Payment Gateway API Documentation

API version 1.9
Documentation version 2.25

June 16, 2016

Contents

Version History

Version	Date	Name	Description
1.0	*2007/12/10*	sb	initial documentation
1.1	*2008/01/06*	sb	Added 3-D secure payment transactions
1.2	*2008/01/09*	sb	Added reconcile method
1.3	*2008/01/17*	sb	Added additional security for asynchronous payment notifications with new signature parameter
1.4	*2008/02/06*	sb	Changed test and live URLs
1.5	*2008/09/01*	la	refunded and chargebacked states added
1.6	*2009/02/19*	la	Processing API changed, cvv can now be optional if configured on channel
1.7	*2009/04/07*	la	Added Chargeback notfications
1.8	*2009/04/20*	la	Added notification signature examples and notification timeout
1.9	*2009/04/20*	la	Added Recurring Transactions
2.0	*2009/09/14*	la	Added Bank-Transfer Transactions
2.1	*2009/09/22*	la	Added MPI passthrough for 3d secure transactions
2.2	*2009/10/13*	la	Added new Error Classes UserLimitExceeded-Error and BlacklistError
2.3	*2009/10/13*	la	Added RiskParams
2.4	*2009/11/05*	la	Added Reconcile by date range
2.5	*2009/11/06*	la	Added Notifications
2.6	*2009/11/16*	la	Added ReferencedFundTransfer and Asynchronous transactions and Notifications
2.7	*2010/04/20*	la	Added DebitSale
2.8	*2010/06/14*	mk	Added 3D Secure testing card numbers
2.9	*2011/04/19*	sv	Added Web Payment Form
2.10	*2012/02/16*	mdk	Added InitRecurringAuthorize and PurchaseOnAccount
2.11	*2012/03/07*	mdk	Added PayInAdvance
2.12	*2012/03/13*	mdk	Added PaymentOnDelivery
2.13	*2012/06/06*	jm	Added PayPal
2.14	*2012/06/14*	mdk	Updated PayPal documentation
2.15	*2012/06/26*	mdk	Added wire_reference_id to DebitSale documentation
2.16	*2012/07/20*	mdk	Added scheduled recurring API extension documentation and advanced testing interface

2.17	*2012/10/24*	mdk	Added scheduled recurring API for WPF sessions and InitRecurringDebitAuthorize, InitRecurringDebitSale and RecurringDebitSale
2.18	*2013/01/17*	mdk	Added Mobile Payment API for payment sessions and recurring notifications
2.19	*2013/03/21*	mdk	Added Barzahlen.de payment method and updated Recurring event details section
2.20	*2014/01/30*	mdk	Added conditional bank account data and risk params to PurchaseOnAccount
2.21	*2014/06/24*	mdk	Added DirectPay24 (Sofortberweisung.de) and GiroPay
2.22	*2014/07/11*	mdk	Added SepaDebitSale, InitRecurringSepaDebitSale and InitRecurringSepaDebitAuthorize
2.23	*2014/07/11*	mdk	Added RecurringSepaDebitSale
2.24	*2014/09/12*	mdk	Added GtdSepaDebitSale, InitRecurringGtdSepaDebitSale, InitRecurringGtdSepaDebitAuthorize, RecurringGtdSepaDebitSale and GtdPurchaseOnAccount
2.25	*2014/11/28*	mdk	Added InitRecurringSale3d, InitRecurringAuthorize3d and RecurringAuthorize

0 Introduction

This documents describes the usage of the payment gateway XML API.

The API allows you to trigger all supported transactions of the gateway and to retrieve information about transactions existing in the gateway. You can also retrieve chargeback information.

The payment API is synchronous (except for 3-D secure payments), it accepts HTTP POST or XML data and returns XML data. Connections are always secured via TLS (v1.2) both in test and live mode. Be sure to set *Content-type: text/xml* in your headers.

Note:

The API cannot be accessed via HTTP GET.

Note:

Current Java SDKs do not have the necessary root certificate installed. You need to download the CA with your browser and import it into cacerts manually.

0.1 Audience

This document is intended for technical staff integrating the XML API in the merchant's organization. It is required that readers have working knowledge of programming languages, XML formats and UTF-8 encodings.

0.2 Authentication

To interact with the payment API, you need to provide login credentials using standard HTTP Basic Authentication. (credentials can be found in the your Admin interface.)

To decrease network traffic and response times, we recommend that you enforce sending authentication credentials directly in the first request.

Some implementations like e.g. the Java HTTPClient automatically try to guess the best authentication scheme. This can be overriden by setting the authorization to preemptive:

```
HTTPClient.getParams().setAuthenticationPreemptive(true);
```

0.3 URLs

0.3.1 Server API

API methods are called with the following structure:

```
https://skprocessing.hypercharge.net/process/CHANNEL_TOKEN/
```

Single transaction reconciles can be done via this URL:

```
https://skprocessing.hypercharge.net/reconcile/CHANNEL_TOKEN/
```

Date range reconciles can be done via this URL:

```
https://skprocessing.hypercharge.net/reconcile/by_date/CHANNEL_TOKEN/
```

Your admin panel can be found here:

```
https://sankyu.hypercharge.net/
```

0.3.2 WPF

The URL for the WPF API create method is:

```
https://payment.hypercharge.net/payment
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment
```

The URL for the WPF API reconcile method is:

```
https://payment.hypercharge.net/payment/reconcile
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment/reconcile
```

Note:

Channel token and login credentials can be found in your admin panel.
To use our testing environment, refer to chapter Testing

1 Testing

For testing first login to the gateway admin and create a channel.

The url for test admin is:

```
https://testmerchant.hypercharge.net/
```

The api base url for test processing is:

```
https://test.hypercharge.net/process/CHANNEL_TOKEN
```

The api base url for test single transaction reconciling is:

```
https://test.hypercharge.net/reconcile/CHANNEL_TOKEN
```

The api base url for test date range reconciling is:

```
https://test.hypercharge.net/reconcile/by_date/CHANNEL_TOKEN
```

For testing the gateway the following credit card numbers can be used:

card number	card brand	transaction result
4200000000000000	Visa	successful transaction
4111111111111111	Visa	transaction declined
5555555555554444	Master Card	successful transaction
5105105105105100	Master Card	transaction declined
378282246310005	American Express	successful transaction
371449635398431	American Express	transaction declined
30569309025904	Diners Club	successful transaction
38520000023237	Diners Club	transaction declined
6011111111111117	Discover	successful transaction
6011000990139424	Discover	transaction declined
3530111333300000	JCB	successful transaction
3566002020360505	JCB	transaction declined
5610591081018250	Australian BankCard	successful transaction
5105105105105100	Australian BankCard	transaction declined

For 3DSecure testing the following credit card numbers can be used:

card number	card brand	transaction result
4711100000000000	Visa	3DSecure enrolled
4012001037461114	Visa	3DSecure enrolled failing authentication
4200000000000000	Visa	3DSecure unavailable
		- Card Not Participating
4012001037484447	Visa	Error in 3DSecure Network in first step of 3DS authentication process
4012001036273338	Visa	Error in 3DSecure Network in
		second (asynchronous) step of 3DS authentication process

When redirected to the dummy authentication page you may enter any password you like.

For DebitSale payment_transactions testing the following Bank account number patterns can be used:

bank account number	transaction result
1...	approved transaction
2...	declined transaction
3...	error transaction

For DebitSale payment_transactions using SEPA testing the following IBAN patterns can be used:

iban	transaction result
DE1...	approved transaction
DE2...	declined transaction
DE3...	error transaction

For PurchaseOnAccount or PayInAdvance payment_transactions testing the following customer_email values can be used:

card number	transaction result
approved@...	approved transaction
declined@...	declined transaction
error@...	error transaction

Note:

In test mode, successful transaction XML responses will include the following error message: "TESTMODE: No real money will be transferred!"

1.1 Advanced Testing - Chargebacks, Retrieval Requests and ELV Workflows

The advanced testing interface provides additional testing scenarios beyond the standard integration of all transaction types. It provides the ability to simulate chargebacks, chargeback reversals, pre arbitrations, retrieval request on credit card related transactions and also simulates chargebacked, rejected and charged debit_sale transactions. Additionaly you can create deposits on purchase_on_account and pay_in_advance transactions. By using this interface, you can simulate a complete transaction lifecycle and test your backend systems to handle those events accordingly. Testing chargeback notifications, chargeback reports and retrieval reports is also possible if you provide a notification URL and an email address to receive the generated reports. To create an event, you send a POST requests to the following URLs using standard HTTP Basic Authentication and providing a channel token and the unique_id of the transaction to create the event for.

1.1.1 Create chargeback

```
https://test.hypercharge.net/bogus_event/chargeback/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>chargebacked</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.2 Create pre arbitration

```
https://test.hypercharge.net/bogus_event/pre_arbitration/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>pre_arbitrated</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
```

```
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.3 Create chargeback reversal

```
https://test.hypercharge.net/bogus_event/chargeback_reversal/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>chargeback_reversed</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.4 Create retrieval request

```
https://test.hypercharge.net/bogus_event/retrieval/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.5 Create deposit

```
https://test.hypercharge.net/bogus_event/deposit/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>purchase_on_account</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.6 Create debit chargeback

```
https://test.hypercharge.net/bogus_event/debit_chargeback/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>debit_sale</transaction_type>
  <status>chargebacked</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.7 Create rejected debit_sale

```
https://test.hypercharge.net/bogus_event/reject/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>debit_sale</transaction_type>
  <status>rejected</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

1.1.8 Create charged debit_sale

```
https://test.hypercharge.net/bogus_event/charge/CHANNEL_TOKEN/UNIQUE_ID
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>debit_sale</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>test</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2 Transactions

2.1 Invoking a transaction

A transaction is invoked via HTTPS POST, parameters are passed as XML with UTF-8 encoding.

2.2 Transaction types

2.2.1 Authorize

With authorize transactions, you can confirm that a credit card is valid and reserve the desired amount on the card.

After settling the transaction (e.g. shipping the goods), you can then **capture** the amount. The customer will not be billed until the capture has taken place, but the amount is reserved and the customer's credit card limit is reduced. Authorizes will automatically be cancelled after a certain timeframe, most likely one week.

For a typical e-commerce application it is recommended to authorize the amount on incoming orders and capture it when shipping the goods. If you are selling services or non-tangible goods, you can use the **sale** transaction, which combines authorize and capture.

If you chose not to serve the customer, consider to **void** the authorize to unfreeze the amount on the client's credit card.

