            "status" : "active",
            "mandate_reference" : null,
            "client" : {<Object>}
        },
        "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

With an offer

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "client=client_64b025ee5955abd5af66" \
  -d "offer=offer_40237e20a7d5a231d99b" \
  -d "payment=pay_95ba26ba2c613ebb0ca8" \
  -d "period_of_validity=2 YEAR" \
  -d "start_at=1400575533"
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setClient('client_81c8ab98a8ac5d69f749')
             ->setOffer('offer_40237e20a7d5a231d99b');
             ->setPayment('pay_5e078197cde8a39e4908f8aa');
             ->setPeriodOfValidity('2 YEAR');
             ->setStartAt(1400575533);

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.create( Subscription.create(
    "pay_95ba26ba2c613ebb0ca8",
    "offer_40237e20a7d5a231d99b"
).withPeriodOfValidity( Interval.period( 2, Unit.YEAR ) )
.withStartDate( new Date(1400575533) )
);
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_with_an_offer = subscription_service.create_with_offer_i
d(
    payment_id='pay_3af44644dd6d25c820a9',
    offer_id='offer_bb33ea77b942f570997b'
);
```

RUBY

```
payment = Paymill::Payment.create( token: '098f6bcd4621d373cade4e8326
27b4f6' )
offer = Paymill::Offer.create(
    amount: 3333,
    currency: 'EUR',
    interval: '1 WEEK',
    name: 'Nerd Special',
    trial_period_days: 30
    )

Paymill::Subscription.create(
    payment: payment,
    offer: offer,
    period_of_validity: '2 YEAR',
    start_at: 2.days.from_now
)
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

subscriptionService.CreateAsync( Subscription.Create(
    "pay_95ba26ba2c613ebb0ca8",
    "offer_40237e20a7d5a231d99b" )
.WithPeriodOfValidity( Interval.period( 2, Interval.TypeUnit.YEAR ) )
.WithStartDate( DateTime.Now.AddDays(5) )
);
```

JS

```
pm.subscriptions.fromOffer("pay_95ba26ba2c613ebb0ca8","offer_40237e20
a7d5a231d99b")
.withClient( "client_64b025ee5955abd5af66" )
.withName( "Example Subscription" )
.withPeriodOfValidity( "2 YEAR" )
.withStartDate( new Date(1400575533)).create().then( function(subscri
ption) {
    console.log("created subscription:" + subscription.id);
}, function(error) {
    console.log("couldnt create subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : {<Object>},
        "livemode" : false,
        "amount" : 3333,
        "temp_amount" : null,
        "currency" : "USD",
        "name" : "Offer Name",
        "interval" : "2 WEEK",
        "trial_start" : 1399908040,
        "trial_end" : 1400575532,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1400575532,
        "created_at" : 1398271207,
        "updated_at" : 1398271207,
        "canceled_at" : null,
        "payment" : {<Object>},
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : {<Object>}
    },
    "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

With offer and different values

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions \
  -u <YOUR_PRIVATE_KEY>: \
  -d "client=client_81c8ab98a8ac5d69f749" \
  -d "payment=pay_5e078197cde8a39e4908f8aa" \
  -d "offer=offer_b33253c73ae0dae84ff4" \
  -d "amount=3000" \
  -d "currency=EUR" \
  -d "interval=1 week,monday" \
  -d "name=Example Subscription" \
  -d "period_of_validity=2 YEAR" \
  -d "start_at=1400575533"
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setClient('client_81c8ab98a8ac5d69f749')
             ->setOffer('offer_40237e20a7d5a231d99b');
             ->setAmount(3000);
             ->setPayment('pay_5e078197cde8a39e4908f8aa');
             ->setCurrency('EUR');
             ->setInterval('1 week,monday');
             ->setName('Example Subscription');
             ->setPeriodOfValidity('2 YEAR');
             ->setStartAt('1400575533');