Note:

Authorize transactions are also available as **3dsecure** transactions

Request:

parameter	req?	format	description
transaction_type	*required*	"authorize"	authorize
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on credit card (first name and last name at least)
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
card_number	*required*	13 to 16 digits	complete cc number of customer
cvv	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166

required* = conditinally required

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>authorize</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>02</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
```

```
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>authorize</transaction_type>
  <status>approved</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>43671</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>authorize</transaction_type>
  <status>error</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>43671</transaction_id>
  <code>340</code>
  <technical_message>’expiration_year’ is invalid</technical_message>
  <message>’expiration_year’ is invalid</message>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_response>
```

2.2.2 Capture

Capture settles a transaction which has been authorized before.

Do this when you are shipping goods, for example. A capture can only be used after an **authorize** on the same transaction.

Therefore, the reference_id of the authorized transaction is mandatory.

Note:

You can also capture a partial amount of the initially authorized amount, e.g. if you want to give customers a discount. However, you cannot capture a higher amount than initially authorized.

Note:

Depending on the acquirer this transaction might be processed as **sync** or **async**. See Asynchronous transaction and Notifications.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"capture"	capture
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`reference_id`	*required*	string(32)	unique id returned by **authorize**
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>capture</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <reference_id>2ee4287e67971380ef7f97d5743bb523</reference_id>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>capture</transaction_type>
  <status>approved</status>
  <unique_id>91449b0b7eb34dca6b0666cbfa8d5d03</unique_id>
  <transaction_id>119643223347501b69b921b</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:17:15Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be "live" or "test"
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>capture</transaction_type>
  <status>error</status>
  <unique_id>91a9adf1cc5095e483006df34b64023f</unique_id>
  <transaction_id>119643234047501bd47e6b4</transaction_id>
  <code>430</code>
  <technical_message>reference transaction has already been captured,
       and acquirer does not support partial/multiple capture</technical_message>
  <message>transaction declined.</message>
  <mode>live</mode>
  <timestamp>2007-11-30T14:19:02Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.2.3 Sale

Sale transactions combine authorize and capture into one step.

Using a sale transaction, the amount is immediately billed to the customer's credit card. It can be reversed via a **void** transaction on the same day of the transaction. Use sale transactions, if you are e.g. selling non-tangible goods or services.

Note:

Sale transactions are also available as **3dsecure** transactions

Note:

Depending on the acquirer this transaction might be processed as **sync** or **async**. See Asynchronous transaction and Notifications.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"sale"	sale
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	description of the transaction for later use
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`card_holder`	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
`card_number`	*required*	string(13..21)	credit card number
`cvv`	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
`expiration_month`	*required*	MM	Expiration month as printed on credit card
`expiration_year`	*required*	YYYY	Expiration year as printed on credit card
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*required*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

required* = conditionally required

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sale</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>error</status>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <transaction_id>1196439228475036bc3a791</transaction_id>
  <code>340</code>
  <technical_message>’billing_address[zip_code]’ is invalid!</technical_message>
  <message>’billing_address[zip_code]’ is invalid!</message>
  <mode>live</mode>
  <timestamp>2007-11-30T16:13:50Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.2.4 Refund

Refunds allow to return already billed amounts to customers.

The amount can be fully or partially refunded. Refunds can only be done on former **capture** or **sale** transactions.

Therefore, the reference_id for the corresponding transaction is mandatory.

Note:

Depending on the acquirer this transaction might be processed as **sync** or **async**. See Asynchronous transaction and Notifications.

Request:

parameter	req?	format	description
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`reference_id`	*required*	string(32)	unique id returned by corresponding transaction
`amount`	*required*	integer > 0	Amount of transaction in cents (Note: Partial amount of capture or sale can be refunded)
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>refund</transaction_type>
  <transaction_id>134rt6</transaction_id>
  <reference_id>57b3a7b166ffe873d0a11863560b410c</reference_id>
  <usage>40208 concert canceld</usage>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>refund</transaction_type>
  <status>approved</status>
  <unique_id>776cc3b95eaf9025113dc18d128e99a0</unique_id>
  <transaction_id>11964393414750372dad370</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T16:15:44Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>refund</transaction_type>
  <status>error</status>
  <unique_id>2515db3cf16d04fa35afb140b84b99d4</unique_id>
  <transaction_id>119643941047503772d9caa</transaction_id>
  <code>410</code>
  <technical_message>no approved reference transaction found</technical_message>
  <message>no approved reference transaction found</message>
  <mode>live</mode>
  <timestamp>2007-11-30T16:16:53Z</timestamp>
  <descriptor></descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.2.5 Void

Void transactions undo other transactions.

Transactions of types **authorize**, **capture**, **sale** and **refund** can be reversed on the same day the transaction took place. The transaction will not show up on the customer's credit card statement if void.

Note:

The **same day** is dependent of timezone of the acquiring bank. You can find the timezone of your acquiring bank in the admin interface.

Note:

Depending on the acquirer this transaction might be processed as **sync** or **async**. See Asynchronous transaction and Notifications.

Request:

parameter	req?	format	description
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`reference_id`	*required*	string(32)	unique id returned by corresponding transaction

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>void</transaction_type>
  <transaction_id>134rt78956</transaction_id>
  <reference_id>57b3a7b166ffe873d0a11863560b410c</reference_id>
  <usage>40208 concert canceled</usage>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>void</transaction_type>
  <status>approved</status>
  <unique_id>4b9e30b641d256d9283be7ef37ea4a0e</unique_id>
  <transaction_id>119643944447503794c8bb2</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T16:17:26Z</timestamp>
  <descriptor>descriptor one</descriptor>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>void</transaction_type>
  <status>error</status>
  <unique_id>5b85106ae3b3789f7242edd5f92c218b</unique_id>
  <transaction_id>1196439500475037ccabd94</transaction_id>
  <code>420</code>
  <technical_message>can not do void on void reference</technical_message>
  <message>can not do void on void reference</message>
  <mode>live</mode>
  <timestamp>2007-11-30T16:18:22Z</timestamp>
  <descriptor>descriptor one</descriptor>
</payment_response>
```

2.2.6 ReferencedFundTransfer

ReferencedFundTransfers allows you to transfer funds to a previously charged card.

The amount can be higher than the charged reference. **ReferencedFundTransfers** can only be done on former **sale**, **sale3d** or **capture** transaction.

Therefore, the `reference_id` for the corresponding transaction is mandatory.

Note:

ReferencedFundTransfers are not suppoted by all acquirers.

Note:

Depending on the acquirer this transaction might be processed as **sync** or **async**. See Asynchronous transaction and Notifications.

Request:

parameter	req?	format	description
`transaction_type`	*required*		"referenced_fund_transfer"
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`reference_id`	*required*	string(32)	unique id returned by corresponding transaction
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>referenced_fund_transfer</transaction_type>
  <transaction_id>B62D6848-78EA-4910-BFD8-920A3BF10CC5</transaction_id>
  <reference_id>57b3a7b166ffe873d0a11863560b410c</reference_id>
  <usage>40208 concert canceld</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>referenced_fund_transfer</transaction_type>
  <status>pending_async</status>
  <unique_id>776cc3b95eaf9025113dc18d128e99a0</unique_id>
  <transaction_id>B62D6848-78EA-4910-BFD8-920A3BF10CC5</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T16:15:44Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>referenced_fund_transfer</transaction_type>
  <status>error</status>
  <unique_id>2515db3cf16d04fa35afb140b84b99d4</unique_id>
  <transaction_id>B62D6848-78EA-4910-BFD8-920A3BF10CC5</transaction_id>
  <code>410</code>
  <technical_message>no approved reference transaction found</technical_message>
  <message>no approved reference transaction found</message>
  <mode>live</mode>
  <timestamp>2007-11-30T16:16:53Z</timestamp>
  <descriptor></descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

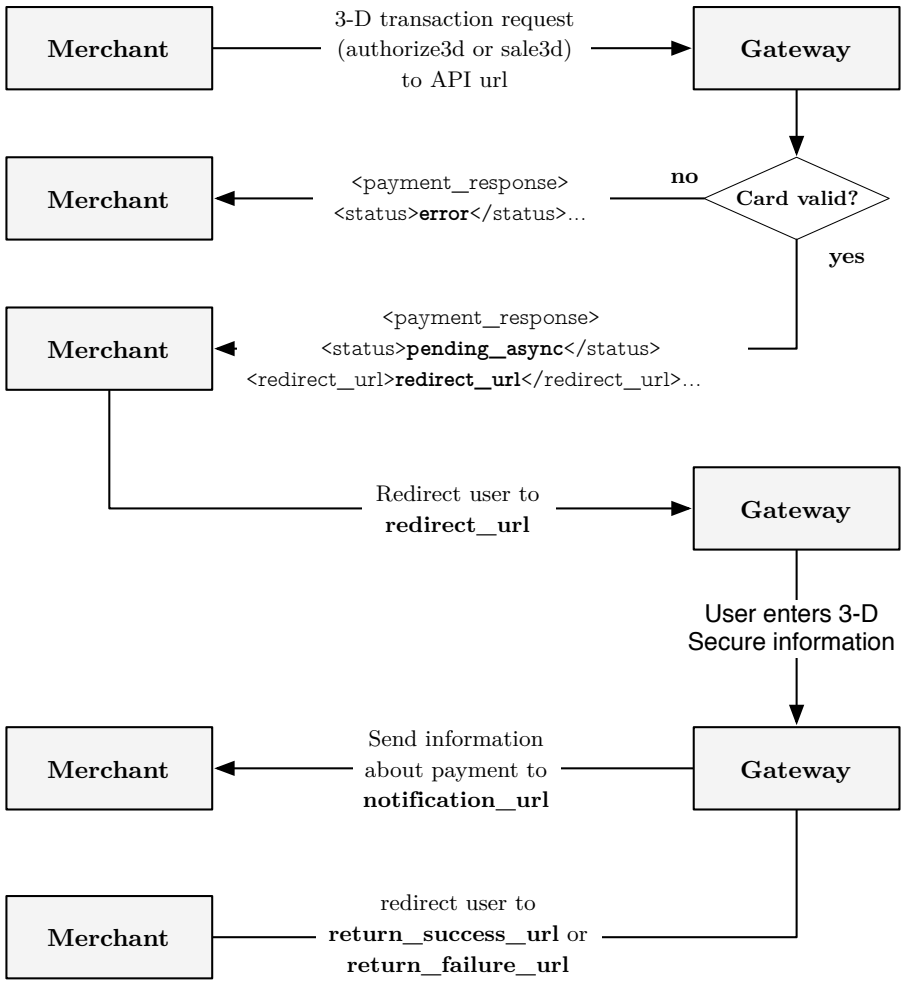
2.3 3-D Secure Transactions

Some contracts may require that transactions are conducted using 3-D secure technology. This means that a payment is processed asynchronously requiring the user to be redirected to a page where he enters personal data during the process. This improves security and reduces fraudulent transactions while shifting liability from merchants to the issuing banks.

VISA calls this technology **Verified by Visa** and Mastercard calls it **MasterCard SecureCode**.

The asynchronous workflow may take place on the merchants site in case the merchant has a MPI or wants to use another MPI. In this case a 3-D Secure transaction will be handled **synchronous** but requires a different parameter signature. See Authorize3d or Sale3d transactions for details.

In case the merchant does not have a MPI the **asynchronous** workflow is handled by hypercharge and looks like shown in the diagram below:



Transactions can fail prior to invoking the 3-D mechanism (in case of e.g. invalid card number, expiry date or risk checks). In this case, payment stays synchronous.

If the card is considered valid for 3-D, payment becomes asynchronous as the user gets redirected to

a form at the gateway where he enters additional information.

The merchant will be notified of the outcome via the notification_url.

He needs to supply both **return_success_url** and **return_failure_url** where the user will be redirected to after the payment has taken place.

2.3.1 Authorize3D

Authorize3D transactions basically have the same request as standard **authorize** transactions.

Note:

Authorize3D transactions can be handled **synchronous** or **asynchronous** depending on the parameters passed. If `mpi_params` is passed the workflow will be synchronous. If `notification_url`, `return_success_url` and `return_failure_url` are passed the workflow will be asynchronous.

Note:

To settle Authorize3D transactions, normal **capture** transactions are used. As the 3-D secure process already took place, there is no need to do the verification again when capturing.

Request :

parameter	req?	format	description
transaction_type	*required*	"authorize3d"	authorize3d
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
notification_url	*required*[1]	url	URL at merchant where gateway sends outcome of transaction after 3-D validation. This should be an SSL secured page.
return_success_url	*required*[1]	url	URL where customer is sent to after successful 3-D validation/payment
return_failure_url	*required*[1]	url	URL where customer is sent to after failed 3-D validation/payment
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on credit card (first name and last name at least)
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
card_number	*required*	13 to 16 digits	complete cc number of customer
cvv	*required*	3 to 4 digits	cvv of cc
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
mpi_params	*required*[2]		
cavv	*required*[3]	string(255)	Verification Id of the authentication. Please note this can be the CAVV for Visa Card or UCAF to identify MasterCard.
eci	*required*[3]	string(255)	Electric Commerce Indicator as retured from the MPI.
xid	*required*[3]	string(255)	Transaction ID generated by the 3D Secure service that uniquely identifies a 3D Secure check request

[1] required if mpi_params is not present, transaction will be handled **asynchronous**.
[2] required if transaction should be handled **synchronous**.
[3] required if mpi_params is present.