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.create( Subscription.create(
    "pay_95ba26ba2c613ebb0ca8",
    "offer_40237e20a7d5a231d99b"
).withAmount( 3000 )
.withCurrency( "EUR" )
.withInterval( Interval.periodWithChargeDay( 1, Unit.WEEK, Weekday.MO
NDAY ) )
.withPeriodOfValidity( Interval.period( 2, Unit.YEAR ) )
.withStartDate( new Date( 1400575533 ) )
);
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_with_offer_and_different_values = subscription_service.c
reate_with_offer_id(
    payment_id='pay_3af44644dd6d25c820a9',
    offer_id='offer_bb33ea77b942f570997b',
    name='Subscription with values',
    period_of_validity='4 WEEKS',
    start_at=1409647372
);
```

RUBY

```
payment = Paymill::Payment.create( token: '098f6bcd4621d373cade4e8326
27b4f6' )
offer = Paymill::Offer.create(
    amount: 4200,
    currency: 'EUR',
    interval: '1 MONTH',
    name: 'Nerd Special',
    trial_period_days: 30
)

Paymill::Subscription.create(
    payment: payment,
    offer: offer,
    amount: 3000,
    currency: 'EUR',
    interval: '1 week, monday',
    name: 'Example Subscription',
    period_of_validity: '2 YEAR',
    start_at: 2.days.from_now
)
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

subscriptionService.CreateAsync( Subscription.Create(
    "pay_95ba26ba2c613ebb0ca8",
    "offer_40237e20a7d5a231d99b" )
.WithAmount( 3000 )
.WithCurrency( "EUR" )
.WithInterval( Interval.periodWithChargeDay( 1, Interval.TypeUnit.WEE
K, Interval.Weekday.MONDAY ) )
.WithPeriodOfValidity( Interval.period( 2, Interval.TypeUnit.YEAR ) )
.withStartDate( DateTime.Now.AddDays(5) )
);
```

JS

```
pm.subscriptions.fromOffer("pay_5e078197cde8a39e4908f8aa","offer_b332
53c73ae0dae84ff4")
.withClient( "client_81c8ab98a8ac5d69f749" )
.withAmount( 3000 )
.withCurrency( "EUR")
.withInterval("1 week,monday")
.withName( "Example Subscription" )
.withPeriodOfValidity( "2 YEAR" )
.withStartDate( new Date(1400575533))
.create().then( function(subscription) {
    console.log("created subscription:" + subscription.id);
}, function(error) {
    console.log("couldnt create subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : {<Object>},
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1399908040,
        "trial_end" : 1400575532,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1400575532,
        "created_at" : 1398271207,
        "updated_at" : 1398271207,
        "canceled_at" : null,
        "payment" : {<Object>},
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : {<Object>}
    },
    "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/v2.1/subscriptions/sub_dc180b755d10da324
864 \
  -u <YOUR_PRIVATE_KEY>:
```

PHP

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

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
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);
```

Attributes ¶

id : string
Unique identifier for the subscription

```
    }
);
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_details = subscription_service.detail(subscription_witho
ut_offer)
```

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_dea86e5c65b2087202e3",
        "offer" : {<Object>},
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1399908040,
        "trial_end" : 1400575532,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1400575532,
        "created_at" : 1398271207,
        "updated_at" : 1398271207,
        "canceled_at" : null,
        "payment" : {<Object>},
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : {<Object>}
    },
    "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 trial_end attribute to stop the trial period. Or you can assign the subscription to another offer (offer=<new_offer_id>), change the amount or pause it. NOTE: changing the amount and offer within one request is not possible (throw an exception).

Attributes ¶

id :	string	Unique identifier for the subscription
payment :	string	Unique identifier describing a payment of the client
offer :	string	Unique identifier describing the offer which is subscribed to the client (optional)
offer_change_ty..	int or null	permitted values: 0,1,2; linked and required with 'offer', default: 0 (optional)
amount :	integer (>0)	the amount of the subscription in cents (optional)
amount_change..	int	permitted values: 0,1; linked and required with 'amount' (optional)
pause :	boolean	deactivate a subscription or reactivate it, false: reactivate, true: deactivate (optional)
currency :	string	ISO 4217 formatted currency code (optional)
interval :	string	Defining how often the client should be charged. Format: number DAY|WEEK|MONTH|YEAR [, WEEKDAY] (optional)
name :	string	name of the subscription (optional)
period_of_valid...	string	limit the validity of the subscription, format: integer MONTH|YEAR|WEEK|DAY, set to "remove" to unlimit the validity period (optional)
trial_end :	boolean	set to false to stop the trial period immediatly (optional)
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