Example XML with **mpi_params** for **synchronous** workflow:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>authorize3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>02</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <mpi_params>
    <cavv>AAACA1BHADYJkIASQkcAAAAAAAA=</cavv>
    <eci>05</eci>
    <xid>0pv62FIrT5qQODB7DCewKgEBAQI=</xid>
  </mpi_params>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Example XML with **notification_url**, **return_success_url** and **return_failure_url** for an **asynchronous** workflow handled by hypercharge:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>authorize3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>02</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
```

```
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be "live" or "test"
redirect_url	url	URL where user has to be redirected to complete payment process
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML (in **synchronous** mode):

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>authorize3d</transaction_type>
  <status>approved</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>43671</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_response>
```

Response XML (in **asynchronous** mode):

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>authorize3d</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>43671</transaction_id>
  <mode>live</mode>
  <redirect_url>https://3dsecured.hypercharge.net/8245201941/30ec0f2387</redirect_url>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>authorize3d</transaction_type>
  <status>error</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>43671</transaction_id>
  <code>340</code>
  <technical_message>’expiration_year’ is invalid</technical_message>
  <message>’expiration_year’ is invalid</message>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>USD</currency>
</payment_response>
```

2.3.2 Sale3D

Sale3D transactions basically have the same request as standard **sale** transactions.

Note:

Sale3D transactions can be handled **synchronous** or **asynchronous** depending on the parameters passed. If `mpi_params` is passed the workflow will be **synchronous**. If `notification_url`, `return_success_url` and `return_failure_url` are passed the workflow will be **asynchronous**.

Request:

parameter	req?	format	description
transaction_type	*required*	"sale3d"	sale3d
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
notification_url	*required*[1]	url	URL at merchant where gateway sends outcome of transaction after 3-D validation. This should be an SSL secured page.
return_success_url	*required*[1]	url	URL where customer is sent to after successful 3-D validation/payment
return_failure_url	*required*[1]	url	URL where customer is sent to after failed 3-D validation/payment
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
card_number	*required*	string(13..21)	credit card number
cvv	*required*	integer(3..4)	cvv of cc
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
mpi_params	*required*[2]		
cavv	*required*[3]	string(255)	Verification Id of the authentication. Please note this can be the CAVV for Visa Card or UCAF to identify MasterCard.
eci	*required*[3]	string(255)	Electric Commerce Indicator as retured from the MPI.
xid	*required*[3]	string(255)	Transaction ID generated by the 3D Secure service that uniquely identifies a 3D Secure check request

[1] required if mpi_params is not present, transaction will be handled **asynchronous**.
[2] required if transaction should be handled **synchronous**.
[3] required if mpi_params is present.

Example XML with mpi params for **synchronous** workow:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sale3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <mpi_params>
    <cavv>AAACA1BHADYJkIASQkcAAAAAAAA=</cavv>
    <eci>05</eci>
    <xid>0pv62FIrT5qQODB7DCewKgEBAQI=</xid>
  </mpi_params>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Example XML with **notification_url**, **return_success_url** and **return_failure_url** for an **asynchronous** workflow handled by hypercharge:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sale3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
```

```
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML (in **synchronous** mode):

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale3d</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

Response XML (in **asynchronous** mode):

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale3d</transaction_type>
  <status>pending_async</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <redirect_url>https://3dsecured.hypercharge.net/8245201941/30ec0f2387</redirect_url>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale3d</transaction_type>
  <status>error</status>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <transaction_id>1196439228475036bc3a791</transaction_id>
  <code>340</code>
  <technical_message>’billing_address[zip_code]’ is invalid!</technical_message>
  <message>’billing_address[zip_code]’ is invalid!</message>
  <mode>live</mode>
  <timestamp>2007-11-30T16:13:50Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.3.3 Notification for asynchronous payments

When using asynchronous payments like 3-D secured transactions, the transaction depends on user input. Therefore, the outcome of the transaction is not available immediately after sending the request.

The gateway will send a notification to the merchant's server as soon as the payment has been completed (either approved or declined).

If the user cancels the payment (or the user doesn't complete the payment within the given time frame, e.g. 2 hours), the transaction will time out and a notification will be sent.

The notification will be sent to the **notification_url** specified in the transaction request XML. See Notifications for the HTTP POST-Data and format. Also see 3-D secure workflow.

2.4 Recurring Transactions

A recurring transaction describes a payment where the cardholder's account is periodically charged for a repeated delivery and use of a product or service (subscription, membership fee, etc.) over time. A recurring payment consists of an initial transaction and one or several repeated transactions. The "initial" transaction contains all relevant card and cardholder data, while the subsequent repeated transaction references an identifier which is returned with the response to the initial request.

2.4.1 InitRecurringSale

A InitRecurringSale transaction initalizes a recurring payment and is equal to a normal SaleTransaction except that it can be referenced as "initial" transaction in a RecurringSale transaction.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"init_recurring_sale"	sale
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	description of the transaction for later use
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`card_holder`	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
`card_number`	*required*	string(13..21)	credit card number
`cvv`	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
`expiration_month`	*required*	MM	Expiration month as printed on credit card
`expiration_year`	*required*	YYYY	Expiration year as printed on credit card
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*required*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

required* = conditionally required

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_sale</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_sale</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.4.2 InitRecurringSale3d

A InitRecurringSale3d transaction initalizes a recurring payment and is equal to a normal Sale3dTransaction except that it can be referenced as "initial" transaction in a recurring transaction.

Note:

InitRecurringSale3d transactions can be handled **synchronous** or **asynchronous** depending on the parameters passed. If `mpi_params` is passed the workflow will be **synchronous**. If `notification_url`, `return_success_url` and `return_failure_url` are passed the workflow will be **asynchronous**.

Request:

parameter	req?	format	description
transaction_type	*required*	"init_recurring_sale3d"	3D
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
notification_url	*required*[1]	url	URL at merchant where gateway sends outcome of transaction after 3-D validation. This should be an SSL secured page.
return_success_url	*required*[1]	url	URL where customer is sent to after successful 3-D validation/payment
return_failure_url	*required*[1]	url	URL where customer is sent to after failed 3-D validation/payment
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
card_number	*required*	string(13..21)	credit card number
cvv	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
mpi_params	*required*[2]		
cavv	*required*[3]	string(255)	Verification Id of the authentication. Please note this can be the CAVV for Visa Card or UCAF to identify MasterCard.
eci	*required*[3]	string(255)	Electric Commerce Indicator as retured from the MPI.
xid	*required*[3]	string(255)	Transaction ID generated by the 3D Secure service that uniquely identifies a 3D Secure check request

required* = conditionally required

[1] required if mpi_params is not present, transaction will be handled **asynchronous**.
[2] required if transaction should be handled **synchronous**.
[3] required if mpi_params is present.

Example XML with mpi params for **synchronous** workow:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_sale3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <mpi_params>
    <cavv>AAACA1BHADYJkIASQkcAAAAAAAA=</cavv>
    <eci>05</eci>
    <xid>0pv62FIrT5qQODB7DCewKgEBAQI=</xid>
  </mpi_params>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Example XML with **notification_url**, **return_success_url** and **return_failure_url** for an **asynchronous** workflow handled by hypercharge:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sale3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
```

```
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_sale3d</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.4.3 InitRecurringAuthorize

A InitRecurringAuthorize transaction initalizes a recurring payment and is equal to a normal AuthorizeTransaction except that it can be referenced as "initial" transaction in a RecurringSale transaction.

Request:

parameter	req?	format	description
transaction_type	*required*	"init_recurring_authorize"	authorize
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
card_number	*required*	string(13..21)	credit card number
cvv	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166

required* = conditionally required

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_authorize</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be "live" or "test"
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_authorize</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.4.4 InitRecurringAuthorize3d

A InitRecurringAuthorize3d transaction initalizes a recurring payment and is equal to a normal Authorize3dTransaction except that it can be referenced as "initial" transaction in a recurring transaction.

Note:

InitRecurringAuthorize3d transactions can be handled **synchronous** or **asynchronous** depending on the parameters passed. If `mpi_params` is passed the workflow will be synchronous. If `notification_url`, `return_success_url` and `return_failure_url` are passed the workflow will be asynchronous.

Note:

To settle InitRecurringAuthorize3d transactions, normal **capture** transactions are used. As the 3-D secure process already took place, there is no need to do the verification again when capturing.

Request:

parameter	req?	format	description
transaction_type	*required*	"init_recurring_authorize3d"	authorize3d
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
notification_url	*required*[1]	url	URL at merchant where gateway sends outcome of transaction after 3-D validation. This should be an SSL secured page.
return_success_url	*required*[1]	url	URL where customer is sent to after successful 3-D validation/payment
return_failure_url	*required*[1]	url	URL where customer is sent to after failed 3-D validation/payment
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
card_number	*required*	string(13..21)	credit card number
cvv	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
mpi_params	*required*[2]		
cavv	*required*[3]	string(255)	Verification Id of the authentication. Please note this can be the CAVV for Visa Card or UCAF to identify MasterCard.
eci	*required*[3]	string(255)	Electric Commerce Indicator as retured from the MPI.
xid	*required*[3]	string(255)	Transaction ID generated by the 3D Secure service that uniquely identifies a 3D Secure check request

[1] required if mpi_params is not present, transaction will be handled **asynchronous**.
[2] required if transaction should be handled **synchronous**.
[3] required if mpi_params is present.

required* = conditionally required

Example XML with **mpi_params** for **synchronous** workflow:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_authorize3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>02</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <mpi_params>
    <cavv>AAACA1BHADYJkIASQkcAAAAAAAA=</cavv>
    <eci>05</eci>
    <xid>0pv62FIrT5qQODB7DCewKgEBAQI=</xid>
  </mpi_params>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Example XML with **notification_url**, **return_success_url** and **return_failure_url** for an **asynchronous** workflow handled by hypercharge:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>authorize3d</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>02</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
```

```
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_authorize3d</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.4.5 RecurringSale

A RecurringSale transaction is a "repeated" transaction which follows and references a InitRecurringSale or a InitRecurringAuthorize transaction. The card and cardholder data is omitted.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"recurring_sale"	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`reference_id`	*required*	string(32)	unique id returned by InitRecurringSale or InitRecurringAuthorize
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>recurring_sale</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <reference_id>2ee4287e67971380ef7f97d5743bb523</reference_id>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>recurring_sale</transaction_type>
  <status>approved</status>
  <unique_id>91449b0b7eb34dca6b0666cbfa8d5d03</unique_id>
  <transaction_id>119643223347501b69b921b</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:17:15Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.4.6 RecurringAuthorize

A RecurringAuthorize transaction is a "repeated" transaction which follows and references a InitRecurringAuthorize or a InitRecurringAuthorize transaction. The card and cardholder data is omitted.

With authorize transactions, you can confirm that a credit card is valid and reserve the desired amount on the card.

After settling the transaction (e.g. shipping the goods), you can then **capture** the amount. The customer will not be billed until the capture has taken place, but the amount is reserved and the customer's credit card limit is reduced. Authorizes will automatically be cancelled after a certain timeframe, most likely one week.

For a typical e-commerce application it is recommended to authorize the amount on incoming orders and capture it when shipping the goods. If you are selling services or non-tangible goods, you can use the **sale** transaction, which combines authorize and capture.

If you chose not to serve the customer, consider to **void** the authorize to unfreeze the amount on the client's credit card.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"recurring_authorize"	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`reference_id`	*required*	string(32)	unique id returned by InitRecurringAuthorize or InitRecurringAuthorize
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>recurring_authorize</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>club membership 2009-06</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <reference_id>2ee4287e67971380ef7f97d5743bb523</reference_id>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>recurring_authorize</transaction_type>
  <status>approved</status>
  <unique_id>91449b0b7eb34dca6b0666cbfa8d5d03</unique_id>
  <transaction_id>119643223347501b69b921b</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:17:15Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.5 Alternative Payment Menthods

Alternative payment method's like PayPal, PaySafeCard, etc.

2.5.1 PayPal

PayPal is an asynchronous PayPal-ExpressCheckout transaction.

Like all asynchronous transactions, PayPal follows a redirect workflow as described for 3d-Secure Transactions and a Notification is sent to the merchant once the payment has reached a final status.

Request:

parameter	req?	format	description
`transaction_type`	*required*	`pay_pal`	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`notification_url`	*required*	url	URL at merchant where gateway sends outcome of transaction. This should be an SSL secured page.
`return_success_url`	*required*	url	URL where customer is sent to after successful payment
`return_failure_url`	*required*	url	URL where customer is sent to after failed payment
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`billing_address`	*required*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>pay_pal</transaction_type>
  <transaction_id>145236</transaction_id>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
redirect_url	url	URL where user has to be redirected to complete payment process
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>pay_pal</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <redirect_url>https://hypercharge.net/redirect/...</redirect_url>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.5.2 Barzahlen.de

Barzahlen is an asynchronous transaction.

Like all asynchronous transactions, Barzahlen.de follows a redirect workflow as described for 3d-Secure Transactions and a Notification is sent to the merchant once the payment has reached a final status.

Request:

parameter	req?	format	description
transaction_type	*required*	barzahlen	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
notification_url	*required*	url	URL at merchant where gateway sends outcome of transaction. This should be an SSL secured page.
return_success_url	*required*	url	URL where customer is sent to after successful payment
return_failure_url	*required*	url	URL where customer is sent to after failed payment
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>barzahlen</transaction_type>
  <transaction_id>145236</transaction_id>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
redirect_url	url	URL where user has to be redirected to complete payment process
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>barzahlen</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <redirect_url>https://hypercharge.net/redirect/...</redirect_url>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6 Bank Account Transfer Transactions

A bank account transfer transaction describes a payment where the money is debited from the customer's bank account.

2.6.1 DebitSale

DebitSale is a synchronous bank account transfer transaction. As with Sale transactions the money will be debited directly from the customer's bank account. Successful states can be 'approved' or 'pending_async'. In case the status is 'pending_async' Hypercharge will track the successful debit and change the transaction status to 'approved' when the money has been wired. A notification can be sent to a pre-configured notification URL if the amount could not be booked or the customer initiates a chargeback. Guaranteed direct debits require a birthdate risk parameter like PurchaseOnAccount.

Request:

parameter	req?	format	description
transaction_type	*required*	debit_sale	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
bank_account_number	*conditionally*	string(16)	customers bank account number
bank_number	*conditionally*	string(16)	customers bank number
iban	*conditional*[1]	string(31)	customers international bank account number (SEPA)
bic	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*conditional*[1]	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*conditional*[1]	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

DebitSale also supports Risk params.

[1] conditionally bank_account_number and bank_number or iban, bic, sepa_mandate_id and sepa_mandate_signature_date are required. iban, bic, sepa_mandate_id and sepa_mandate_signature_date are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>debit_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>debit_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.2 InitRecurringDebitSale

InitRecurringDebitSale initalizes a recurring a synchronous bank account transfer transaction and is equal to a normal DebitSaleTransaction except that it can be referenced as "initial" transaction in a RecurringDebitSale transaction.

As with Sale transactions the money will be debited directly from the customer's bank account. Successful states can be 'approved' or 'pending_async'. In case the status is 'pending_async' Hypercharge will track the successful debit and change the transaction status to 'approved' when the money has been wired. A notification can be sent to a pre-configured notification URL if the amount could not be booked or the customer initiates a chargeback.

Request:

parameter	req?	format	description
transaction_type	*required*	init_recurring_debit_sale	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
bank_account_number	*conditional*[1]	string(16)	customers bank account number
bank_number	*conditional*[1]	string(16)	customers bank number
iban	*conditional*[1]	string(31)	customers international bank account number (SEPA)
bic	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*conditional*[1]	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*conditional*[1]	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

InitRecurringDebitSale also supports Risk params.

[1] conditionally bank_account_number and bank_number or iban, bic, sepa_mandate_id and sepa_mandate_signature_date are required. iban, bic, sepa_mandate_id and sepa_mandate_signature_date are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_debit_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_debit_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.3 InitRecurringDebitAuthorize

InitRecurringDebitAuthorize initalizes a recurring a synchronous bank account transfer transaction and is equal to a normal DebitSaleTransaction except that it can be referenced as "initial" transaction in a RecurringDebitSale transaction.

As with Authorize transactions the money will *NOT* be debited directly from the customer's bank account. Successful state is 'approved'. Further more there is no way to capture a InitRecurringDebitAuthorizeTranscation. It only exists to validate a bank account and initialize a recurring chain.

Request:

parameter	req?	format	description
transaction_type	*required*	init_recurring_debit_authorize	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
bank_account_number	*conditional*[1]	string(16)	customers bank account number
bank_number	*conditional*[1]	string(16)	customers bank number
iban	*conditional*[1]	string(31)	customers international bank account number (SEPA)
bic	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*conditional*[1]	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*conditional*[1]	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

InitRecurringDebitAuthorize also supports Risk params.

[1] conditionally bank_account_number and bank_number or iban, bic, sepa_mandate_id and sepa_mandate_signature_date are required. iban, bic, sepa_mandate_id and sepa_mandate_signature_date are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_debit_authorize</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_debit_authorize</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.4 RecurringDebitSale

A RecurringDebitSale transaction is a "repeated" transaction which follows and references a InitRecurringDebitSale or a InitRecurringDebitAuthorize transaction. The bank account and bank account holder data is omitted.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"recurring_debit_sale"	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`reference_id`	*required*	string(32)	unique id returned by InitRecurringDebitSale or InitRecurringDebitAuthorize
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>recurring_debit_sale</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <reference_id>2ee4287e67971380ef7f97d5743bb523</reference_id>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>recurring_debit_sale</transaction_type>
  <status>approved</status>
  <unique_id>91449b0b7eb34dca6b0666cbfa8d5d03</unique_id>
  <transaction_id>119643223347501b69b921b</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:17:15Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

2.6.5 SepaDebitSale

SepaDebitSale is a synchronous bank account transfer transaction. As with Sale transactions the money will be debited directly from the customer's bank account. Successful states can be 'approved' or 'pending_async'. In case the status is 'pending_async' Hypercharge will track the successful debit and change the transaction status to 'approved' when the money has been wired. A notification can be sent to a pre-configured notification URL if the amount could not be booked or the customer initiates a chargeback. Guaranteed direct debits require a birthdate risk parameter like PurchaseOnAccount.

Request:

parameter	req?	format	description
transaction_type	*required*	sepa_debit_sale	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
iban	*required*	string(31)	customers international bank account number (SEPA)
bic	*required*	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*required*	string(35)	signed mandate ID between merchant and customer
sepa_mandate_signature_date	*required*	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

SepaDebitSale also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sepa_debit_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sepa_debit_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.6 InitRecurringSepaDebitSale

InitRecurringSepaDebitSale initalizes a recurring a synchronous bank account transfer transaction and is equal to a normal SepaDebitSaleTransaction except that it can be referenced as "initial" transaction in a RecurringDebitSale transaction.

As with Sale transactions the money will be debited directly from the customer's bank account. Successful states can be 'approved' or 'pending_async'. In case the status is 'pending_async' Hypercharge will track the successful debit and change the transaction status to 'approved' when the money has been wired. A notification can be sent to a pre-configured notification URL if the amount could not be booked or the customer initiates a chargeback.

Request:

parameter	req?	format	description
transaction_type	*required*	init_recurring_sepa_debit_sale	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
iban	*required*	string(31)	customers international bank account number (SEPA)
bic	*required*	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*required*	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*required*	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

InitRecurringSepaDebitSale also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_sepa_debit_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_sepa_debit_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.7 InitRecurringSepaDebitAuthorize

InitRecurringSepaDebitAuthorize initalizes a recurring a synchronous bank account transfer transaction and is equal to a normal SepaDebitSaleTransaction except that it can be referenced as "initial" transaction in a RecurringDebitSale transaction.

As with Authorize transactions the money will *NOT* be debited directly from the customer's bank account. Successful state is 'approved'. Further more there is no way to capture a InitRecurringSepaDebitAuthorizeTranscation. It only exists to validate a bank account and initialize a recurring chain.

Request:

parameter	req?	format	description
transaction_type	*required*	init_recurring_sepa_debit_authorize	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
iban	*required*	string(31)	customers international bank account number (SEPA)
bic	*required*	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*required*	string(35)	signed mandate ID between merchant and customer
sepa_mandate_signature_date	*required*	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

InitRecurringSepaDebitAuthorize also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_sepa_debit_authorize</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_sepa_debit_authorize</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.8 RecurringSepaDebitSale