General

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions/sub_dea86e5c65b208720
2e3 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "payment=pay_95ba26ba2c613ebb0ca8" \
  -d "currency=USD" \
  -d "interval=1 month,friday" \
  -d "name=Changed Subscription" \
  -d "period_of_validity=14 MONTH" \
  -d "trial_end=false" \
  -X PUT
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dea86e5c65b2087202e3');
             ->setClient('client_81c8ab98a8ac5d69f749')
             ->setOffer('offer_40237e20a7d5a231d99b');
             ->setAmount(3000);
             ->setPayment('pay_95ba26ba2c613ebb0ca8');
             ->setCurrency('USD');
             ->setInterval('1 month,friday');
             ->setName('Changed Subscription');
             ->setPeriodOfValidity('14 MONTH');
             ->setTrialEnd(false);

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
Subscription subscription = subscriptionService.get( "sub_dea86e5c65b
2087202e3" );
subscription.setName( "Changed Subscription" );
subscriptionService.update( subscription );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_without_offer.name = 'Updated Subscription';
subscription_service.update(subscription_without_offer);
```

RUBY

```
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
subscription.payment = 'pay_95ba26ba2c613ebb0ca8'
subscription.currency = 'USD'
subscription.interval = '1 month,friday'
subscription.name = 'Changed Subscription'
subscription.trial_end = false
subscription.update

# for limit / unlimit use
```

digits 0-9

letters a-z A-Z

allowed special characters: ‘ , . : + - / ( ) ?

```
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
subscription.unlimit
subscription.limit( '12 MONTH' )
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

Subscription subscription = subscriptionService.GetAsync( "sub_dea86e
5c65b2087202e3" ).Result;

subscription.Name = "Changed Subscription" ;
subscription.Offer = null; // Do not update Offer
subscription.Currency = null;// Do not update Currency
subscription.Interval = null;// Do not update Interval

Subscription updatedSubscription = subscriptionService.UpdateAsync(su
bscription).Result;
```

JS

```
pm.subscriptions.detail("sub_dea86e5c65b2087202e3").then(function(sub
scription) {
    subscription.name = "Changed Subscription";
    return pm.transactions.update(subscription);
}).then(function(updatedSubscription) {
    console.log("updated subscription name:" + updatedSubscription.na
me);
}, function(error) {
    console.log("couldnt update the subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : {<Object>},
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "USD",
        "name" : "Changed Subscription",
        "interval" : "1 MONTH,FRIDAY",
        "trial_start" : 1399908040,
        "trial_end" : null,
        "period_of_validity" : "12 MONTH",
        "end_of_period" : 1435063506,
        "next_capture_at" : 1400575532,
        "created_at" : 1398271207,
        "updated_at" : 1398343548,
        "canceled_at" : null,
        "payment" : {<Object>},
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : {<Object>}
    },
    "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

Amount

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions/sub_dea86e5c65b208720
2e3 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "amount=1234" \
  -d "amount_change_type=0" \
  -X PUT
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dea86e5c65b2087202e3');
             ->setAmount(1234);
             ->setAmountChangeType(0);

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.changeAmountTemporary( "sub_dea86e5c65b2087202e3"
, 1234 );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_without_offer.amount = 5200;
subscription_service.update_with_amount(subscription_without_offer, a
mount_change_type=1);
```

RUBY

```
#amount temporary:
    subscription = Paymill::Subscription.find( 'sub_dea86e5c65b208720
2e3' )
    subscription.update_amount_once( 1234 )

#amount permanently:
    subscription = Paymill::Subscription.find( 'sub_dea86e5c65b208720
2e3' )
    subscription.update_amount_permanently( 1234 )
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

Subscription updatedSubscription = subscriptionService.ChangeAmountTe
mporaryAsync( "sub_dea86e5c65b2087202e3" , 1234 ).Result;
```

JS

```
pm.subscriptions.detail("sub_dea86e5c65b2087202e3").then(function(sub
scription) {
    return pm.transactions.changeAmountTemporary(subscription, 1234);
}).then(function(updatedSubscription) {
    console.log("updated subscription amount:" + updatedSubscription.
amount);
}, function(error) {
    console.log("couldnt update the subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : "<Object>",
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : "1234",
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1398271207,
        "trial_end" : 1399196896,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1399308007,
        "created_at" : 1398271207,
        "updated_at" : 1398271302,
        "canceled_at" : null,
        "payment" :  "<Object>",
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : "<Object>"
    },
    "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