A RecurringSepaDebitSale transaction is a "repeated" transaction which follows and references a InitRecurringSepaDebitSale or a InitRecurringSepaDebitAuthorize transaction. The bank account and bank account holder data is omitted.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"recurring_sepa_debit_sale"	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`reference_id`	*required*	string(32)	unique id returned by InitRecurringSepaDebitSale or InitRecurringSepaDebitAuthorize
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
    <payment_transaction>
    <transaction_type>recurring_sepa_debit_sale</transaction_type>
  <transaction_id>43671</transaction_id>
<usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
<amount>5000</amount>
  <currency>USD</currency>
<reference_id>2ee4287e67971380ef7f97d5743bb523</reference_id>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
    <payment_response>
    <transaction_type>recurring_sepa_debit_sale</transaction_type>
  <status>approved</status>
<unique_id>91449b0b7eb34dca6b0666cbfa8d5d03</unique_id>
  <transaction_id>119643223347501b69b921b</transaction_id>
<mode>live</mode>
  <timestamp>2007-11-30T14:17:15Z</timestamp>
<descriptor>descriptor one</descriptor>
  <amount>9000</amount>
<currency>USD</currency>
</payment_response>
```

2.6.9 GtdSepaDebitSale

GtdSepaDebitSale is a synchronous bank account transfer transaction. As with Sale transactions the money will be debited directly from the customer's bank account. Successful states can be 'approved' or 'pending_async'. In case the status is 'pending_async' Hypercharge will track the successful debit and change the transaction status to 'approved' when the money has been wired. A notification can be sent to a pre-configured notification URL if the amount could not be booked or the customer initiates a chargeback. Guaranteed direct debits require a birthdate risk parameter like GtdPurchaseOnAccount.

Note:

The prefix gtd stands for guaranteed. According to your contract factoring will be applied to your gtd transactions.

Request:

parameter	req?	format	description
transaction_type	*required*	gtd_sepa_debit_sale	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
iban	*required*	string(31)	customers international bank account number (SEPA)
bic	*required*	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*required*	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*required*	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

GtdSepaDebitSale also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>gtd_sepa_debit_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
  <risk_params>
    <birthday>1978-03-03</birthday>
  </risk_params>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>gtd_sepa_debit_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.10 InitRecurringGtdSepaDebitSale

InitRecurringGtdSepaDebitSale initalizes a recurring a synchronous bank account transfer transaction and is equal to a normal GtdSepaDebitSaleTransaction except that it can be referenced as "initial" transaction in a RecurringGtdSepaDebitSale transaction.

As with Sale transactions the money will be debited directly from the customer's bank account. Successful states can be 'approved' or 'pending_async'. In case the status is 'pending_async' Hypercharge will track the successful debit and change the transaction status to 'approved' when the money has been wired. A notification can be sent to a pre-configured notification URL if the amount could not be booked or the customer initiates a chargeback. Guaranteed direct debits require a birthdate risk parameter like GtdPurchaseOnAccount.

Note:

The prefix gtd stands for guaranteed. According to your contract factoring will be applied to your gtd transactions.

Request:

parameter	req?	format	description
transaction_type	*required*	init_recurring_gtd_sepa_debit_sale	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
iban	*required*	string(31)	customers international bank account number (SEPA)
bic	*required*	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*required*	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*required*	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

InitRecurringGtdSepaDebitSale also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_gtd_sepa_debit_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
  <risk_params>
    <birthday>1978-03-03</birthday>
  </risk_params>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_gtd_sepa_debit_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.11 InitRecurringGtdSepaDebitAuthorize

InitRecurringGtdSepaDebitAuthorize initalizes a recurring a synchronous bank account transfer transaction and is equal to a normal GtdSepaDebitSaleTransaction except that it can be referenced as "initial" transaction in a RecurringGtdSepaDebitSale transaction.

As with Authorize transactions the money will *NOT* be debited directly from the customer's bank account. Successful state is 'approved'. Further more there is no way to capture a InitRecurringGtdSepaDebitAuthorizeTranscation. It only exists to validate a bank account and initialize a recurring chain. Guaranteed direct debits require a birthdate risk parameter like GtdPurchaseOnAccount.

Note:

The prefix gtd stands for guaranteed. According to your contract factoring will be applied to your gtd transactions.

Request:

parameter	req?	format	description
transaction_type	*required*	init_recurring_gtd_sepa_debit_authorize	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
wire_reference_id	*optional*	string(10)	Wire reference id for the transaction to show up on the bank statement
bank_account_holder	*required*	string(255)	name of the bank account holder
iban	*required*	string(31)	customers international bank account number (SEPA)
bic	*required*	string(11)	Business Identifier Codes / Swift ID (SEPA)
sepa_mandate_id	*required*	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*required*	date	mandate signature date; Format: YYYY-MM-DD
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
risk_params	*conditional*		Depends on service provider. Contact support for further information.
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

InitRecurringGtdSepaDebitAuthorize also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>init_recurring_gtd_sepa_debit_authorize</transaction_type>
  <transaction_id>145236</transaction_id>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
  <risk_params>
    <birthday>1978-03-03</birthday>
  </risk_params>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string(10)	Wire reference id for the transaction to show up on the bank statement
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>init_recurring_gtd_sepa_debit_authorize</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.12 RecurringGtdSepaDebitSale

A RecurringGtdSepaDebitSale transaction is a "repeated" transaction which follows and references a InitRecurringGtdSepaDebitSale or a InitRecurringGtdSepaDebitAuthorize transaction. The bank account and bank account holder data is omitted.

Request:

parameter	req?	format	description
transaction_type	*required*	"recurring_gtd_sepa_debit_sale"	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
reference_id	*required*	string(32)	unique id returned by InitRecurringGtdSepaDebitSale or InitRecurringGtdSepaDebitAuthorize
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
    <payment_transaction>
    <transaction_type>recurring_gtd_sepa_debit_sale</transaction_type>
  <transaction_id>43671</transaction_id>
<usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
<amount>5000</amount>
  <currency>USD</currency>
<reference_id>2ee4287e67971380ef7f97d5743bb523</reference_id>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
    <payment_response>
    <transaction_type>recurring_gtd_sepa_debit_sale</transaction_type>
  <status>approved</status>
<unique_id>91449b0b7eb34dca6b0666cbfa8d5d03</unique_id>
  <transaction_id>119643223347501b69b921b</transaction_id>
<mode>live</mode>
  <timestamp>2007-11-30T14:17:15Z</timestamp>
<descriptor>descriptor one</descriptor>
  <amount>9000</amount>
<currency>USD</currency>
</payment_response>
```

2.6.13 IdealSale

IdealSale is an asynchronous bank account transfer transaction which is only available to customers in the Netherlands. As with Sale transactions the money will be debited directly from the customer's bank account.

Like all asynchronous transactions, IdealSale follows a redirect workflow as described for 3d-Secure Transactions and a Notification is sent to the merchant once the payment has reached a final status.

As the customer will be prompted to authorize the transaction via his/her bank's website, a Chargeback will not be possible.

Request:

parameter	req?	format	description
`transaction_type`	*required*	`ideal_sale`	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`notification_url`	*required*	url	URL at merchant where gateway sends outcome of transaction. This should be an SSL secured page.
`return_success_url`	*required*	url	URL where customer is sent to after successful payment
`return_failure_url`	*required*	url	URL where customer is sent to after failed payment
`bank_account_holder`	*required*	string(255)	name of the bank account holder
`bank_name`	*conditional*[1]	constant	name of the bank. See available bank names table
`bank_account_number`	*conditional*[1]	string(16)	account number
`bank_number`	*conditional*[1]	string(16)	the bank's routing number
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*required*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	constant(NL)	only NL is valid as country code

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>ideal_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <bank_name>RABOBANK</bank_name>
  <bank_account_holder>Thomas van der Landen</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>thomas@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Thomas</first_name>
    <last_name>van der Landen</last_name>
    <address1>Boschdijk 1092</address1>
    <zip_code>5631 AV</zip_code>
    <city>Eindhoven</city>
    <state></state>
    <country>NL</country>
  </billing_address>
</payment_transaction>
```

[1] either `bank_name` or `bank_account_number` and `bank_number` are required. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
redirect_url	url	URL where user has to be redirected to complete payment process
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>ideal_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <redirect_url>https://hypercharge.net/redirect/...</redirect_url>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>ideal_sale</transaction_type>
  <status>error</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <code>340</code>
  <technical_message>’bank_name’ is invalid</technical_message>
  <message>’bank_name’ is invalid</message>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

Banknames for IdealSale:

XML tag Value	Full bank name to display on web page
ABN_AMRO	ABN AMRO
ASN_BANK	ASN Bank
ING	ING
KNAB	Knab
RABOBANK	Rabobank
REGIO_BANK	RegioBank
SNS_BANK	SNS Bank
TRIODOS_BANK	Triodos Bank
VAN_LANSCHOT	Van Lanschot

Note:

Please use FORTIS_TEST for testing on the test system. The other banks will not work on the test system.

It is possible to force FORTS_TEST to return certain results on the amount sent in:

Amount in EURO	Result	Transaction status
EUR 1	Successful Transaction	approved
EUR 2	User has cancelled	error
EUR 3	User has closed the browser window	error
EUR 5	General error during payment process	error

2.6.14 DirectPay24Sale

DirectPay24 (Sofortberweisung.de) is an asynchronous bank account transfer transaction which is only available to customers in the Europe. As with Sale transactions the money will be debited directly from the customer's bank account.

Like all asynchronous transactions, DirectPay24 follows a redirect workflow as described for 3d-Secure Transactions and a Notification is sent to the merchant once the payment has reached a final status.

As the customer will be prompted to authorize the transaction via his/her bank's website, a Chargeback will not be possible.

Request:

parameter	req?	format	description
`transaction_type`	*required*	`direct_pay24_sale`	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`notification_url`	*required*	url	URL at merchant where gateway sends outcome of transaction. This should be an SSL secured page.
`return_success_url`	*required*	url	URL where customer is sent to after successful payment
`return_failure_url`	*required*	url	URL where customer is sent to after failed payment
`bank_account_holder`	*conditional*[1]	string(255)	name of the bank account holder
`bank_account_number`	*conditional*[1]	string(16)	customers bank account number
`bank_number`	*conditional*[1]	string(16)	customers bank number
`iban`	*conditional*[1]	string(31)	customers international bank account number (SEPA)
`bic`	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*conditional*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>direct_pay24_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <bank_account_holder>Thomas van der Landen</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>thomas@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Thomas</first_name>
    <last_name>van der Landen</last_name>
    <address1>Boschdijk 1092</address1>
    <zip_code>5631 AV</zip_code>
    <city>Eindhoven</city>
    <state></state>
    <country>NL</country>
  </billing_address>
</payment_transaction>
```

[1] conditionally `bank_account_holder`, `bank_account_number`, `bank_number`, `iban`, `bic` or none of them are required. `iban` and `bic` are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
redirect_url	url	URL where user has to be redirected to complete payment process
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>direct_pay24_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <redirect_url>https://hypercharge.net/redirect/...</redirect_url>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>giro_pay_sale</transaction_type>
  <status>error</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <code>340</code>
  <technical_message>’bank_name’ is invalid</technical_message>
  <message>’bank_name’ is invalid</message>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.15 GiroPaySale

GiroPaySale is an asynchronous bank account transfer transaction which is only available to customers in the Europe. As with Sale transactions the money will be debited directly from the customer's bank account.

Like all asynchronous transactions, GiroPay follows a redirect workflow as described for 3d-Secure Transactions and a Notification is sent to the merchant once the payment has reached a final status.

As the customer will be prompted to authorize the transaction via his/her bank's website, a Chargeback will not be possible.

Request:

parameter	req?	format	description
`transaction_type`	*required*	`giro_pay_sale`	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`notification_url`	*required*	url	URL at merchant where gateway sends outcome of transaction. This should be an SSL secured page.
`return_success_url`	*required*	url	URL where customer is sent to after successful payment
`return_failure_url`	*required*	url	URL where customer is sent to after failed payment
`bank_account_holder`	*conditional*[1]	string(255)	name of the bank account holder
`bank_account_number`	*conditional*[1]	string(16)	customers bank account number
`bank_number`	*conditional*[1]	string(16)	customers bank number
`iban`	*conditional*[1]	string(31)	customers international bank account number (SEPA)
`bic`	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*conditional*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>giro_pay_sale</transaction_type>
  <transaction_id>145236</transaction_id>
  <notification_url>https://www.example.com/notification</notification_url>
  <return_success_url>http://www.example.com/success.html</return_success_url>
  <return_failure_url>http://www.example.com/failure.html</return_failure_url>
  <bank_account_holder>Thomas van der Landen</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>thomas@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Thomas</first_name>
    <last_name>van der Landen</last_name>
    <address1>Boschdijk 1092</address1>
    <zip_code>5631 AV</zip_code>
    <city>Eindhoven</city>
    <state></state>
    <country>NL</country>
  </billing_address>
</payment_transaction>
```

[1] conditionally `bank_account_holder`, `bank_account_number`, `bank_number`, `iban`, `bic` or none of them are required. `iban` and `bic` are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
redirect_url	url	URL where user has to be redirected to complete payment process
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>giro_pay_sale</transaction_type>
  <status>pending_async</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <redirect_url>https://hypercharge.net/redirect/...</redirect_url>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

Error response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 20071023T1200Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>giro_pay_sale</transaction_type>
  <status>error</status>
  <unique_id>011e8d5cc1a56058cc50440c264f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <code>340</code>
  <technical_message>’bank_name’ is invalid</technical_message>
  <message>’bank_name’ is invalid</message>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.16 PurchaseOnAccount

PurchaseOnAccount is a synchronous invoice based transaction. The money will not be debited from the customer's bank account. Instead a wire_reference_id is returned, which must be presented to the customer. The customer then needs to wire the money to your merchant bank account by himself referencing the wire_reference_id. The returned status will be returned as 'pending_async' or 'approved' if the transaction was accepted. In case the status is 'pending_async' Hypercharge will track the incoming bank transfer and update the transaction status afterwards. Both states, 'approved' and 'pending_async' can be considered as successful payment and the order can be completed.

Note:

For some service providers it is necessary to 'caputre' the 'approved' or 'pending_async' PurchaseOnAccountTransaction when the goods are delivered. Please contact support to find out if you are required to capture your transactions!!!

Example configuration for returned status:

payment guarantee	inkasso enabled	successful transaction status
yes	no	approved
yes	yes	pending_async
no	no	pending_async
no	yes	pending_async

Request:

parameter	req?	format	description
transaction_type	*required*	"purchase_on_account"	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
wire_reference_id	*optional*	string(10)	Wire reference id for the customer to use with the bankwire
company_name	*conditional*	string(255)	Company name in case of B2B. Contact support for further information.
bank_account_holder	*conditional*	string(255)	name of the bank account holder
bank_account_number	*conditional*	string(16)	customers bank account number
bank_number	*conditional*	string(16)	customers bank number
iban	*conditional*[1]	string(31)	customers international bank account number (SEPA)
bic	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	only DE/AT/CH is valid as country code
shipping_address	*conditional*		Depends on service provider. Contact support for further information.
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	only DE/AT/CH is valid as country code
risk_params	*conditional*	Depends on service provider. Contact support for further information.	
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

PurchaseOnAccount also supports Risk params.

[1] conditionally `bank_account_holder`, `bank_account_number`, `bank_number`, `iban`, `bic` or none of them are required. `iban` and `bic` are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>purchase_on_account</transaction_type>
  <transaction_id>145236</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <company_name>Acme Inc.</company_name>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
  <shipping_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </shipping_address>
  <risk_params>
    <birthday>1978-03-03</birthday>
  </risk_params>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string	Wire reference id for the customer to use with the bankwire
payment_guarantee	boolean	Payment guarantee
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>purchase_on_account</transaction_type>
  <status>approved</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>A72NH093JF</wire_reference_id>
  <payment_guarantee>true</payment_guarantee>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.17 GtdPurchaseOnAccount

GtdPurchaseOnAccount is a synchronous invoice based transaction. The money will not be debited from the customer's bank account. Instead a wire_reference_id is returned, which must be presented to the customer. The customer then needs to wire the money to your merchant bank account by himself referencing the wire_reference_id. The returned status will be returned as 'pending_async' or 'approved' if the transaction was accepted. In case the status is 'pending_async' Hypercharge will track the incoming bank transfer and update the transaction status afterwards. Both states, 'approved' and 'pending_async' can be considered as successful payment and the order can be completed.

Note:

For some service providers it is necessary to 'caputre' the 'approved' or 'pending_async' GtdPurchaseOnAccountTransaction when the goods are delivered. Please contact support to find out if you are required to capture your transactions!!!

Note:

The prefix gtd stands for guaranteed. According to your contract factoring will be applied to your gtd transactions.

Example configuration for returned status:

payment guarantee	inkasso enabled	successful transaction status
yes	no	approved
yes	yes	pending_async
no	no	pending_async
no	yes	pending_async

Request:

parameter	req?	format	description
transaction_type	*required*	"gtd_purchase_on_account"	
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	
remote_ip	*required*	IPv4 address	IPv4 address of customer
wire_reference_id	*optional*	string(10)	Wire reference id for the customer to use with the bankwire
company_name	*conditional*	string(255)	Company name in case of B2B. Contact support for further information.
bank_account_holder	*conditional*	string(255)	name of the bank account holder
bank_account_number	*conditional*	string(16)	customers bank account number
bank_number	*conditional*	string(16)	customers bank number
iban	*conditional*[1]	string(31)	customers international bank account number (SEPA)
bic	*conditional*[1]	string(11)	Business Identifier Codes / Swift ID (SEPA)
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	only DE/AT/CH is valid as country code
shipping_address	*conditional*		Depends on service provider. Contact support for further information.
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	only DE/AT/CH is valid as country code
risk_params	*conditional*	Depends on service provider. Contact support for further information.	
birthday	*conditional*	string(255)	Format: YYYY-MM-DD

GtdPurchaseOnAccount also supports Risk params.

[1] conditionally `bank_account_holder`, `bank_account_number`, `bank_number`, `iban`, `bic` or none of them are required. `iban` and `bic` are required for SEPA transactions. Requirements may differ based on contract and acquiring bank. Contact hypercharge support to find out which requirements apply for you.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>gtd_purchase_on_account</transaction_type>
  <transaction_id>145236</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <wire_reference_id>ABCD1234EF</wire_reference_id>
  <company_name>Acme Inc.</company_name>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <bank_account_number>1290701</bank_account_number>
  <bank_number>20050550</bank_number>
  <iban>DE68210501700012345678</iban>
  <bic>PBNKDEFF</bic>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
  <shipping_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </shipping_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string	Wire reference id for the customer to use with the bankwire
payment_guarantee	boolean	Payment guarantee
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>gtd_purchase_on_account</transaction_type>
  <status>approved</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>A72NH093JF</wire_reference_id>
  <payment_guarantee>true</payment_guarantee>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.18 PayInAdvance

PayInAdvance is a synchronous bankwire based transaction. The money will not be debited from the customer's bank account. Instead a wire_reference_id is returned, which must be presented to the customer. The customer then needs to wire the money to your merchant bank account by himself referencing the wire_reference_id. The returned status will be returned as 'pending_async'. Hypercharge will track the incoming bank transfer and update the transaction status afterwards. Only an 'approved' state can be considered as successful payment and the order can be completed.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"pay_in_advance"	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`wire_reference_id`	*optional*	string	Wire reference id for the customer to use with the bankwire
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*required*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

PayInAdvance also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>pay_in_advance</transaction_type>
  <transaction_id>145236</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
wire_reference_id	string	Wire reference id for the customer to use with the bankwire
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>pay_in_advance</transaction_type>
  <status>approved</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <wire_reference_id>A72NH093JF</wire_reference_id>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.6.19 PaymentOnDelivery

PaymentOnDelivery is a synchronous dummy transaction. No payment will be executed with this transaction. The payment is done on delivery by the customer. Hypercharge can not yet track the delivery or payment status. You will have to update the payment manually by refunding refused deliveries. The response will always be 'approved'. In future hypercharge may support delivery tracking.

Request:

parameter	req?	format	description
`transaction_type`	*required*	"payment_on_delivery"	
`transaction_id`	*required*	string(255)	unique transaction id defined by merchant
`usage`	*optional*	string(255)	
`remote_ip`	*required*	IPv4 address	IPv4 address of customer
`amount`	*required*	integer > 0	Amount of transaction in cents
`currency`	*required*	string(3)	Currency code in ISO 4217
`customer_email`	*required*	e-mail address	Must contain valid e-mail of customer
`customer_phone`	*optional*	string(32)	Must contain valid phone number of customer
`billing_address`	*required*		
`first_name`	*required*	string(255)	Customer first name
`last_name`	*required*	string(255)	Customer last name
`address1`	*required*	string(255)	first line of address
`address2`	*optional*	string(255)	second line of address
`zip_code`	*required*	string(32)	zip code
`city`	*required*	string(255)	city
`state`	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
`country`	*required*	string(2)	country code in ISO 3166

PaymentOnDelivery also supports Risk params.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>payment_on_delivery</transaction_type>
  <transaction_id>145236</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>127.0.0.1</remote_ip>
  <amount>5000</amount>
  <currency>EUR</currency>
  <customer_email>manfred.test@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Manfred</first_name>
    <last_name>Test</last_name>
    <address1>Test Str. 123</address1>
    <zip_code>12345</zip_code>
    <city>Berlin</city>
    <country>DE</country>
  </billing_address>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be ”live” or ”test”
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>payment_on_delivery</transaction_type>
  <status>approved</status>
  <unique_id>011e8d5cc1a56058cc50440c265f5063</unique_id>
  <transaction_id>145236</transaction_id>
  <mode>live</mode>
  <timestamp>2007-08-30T17:46:11Z</timestamp>
  <descriptor>Descriptor One</descriptor>
  <amount>5000</amount>
  <currency>EUR</currency>
</payment_response>
```

2.7 Advanced risk management with RiskParams

The `risk_params` section in the `payment_transaction` xml allows you to pass user specific values along with the payment transaction. These values may be used by advanced risk management features and checked against a blacklist.

RiskParams can be used in any user triggered payment transaction. User triggerd transactions types are Authorize, Authorize3d, Sale, Sale3d, InitRecurringSale, DebitSale, and IdealSale.

parameter	req?	format	description
`risk_params`	*optional*		
`ssn`	*optional*	string(128)	Social Security number or equivalent value for non US customers.
`mac_address`	*optional*	string(128)	the customers mac address.
`session_id`	*optional*	string(128)	the customers session_id.
`user_id`	*optional*	string(128)	the customers user_id.
`user_level`	*optional*	string(128)	a value describing the customers trust level, may be used by the risk management for configurable differentiated limits.
`email`	*optional*	string(128)	the customers email.
`phone`	*optional*	string(128)	the customers phone.
`remote_ip`	*optional*	string(128)	the customers ip address.
`serial_number`	*optional*	string(128)	custom serial number.
`infocapture_token`	*optional*	string(128)	Iovation token.

Note:

The risk management needs to be configured to use these values, passing the values alone will not trigger any risk management features. To use these values for risk management please contact support.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sale</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
  <risk_params>
    <ssn>987-65-4320</ssn>
    <mac_address>12-34-56-78-9A-BC</mac_address>
    <session_id>1DA53551-5C60-498C-9C18-8456BDBA74A9</session_id>
    <user_id>1002547</user_id>
    <user_level>vip</user_level>
    <email>emil@example.com</email>
    <phone>+49301234567</phone>
    <remote_ip>245.253.2.12</remote_ip>
  </risk_params>
</payment_transaction>
```

2.8 Transaction states

Transactions will have one of the following states. These will be returned by transaction responses, reconcile responses and will be shown in the browser interface:

status	description
`approved`	Transaction was approved by acquirer and is successful.
`declined`	Transaction was declined by acquirer or risk management.
`pending_async`	An asynchronous transaction (3-D secure payment) has been initiated and is waiting for user input. Updates of this state will be sent to notification_url specified in request.
`pending`	The outcome of the transaction could not be determined, e.g. at a timeout situation. Transaction state will eventually change, so make a reconcile after a certain time frame.
`error`	An error has occurred while negotiating with the acquirer. As the transaction could have taken place successfully, a reconcile is needed to verify the transaction.
`captured`	Once an approved authorize transaction is captured the state changes to captured.
`refunded`	Once an approved transaction is refunded the state changes to refunded.
`chargebacked`	Once an approved transaction is chargebacked the state changes to chargebacked.
`voided`	Once an approved transaction is voided the state changes to voided.
`chargeback_reversed`	Once a chargebacked transaction is charged the state changes to chargeback_reversed.
`pre_arbitrated`	Once a chargeback_reversed transaction is chargebacked the state changes to pre_arbitrated.
`rejected`	Once an EFT transaction like debit_sale could not be charged from the customers account the state changes to rejected.

3 Recurring

The Hypercharge API has been extended with a scheduled recurring logic. You are able to register transactions for a scheduled payment plan. The credit card or debit account will be charged periodically until you unsubscribe the schedule or a defined end date is exceeded. To register a transaction for recurring you need to add a recurring_schedule node to the payment_transaction XML request as described below.

The following transaction types are supported for scheduled recurring:

- init_recurring_sale
- init_recurring_authorize
- init_recurring_debit_sale
- init_recurring_debit_authorize

The following transaction types will be used as recurring transactions:

- recurring_sale
- recurring_debit_sale

3.1 Register recurring schedule

Request:

parameter	req?	format	description
transaction_type	*required*	"debit_sale"	init_recurring_sale or debit_sale
transaction_id	*required*	string(255)	unique transaction id defined by merchant
usage	*optional*	string(255)	description of the transaction for later use
remote_ip	*required*	IPv4 address	IPv4 address of customer
amount	*required*	integer > 0	Amount of transaction in cents
currency	*required*	string(3)	Currency code in ISO 4217
card_holder	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
card_number	*required*	string(13..21)	credit card number
cvv	*required*	3 to 4 digits	cvv of cc, requirement is based on channel configuration
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card
customer_email	*required*	e-mail address	Must contain valid e-mail of customer
customer_phone	*optional*	string(32)	Must contain valid phone number of customer
billing_address	*required*		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
zip_code	*required*	string(32)	zip code
city	*required*	string(255)	city
state	*optional*	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
recurring_schedule	*optional*		
start_date	*required*	yyyy-mm-dd	Start date to begin with recurring
end_date	*optional**	yyyy-mm-dd	End date to stop the recurring schedule
amount	*required*	integer > 0	Amount to rebill in cents
interval	*required*	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
max_retries	*optional*	integer	number of times to retry the rebilling before giving up

optional* = If omitted, end_date will be set to the credit card expiry date if a credit card transaction is registered for recurring. Also if end_date exceeds credit card expiry, end_date will be set to expiry date.

Example XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_transaction>
  <transaction_type>sale</transaction_type>
  <transaction_id>43671</transaction_id>
  <usage>40208 concert tickets</usage>
  <remote_ip>245.253.2.12</remote_ip>
  <amount>5000</amount>
  <currency>USD</currency>
  <card_holder>Emil Example</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_month>2</expiration_month>
  <expiration_year>2010</expiration_year>
  <customer_email>emil@example.com</customer_email>
  <customer_phone>+49301234567</customer_phone>
  <billing_address>
    <first_name>Max</first_name>
    <last_name>Mustermann</last_name>
    <address1>Muster Str. 12</address1>
    <zip_code>10178</zip_code>
    <city>Berlin</city>
    <state>DE1</state>
    <country>DE</country>
  </billing_address>
  <recurring_schedule>
        <start_date>2012-08-15</start_date>
        <end_date>2013-04-15</end_date>
        <interval>monthly</interval>
        <amount>1000</amount>
        <max_retries>1</max_retries>
  </recurring_schedule>
</payment_transaction>
```