Offer

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions/sub_dea86e5c65b2087202e3 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "offer=offer_d7e9813a25e89c5b78bd" \
  -d "offer_change_type=2" \
  -X PUT
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dea86e5c65b2087202e3');
             ->setOffer('offer_d7e9813a25e89c5b78bd');
             ->setOfferChangeType(2);

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

JAVA

```
Offer offer = paymillContext.getOfferService().get( "offer_d7e9813a25e89c5b78bd" );
SubscriptionService subscriptionService = paymillContext.getSubscriptionService();
Subscription subscription = subscriptionService.get( "sub_dea86e5c65b
2087202e3" );
subscriptionService.changeOfferChangeCaptureDateAndRefund( subscripti
on, offer );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_with_an_offer.offer_id='offer_40237e20a7d5a231d99b';
subscription_service.update_with_offer_id(
    subscription_with_an_offer,
    offer_change_type=2
);
```

RUBY

```
#offer with no refund and unchanged capture date:
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
offer = Offer.create( name: 'Foo', amount: 4990, currency: 'EUR', int
erval: '2 WEEK')
subscription.update_offer_without_changes( offer )

#offer with refund and unchanged capture date:
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
offer = Offer.create( name: 'Foo', amount: 4990, currency: 'EUR', int
erval: '2 WEEK')
subscription.update_offer_with_refund( offer )

#offer with refund and capture date:
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
offer = Offer.create( name: 'Foo', amount: 4990, currency: 'EUR', int
erval: '2 WEEK')
subscription.update_offer_with_refund_and_capture_date( offer )
```

.NET

```
Offer offer = paymillContext.OfferService.GetAsync( "offer_d7e9813a25
e89c5b78bd" ).Result;

SubscriptionService subscriptionService = paymillContext.Subscription
Service;
Subscription subscription = subscriptionService.GetAsync( "sub_dea86e
5c65b2087202e3" ).Result;

Subscription updatedSubscription = subscriptionService.ChangeOfferCha
ngeCaptureDateAndRefundAsync( subscription, offer ).Result;
```

JS

```
pm.subscriptions.detail("sub_dea86e5c65b2087202e3").then(function(sub
scription) {
    return pm.transactions.changeOfferChangeCaptureDateAndRefund(subs
cription, "offer_d7e9813a25e89c5b78bd");
}).then(function(updatedSubscription) {
    console.log("updated subscription offer:" + updatedSubscription.o
ffer.id);
}, function(error) {
    console.log("couldnt update the subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : "<Object>",
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1398271207,
        "trial_end" : 1399196896,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1399308007,
        "created_at" : 1398271207,
        "updated_at" : 1398271302,
        "canceled_at" : null,
        "payment" : "<Object>",
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : "<Object>"
    },
    "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

Pause

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions/sub_dea86e5c65b208720
2e3 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "pause=true" \
  -X PUT
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dea86e5c65b2087202e3');
             ->setPause(true);

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.pause( "sub_dea86e5c65b2087202e3" );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
# pause
subscription_service = paymill_context.get_subscription_service();
```

```
subscription_service.pause(subscription_without_offer);
# unpause
subscription_service = paymill_context.get_subscription_service();
subscription_service.unpause(subscription_without_offer);
```

RUBY

```
#pause:
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
subscription.pause

#play:
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
subscription.play
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

Subscription updatedSubscription = subscriptionService.PauseAsync( "s
ub_dea86e5c65b2087202e3" ).Result;
```