Successful response:

name	type	description
transaction_type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be "live" or "test"
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217
recurring_schedule	optional	
unique_id	string(32)	unique id referencing the registered schedule. See below for further use on the schedule
start_date	yyyy-mm-dd	Start date to begin with recurring
end_date	yyyy-mm-dd	End date to stop the recurring schedule
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semi-monthly, quarterly, semianually and anually or an integer between 1 and 365
active	boolean	activation status of the recurring schedule
enabled	boolean	subscription status of the recurring schedule.

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <transaction_type>sale</transaction_type>
  <status>approved</status>
  <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
  <recurring_schedule>
        <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
        <start_date>2012-08-15</start_date>
        <end_date>2013-04-15</end_date>
        <interval>monthly</interval>
        <amount>1000</amount>
        <active>true</active>
        <enabled>true</enabled>
  </recurring_schedule>
</payment_response>
```

3.2 Recurring API interface

The Recurring API Interface provides several functions to gather information on the recurring schedules and payment events. It is also possible to temporary suspend schedules and reactivate them or deactivate a schedule permanently.

3.2.1 Recurring events by schedule

Lists all payment events processed for a recurring schedule.

The URL to query recurring events for a recurring schedule is:

```
https://hypercharge.net/recurring/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
unique_id	*required*	sring(32)	unique_id of the recurring schedule to query
page	*optional*	intger	the page within the paginated result, defaults to 1

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <page>2</page>
</recurring>
```

Response:
The attributes in the node `recurring_events` includes information about the pagination of the response.

name	type	description
unique_id	string(32)	unique_id of the recurring schedule queried
start_date	yyyy-mm-dd	date of the first recurring event
end_date	yyyy-mm-dd	date of the final recurring event
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
active	boolean	activation status of the recurring schedule
enabled	boolean	subscription status of the recurring schedule
recurring_events		list of recurring events the queried schedule
@per_page	integer	number of entries per page
@page	integer	the current page
@total_count	integer	total number of all entries
@pages_count	integer	total number of pages
recurring_event		recurring event
unique_id	string(32)	unique_id of the recurring event
recurring_schedule_unique_id	string(32)	unique_id of the recurring schedule
status	string	status of the event, see **states**
due_date	yyyy-mm-dd	date the recurring schedule is due
finalized_at	yyyy-mm-dd	date the recurring event was processed successfuly

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_schedule>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <start_date>2012-04-15</start_date>
  <end_date>2013-03-15</end_date>
  <amount>1000</amount>
  <interval>monthly</interval>
  <active>true</active>
  <enabled>true</enabled>
  <recurring_events per_page="100" pages_count="3" page="1" total_count = "250">
    <recurring_event>
      <unique_id>19aa2dcfb5b4e1bec6a71a58cb3fbdc5</unique_id>
      <recurring_schedule_unique_id>44177a21403427eb96664a6d7e5d5d48</recurring_schedule_unique_id>
      <status>approved</status>
      <due_date>2012-04-15</due_date>
      <finalized_at>2012-04-15</finalized_at>
    </recurring_event>
    <recurring_event>
      <unique_id>0594f69e203086ced3d30bc39a298504</unique_id>
      <recurring_schedule_unique_id>44177a21403427eb96664a6d7e5d5d48</recurring_schedule_unique_id>
      <status>approved</status>
      <due_date>2012-05-15</due_date>
      <finalized_at>2012-05-16</finalized_at>
    </recurring_event>
```

```
    <recurring_event>
      <unique_id>49b8b6cc059d8e9cee163e0dffa9a1fa</unique_id>
      <recurring_schedule_unique_id>44177a21403427eb96664a6d7e5d5d48</recurring_schedule_unique_id>
      <status>approved</status>
      <due_date>2012-06-15</due_date>
      <finalized_at>2012-06-15</finalized_at>
    </recurring_event>
  </recurring_events>
</recurring_schedule>
```

3.2.2 Recurring event details

Shows payment details for a payment event.

The URL to query recurring event is:

```
https://hypercharge.net/recurring/event/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
unique_id	*required*	sring(32)	unique_id of the recurring event to query

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
</recurring>
```

Response:

name	type	description
unique_id	string(32)	unique_id of the recurring event
recurring_schedule_unique_id	string(32)	unique_id of the recurring schedule
status	string	status of the event, see **states**
due_date	yyyy-mm-dd	date the recurring schedule is due
finalized_at	yyyy-mm-dd	date the recurring event was processed successfuly
payment_response		payment response
type	string	the transaction type
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if capturing, voiding or refunding a transaction)
transaction_id	string(255)	unique transaction id defined by merchant
mode	string	Mode of the transaction, can be "live" or "test"
timestamp	string	time when the transaction was processed in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
descriptor	string	Channel information as configured in gate
amount	integer	Amount of transaction in cents
currency	string	Currency code in ISO 4217

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_event>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <recurring_schedule_unique_id>5e2cbbad71d2b13432323153c208223a</recurring_schedule_unique_id>
  <status>approved</status>
  <due_date>2012-04-15</due_date>
  <finalized_at>2012-04-15</finalized_at>
  <payment_response>
    <transaction_type>recurring_sale</transaction_type>
    <status>approved</status>
    <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
    <transaction_id>3872422-001-1</transaction_id>
    <technical_message>TESTMODE: No real money will be transferred!</technical_message>
    <message>TESTMODE: No real money will be transferred!</message>
    <mode>test</mode>
    <timestamp>2012-04-15T02:00:12Z</timestamp>
    <descriptor>descriptor one</descriptor>
    <amount>1000</amount>
    <currency>EUR</currency>
  </payment_response>
</recurring_event>
```

3.2.3 Unsubscribe recurring schedule

Permanently deactivates a recurring schedule.

The URL to deactivate a recurring schedule is:

```
https://hypercharge.net/recurring/unsubscribe/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
unique_id	*required*	sring(32)	unique_id of the recurring schedule to query

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
</recurring>
```

Response:
The attributes in the node `recurring_events` includes information about the pagination of the response.

name	type	description
unique_id	string(32)	unique_id of the recurring schedule queried
start_date	yyyy-mm-dd	date of the first recurring event
end_date	yyyy-mm-dd	date of the final recurring event
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
active	boolean	activation status of the recurring schedule
enabled	boolean	subscription status of the recurring schedule

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_schedule>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <start_date>2012-04-15</start_date>
  <end_date>2013-03-15</end_date>
  <amount>1000</amount>
  <interval>monthly</interval>
  <active>true</active>
  <enabled>false</enabled>
</recurring_schedule>
```

3.2.4 Activate recurring schedule

Reactivate a suspended recurring schedule.

The URL to reactivate a suspended recurring schedule is:

```
https://hypercharge.net/recurring/activate/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
unique_id	*required*	sring(32)	unique_id of the recurring schedule to query

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
</recurring>
```

Response:
The attributes in the node `recurring_events` includes information about the pagination of the response.

name	type	description
unique_id	string(32)	unique_id of the recurring schedule queried
start_date	yyyy-mm-dd	date of the first recurring event
end_date	yyyy-mm-dd	date of the final recurring event
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
active	boolean	activation status of the recurring schedule
enabled	boolean	subscription status of the recurring schedule

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_schedule>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <start_date>2012-04-15</start_date>
  <end_date>2013-03-15</end_date>
  <amount>1000</amount>
  <interval>monthly</interval>
  <active>true</active>
  <enabled>true</enabled>
</recurring_schedule>
```

3.2.5 Deactivate recurring schedule

Temporary deactivate a recurring schedule.

The URL to deactivate a recurring schedule is:

```
https://hypercharge.net/recurring/deactivate/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
unique_id	*required*	sring(32)	unique_id of the recurring schedule to query

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
</recurring>
```

Response:
The attributes in the node `recurring_events` includes information about the pagination of the response.

name	type	description
unique_id	string(32)	unique_id of the recurring schedule queried
start_date	yyyy-mm-dd	date of the first recurring event
end_date	yyyy-mm-dd	date of the final recurring event
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
active	boolean	activation status of the recurring schedule
enabled	boolean	subscription status of the recurring schedule

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_schedule>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <start_date>2012-04-15</start_date>
  <end_date>2013-03-15</end_date>
  <amount>1000</amount>
  <interval>monthly</interval>
  <active>false</active>
  <enabled>true</enabled>
</recurring_schedule>
```

3.2.6 Update recurring schedule

Update a recurring schedule.