JS

```
pm.subscriptions.detail("sub_dea86e5c65b2087202e3").then(function(sub
scription) {
    return pm.transactions.pause(subscription);
}).then(function(updatedSubscription) {
    console.log("updated subscription status:" + updatedSubscription.
status);
}, function(error) {
    console.log("couldnt update the subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : "<Object>",
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1398271207,
        "trial_end" : 1399196896,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1399308007,
        "created_at" : 1398271207,
        "updated_at" : 1398271302,
        "canceled_at" : null,
        "payment" : "<Object>",
        "app_id" : null,
        "is_canceled" : false,
        "is_deleted" : false,
        "status" : "inactive",
        "mandate_reference" : null,
        "client" : "<Object>"
    },
    "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

Cancel or Delete `Subscription` ¶

This function cancels or remove an existing subscription. The subscription will be directly terminated or deleted and no pending transactions will be charged. Deleted subscriptions will not be displayed.

Attributes ¶

id :	string Unique identifier for the subscription
remove :	boolean cancel (false) or delete (true) a subscription

Cancel

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions/sub_d68bcdc8656a7932e
b44 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "remove=false" \
  -X DELETE
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dea86e5c65b2087202e3');
             ->setRemove(false);

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

JAVA

```
SubscriptionService subscriptionService = paymillContext.getSubscript
ionService();
subscriptionService.cancel( "sub_dea86e5c65b2087202e3" );
```

NODE.JS

```
/* … not yet implemented for subscription v2.1 for this wrapper.
Please use the old version of subscription v2.0 and have a look in th
e V2.0 PDF … */
```

PYTHON

```
subscription_service = paymill_context.get_subscription_service();
subscription_service.cancel(subscription_without_offer);
```

RUBY

```
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
subscription.cancel
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

Subscription updatedSubscription = subscriptionService.CancelAsync( "
sub_dea86e5c65b2087202e3" ).Result;
```

JS

```
pm.subscriptions.cancel("sub_d68bcdc8656a7932eb44").then(function(sub
scription) {
    console.log("canceled subscription :" + subscription.status);
}, function(error) {
    console.log("couldnt cancel subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : "<Object>",
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1398271207,
        "trial_end" : 1399196896,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1399308007,
        "created_at" : 1398271207,
        "updated_at" : 1398271302,
        "canceled_at" : 1401194748,
        "payment" : "<Object>",
        "app_id" : null,
        "is_canceled" : true,
        "is_deleted" : false,
        "status" : "active",
        "mandate_reference" : null,
        "client" : "<Object>"
    },
    "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

Delete

Request

CURL

```
curl https://api.paymill.com/v2.1/subscriptions/sub_d68bcdc8656a7932e
b44 \
  -u <YOUR_PRIVATE_KEY>: \
  -d "remove=true" \
  -X DELETE
```

PHP

```
$subscription = new Paymill\Models\Request\Subscription();
$subscription->setId('sub_dea86e5c65b2087202e3');
             ->setRemove(true);

$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

```
subscription_service = paymill_context.get_subscription_service();
subscription_service.remove(subscription_without_offer);
```

RUBY

```
subscription = Paymill::Subscription.find( 'sub_dea86e5c65b2087202e3'
)
subscription.remove
```

.NET

```
SubscriptionService subscriptionService = paymillContext.Subscription
Service;

subscriptionService.DeleteAsync( "sub_dea86e5c65b2087202e3" ).Wait();
```

JS

```
pm.subscriptions.delete("sub_d68bcdc8656a7932eb44").then(function(sub
scription) {
    console.log("deleted subscription :" + subscription.status);
}, function(error) {
    console.log("couldnt delete subscription:" + error);
});
```

Response

```
{
    "data" : {
        "id" : "sub_dea86e5c65b2087202e3",
        "offer" : "<Object>",
        "livemode" : false,
        "amount" : 3000,
        "temp_amount" : null,
        "currency" : "EUR",
        "name" : "Example Subscription",
        "interval" : "1 WEEK,MONDAY",
        "trial_start" : 1398271207,
        "trial_end" : 1399196896,
        "period_of_validity" : "2 YEAR",
        "end_of_period" : 1461429607,
        "next_capture_at" : 1399308007,
        "created_at" : 1398271207,
        "updated_at" : 1398271302,
        "canceled_at" : 1401194748,
        "payment" : "<Object>",
        "app_id" : null,
        "is_canceled" : true,
        "is_deleted" : true,
        "status" : "active",
        "mandate_reference" : null,
        "client" : "<Object>"
    },
    "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/v2.1/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

```
subscription_service = paymill_context.get_subscription_service();
subscription_list = subscription_service.list();
```

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/v2.1/subscriptions \
  -u <YOUR_PRIVATE_KEY>: \
  -H "Accept: text/csv"
```

PHP

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

JAVA

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

PYTHON

```
# Not implemented yet
```

RUBY

```ruby
# Not implemented yet
```

.NET

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

JS

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

Response

```
"id";"livemode";"amount";"temp_amount";"currency";"name";"interval";"
trial_start";"trial_end";"period_of_validity";"end_of_period";"next_c
apture_at";"created_at";"updated_at";"canceled_at";"app_id";"is_cance
led";"is_deleted";"status";"offer_id";"client_id";"payment_id"
"sub_c84aadd0c1c7529158ee";"";"499";"";"EUR";"Example Subscription";"
3 WEEK";"1401983620";"1404575620";"";"";"";"1342427064";"1342427064";
"1402640050";"";"";"";"active";"offer_1a5d80dc75db9b5c0c64";"client_3
3c8f8c13d759d00b144";"pay_2311e5a076ab0b9c2cdb0399"
```

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

Available Events

- chargeback.executed: returns a transaction-object with state set to chargeback
- 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

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.

- 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
- subscription.expiring: returns a subscription-object
- subscription.deactivated: returns a subscription-object
- subscription.activated: returns a subscription-object
- subscription.canceled: returns 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.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,
```

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.

```
    "updated_at":1358982000,
    "active":true,
    "app_id":null
}
```

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.

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

Request

CURL

```
curl https://api.paymill.com/v2.1/webhooks \
-u <YOUR_PRIVATE_KEY>: \
-d "url=<your-webhook-url>" \
-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
);
```

PYTHON

```
webhook_service = paymill_context.get_webhook_service();
url_webhook = webhook_service.create_url(
    url='<your-webhook-url>',
    event_types=['transaction.succeeded', 'transaction.failed'],
    active=True
)
```

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);
    }
);
```

RUBY

```
Paymill::Webhook.create(
    url: 'http://example.com',
    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/v2.1/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
);
```

PYTHON

```
webhook_service = paymill_context.get_webhook_service();
email_webhook = webhook_service.create_email(
    email='lovely-webhook@example.com',
    event_types=['transaction.succeeded', 'transaction.failed'],
    active=True
)
```

NODE.JS

```
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);
    }
);
```

RUBY

```
Paymill::Webhook.create(
    email: 'webhook@example.com',
    event_types: ['transaction.succeeded', 'transaction.failed'],
    active: false
)
```

.NET

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

JS

```
pm.webhooks.createEmail("<your-webhook-email>",[
    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",
            "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/v2.1/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");
```

PYTHON

```
webhook_service = paymill_context.get_webhook_service();
webhook_details = webhook_service.detail(email_webhook)
```

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 ¶

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/v2.1/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 );
```

PYTHON

```
webhook_service = paymill_context.get_webhook_service();
email_webhook.email = 'updated-lovely-webhook@example.com'
webhook_service.update(email_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

```
webhook = Paymill::Webhook.find( active_webhook_id )
webhook.email = 'mail@example.com'
webhook.update
```

.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/v2.1/webhooks/hook_40237e20a7d5a231d99b
\
-u <YOUR_PRIVATE_KEY>: \
-X DELETE
```

PHP

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

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

JAVA

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

PYTHON

```python
webhook_service = paymill_context.get_webhook_service();
webhook_service.remove(email_webhook);
```

NODE.JS

```javascript
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

```ruby
webhook = Paymill::Webhook.find( active_webhook_id )
webhook.delete
```

.NET

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

JS

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

Response

```json
{
    "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/v2.1/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();
```

PYTHON

```
webhook_service = paymill_context.get_webhook_service();
webhooks_list = webhook_service.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

```
# Not implemented yet
```

.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/v2.1/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";"created_at";"updated_at";"active";"app_id";"version"
;"url";"event_types"
"hook_f0f84bc71b86f16fa1f5";"";"1342427064";"1342427064";"1";"";"2.1"
;;"refund.succeeded"
```

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

```
{
    "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
}
```

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

`Merchant` Object ¶

Example

```
{
    "identifier_key": "mer_123456789",
    "email": "mail@example.com",
    "locale": "de_DE",
    "country": "DEU",
    "currencies": ["EUR", "GPB"],
    "methods": ["visa", "mastercard"]
}
```

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

`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"
}
```