The URL to update a recurring schedule is:

```
https://hypercharge.net/recurring/update/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
unique_id	*required*	sring(32)	unique_id of the recurring schedule to query

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <start_date>2012-04-15</start_date>
  <end_date>2013-03-15</end_date>
  <amount>1000</amount>
  <interval>monthly</interval>
  <max_retries>3</max_retries>
</recurring>
```

Response:
The attributes in the node `recurring_events` includes information about the pagination of the response.

name	type	description
unique_id	string(32)	unique_id of the recurring schedule queried
start_date	yyyy-mm-dd	date of the first recurring event
end_date	yyyy-mm-dd	date of the final recurring event
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
amount	integer > 0	Amount to rebill in cents

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_schedule>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <start_date>2012-04-15</start_date>
  <end_date>2013-03-15</end_date>
  <amount>1000</amount>
  <interval>monthly</interval>
  <active>false</active>
  <enabled>true</enabled>
</recurring_schedule>
```

3.2.7 Reucrring schedules by date

Returns a paginated list of recurring schedules selected by start_date.

The URL to query recurring schedules by start_date is:

```
https://hypercharge.net/recurring/schedules_by_date/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
start_date	*required*	yyyy-mm-dd	start of the requested date range
end_date	*optional*	yyyy-mm-dd	end of the requested date range
page	*optional*	intger	the page within the paginated result, defaults to 1

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <start_date>2012-04-01</start_date>
  <end_date>2012-07-31</end_date>
  <page>2</page>
</recurring>
```

Response:
The attributes in the node `recurring_schedules` includes information about the pagination of the response.

name	type	description
@per_page	integer	number of entries per page
@page	integer	the current page
@total_count	integer	total number of all entries
@pages_count	integer	total number of pages
unique_id	string(32)	unique_id of the recurring schedule queried
start_date	yyyy-mm-dd	date of the first recurring event
end_date	yyyy-mm-dd	date of the final recurring event
amount	integer > 0	Amount to rebill in cents
interval	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
active	boolean	activation status of the recurring schedule
enabled	boolean	subscription status of the recurring schedule

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_schedules per_page="100" pages_count="3" page="1" total_count = "250">
       <recurring_schedule>
         <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
         <start_date>2012-05-15</start_date>
         <end_date>2013-04-15</end_date>
         <amount>1000</amount>
         <interval>monthly</interval>
         <active>true</active>
         <enabled>true</enabled>
       </recurring_schedule>
       <recurring_schedule>
         <unique_id>5e2cbbad71d2b13432323153c208223a</unique_id>
         <start_date>2012-06-15</start_date>
         <end_date>2013-05-15</end_date>
         <amount>500</amount>
         <interval>quarterly</interval>
         <active>true</active>
         <enabled>true</enabled>
       </recurring_schedule>
       <recurring_schedule>
         <unique_id>3641b4fbe80051319a60ceeacf15c8c7</unique_id>
         <start_date>2012-07-15</start_date>
         <end_date>2013-06-15</end_date>
         <amount>1500</amount>
         <interval>monthly</interval>
         <active>true</active>
         <enabled>true</enabled>
       </recurring_schedule>
</recurring_schedules>
```

3.2.8 Reucrring events by date

Returns a paginated list of recurring events selected by creation date.

The URL to query recurring events by start_date is:

```
https://hypercharge.net/recurring/events_by_date/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
start_date	*required*	yyyy-mm-dd	start of the requested date range
end_date	*optional*	yyyy-mm-dd	end of the requested date range
page	*optional*	intger	the page within the paginated result, defaults to 1

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring>
  <start_date>2012-04-01</start_date>
  <end_date>2012-07-31</end_date>
  <page>2</page>
</recurring>
```

Response:
The attributes in the node `recurring_events` includes information about the pagination of the response.

name	type	description
@per_page	integer	number of entries per page
@page	integer	the current page
@total_count	integer	total number of all entries
@pages_count	integer	total number of pages
recurring_event		recurring event
unique_id	string(32)	unique_id of the recurring event
recurring_schedule_unique_id	string(32)	unique_id of the recurring schedule
status	string	status of the event, see **states**
due_date	yyyy-mm-dd	date the recurring schedule is due
finalized_at	yyyy-mm-dd	date the recurring event was processed successfuly

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<recurring_events per_page="100" pages_count="3" page="1" total_count = "250">
  <recurring_event>
    <unique_id>19aa2dcfb5b4e1bec6a71a58cb3fbdc5</unique_id>
    <recurring_schedule_unique_id>44177a21403427eb96664a6d7e5d5d48</recurring_schedule_unique_id>
    <status>approved</status>
    <due_date>2012-04-15</due_date>
    <finalized_at>2012-04-15</finalized_at>
  </recurring_event>
  <recurring_event>
    <unique_id>0594f69e203086ced3d30bc39a298504</unique_id>
    <recurring_schedule_unique_id>5e2cbbad71d2b13432323153c208223a</recurring_schedule_unique_id>
    <status>approved</status>
    <due_date>2012-05-15</due_date>
    <finalized_at>2012-05-16</finalized_at>
  </recurring_event>
  <recurring_event>
    <unique_id>49b8b6cc059d8e9cee163e0dffa9a1fa</unique_id>
    <recurring_schedule_unique_id>3641b4fbe80051319a60ceeacf15c8c7</recurring_schedule_unique_id>
    <status>approved</status>
    <due_date>2012-06-15</due_date>
    <finalized_at>2012-06-15</finalized_at>
  </recurring_event>
</recurring_events>
```

3.3 Recurring and WPF

Registering a recurring schedule for a WPF payment works analog to the above described by adding the optional scheduling parameters.

Request:

parameter	req?	format	description
recurring_schedule	*optional*		
start_date	*required*	yyyy-mm-dd	Start date to begin with recurring
end_date	*optional**	yyyy-mm-dd	End date to stop the recurring schedule
amount	*required*	integer > 0	Amount to rebill in cents
interval	*required*	string(255)	recurring interval. must be one of weekly, monthly, semimonthly, quarterly, semianually and anually or an integer between 1 and 365
max_retries	*optional*	integer	number of times to retry the rebilling before giving up

3.4 Recurring event notification

Recurring event notifications are sent once the recurring payment has been successfuly rebilled or failed and are transmitted via HTTP POST (application/x-www-form-urlencoded) with the following parameters:

An example notification:

```
recurring_schedule_unique_id=26aa150ee68b1b2d6758a0e6c44fce4c
&recurring_event_unique_id=bad08183a9ec545daf0f24c48361aa10
&recurring_event_due_date=2013-01-21 00:00:00
&notification_type=recurring
&signature=c5219b3d385e74496b2b48a5497b347e102849f10eacd25b062f823b
&payment_transaction_channel_token=e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa
&payment_transaction_unique_id=bad08183a9ec545daf0f24c48361aa10
&payment_transaction_transaction_type=recurring_sale
&payment_transaction_status=approved
```

Parameters:

name	type	description
recurring_schedule_unique_id	string	unique_id of the recurring schedule the payment was processed for
recurring_event_unique_id	string	unique_id of the recurring event
recurring_event_due_date	date	date the recurring payment should be processed by
notification_type	string	parameter cotains 'recurring'
signature	string	the signature of the notification, should be used to verify the the notifcation was send by hypercharge
payment_transaction_channel_token	string	the channel_token as used in the processing url
payment_transaction_unique_id	string	unique_id generated by hypercharge
payment_transaction_transaction_type	string	transaction_type for the transaction eg: recurring_sale
payment_transaction_status	string	status of the payment transaction

Status will be one of the following: approved, declined, error, timeout, pending, pending_async, canceled, refunded, chargebacked, chargeback_reversed, pre_arbitrated, captured, and voided.

The signature is a mean of security to ensure that the gate is really the sender of the notification. It is generated by concatenating the unique_id of the recurring event with your API password and generating a SHA-512 Hash (Hex) of the string:

```
SHA-512 Hash (Hex) of <recurring_event_unique_id><Your Merchant API password>
```

3.5 Expiring recurring schedule notification

Expiring recurring schedule notifications are sent when a schedule is about to expire and are transmitted via HTTP POST (application/x-www-form-urlencoded) with the following parameters:

An example notification:

```
recurring_schedule_unique_id=26aa150ee68b1b2d6758a0e6c44fce4c
&recurring_schedule_end_date=2013-01-21 00:00:00
&recurring_schedule_status=expiring
&notification_type=recurring_schedule
&signature=c5219b3d385e74496b2b48a5497b347e102849f10eacd25b062f823b
```

Parameters:

name	type	description
recurring_schedule_unique_id	string	reference_id of the recurring schedule expiring
recurring_schedule_end_date	date	date the recurring schedule expires
recurring_schedule_status	string	'expiring' status of the recurring schedule
notification_type	string	parameter cotains 'recurring_schedule'
signature	string	the signature of the notification, should be used to verify the the notifcation was send by hypercharge

Status will be 'expiring'.

The signature is a mean of security to ensure that the gate is really the sender of the notification. It is generated by concatenating the unique_id of the recurring schedule with your API password and generating a SHA-512 Hash (Hex) of the string:

```
SHA-512 Hash (Hex) of <recurring_schedule_unique_id><Your Merchant API password>
```

3.5.1 Notification signature examples (Recurring)

unique_id	API password	signature
26aa150ee68b1b2d 6758a0e6c44fce4c	b5af4c9cf497662e00b7 8550fd87e65eb415f42f	c5219b3d385e74496b2b48a5497b347e102849f1 0eacd25b062f823bbd11249ce4e233f031c0eceb c9b691e69d23eb0c1cd65a79621152467b56ac2b f103b512
3f760162ef57a829 011e5e2379b3fa17	50fd87e65eb415f42fb5 af4c9cf497662e00b785	14519d0db2f7f8f407efccc9b099c5303f55c026 2e3b9132e5bcc97f7febf5f9ab19df03929c1ead 271be79807b4086321a023743d2b6b1278c2082b 61cf3ff0

Note:

You **must** either use the signature to verify the notification's integrity or make a reconcile to check the final transaction status.

When receiving the notification, you are required to render an xml page containing the recurring schedule's recurring_schedule_unique_id so that the gateway knows that you have accepted the notification. If the XML is not delivered, the notification is sent periodically until the XML is received.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification_echo>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</notification_echo>
```

4 Reconcile

Reconcile can be used to retrieve data about a transaction. This can be useful if you want to retrieve information about a transaction whose status is timeout, which returned an error or has changed eg. has beed chargebacked.

Reconcile request are handled exactly like transaction requests via XML.

4.1 Single Transaction

The URL for single transaction reconciling is similar to the processing url:

```
https://skprocessing.hypercharge.net/reconcile/CHANNEL_TOKEN/
```

XML Request to reconcile:

```
<?xml version="1.0" encoding="UTF-8"?>
<reconcile>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
</reconcile>
```

XML Response:

Response is a standard payment_response like it would be returned by any transaction. It can have either state as shown in the **states** section.

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_response>
  <status>approved</status>
  <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  <transaction_id>119643250547501c79d8295</transaction_id>
  <mode>live</mode>
  <timestamp>2007-11-30T14:21:48Z</timestamp>
  <descriptor>descriptor one</descriptor>
  <amount>9000</amount>
  <currency>USD</currency>
</payment_response>
```

4.2 By date range

Date range based reconciliation allows you to fetch information for all payment transactions from a channel within a given date range. The response is paginated, each request will return 100 entries max.

The URL for date range reconciling is:

```
https://skprocessing.hypercharge.net/reconcile/by_date/CHANNEL_TOKEN/
```

Request:

parameter	req?	format	description
start_date	*required*	yyyy-mm-dd	start of the requested date range
end_date	*optional*	yyyy-mm-dd	end of the requested date range
page	*optional*	intger	the page within the paginated result, defaults to 1

XML Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<reconcile>
  <start_date>2009-01-01</start_date>
  <end_date>2009-01-31</end_date>
  <page>2</page>
</reconcile>
```

Response:
The attributes in the root node `payment_responses` includes information about the pagination of the response.

name	type	description
`@per_page`	`integer`	number of entries per page
`@page`	`integer`	the current page
`@total_count`	`integer`	total number of all entries
`@pages_count`	`integer`	total number of pages

XML Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment_responses per_page="100" page="2" total_count="19" pages_count="7">
  <payment_response>
    <transaction_type>sale</transaction_type>
    <status>approved</status>
    <unique_id>130319cfb3bf65ff3c4a4045487b174f</unique_id>
    <transaction_id>EFBFBC7D-82CD-4375-9A69-15F19C88A134</transaction_id>
    <technical_message>Transaction successful!</technical_message>
    <message>Transaction successful!</message>
    <mode>live</mode>
    <timestamp>2009-11-05T15:04:00Z</timestamp>
    <descriptor>descriptor one</descriptor>
    <amount>500</amount>
    <currency>USD</currency>
  </payment_response>
  <payment_response>
    <transaction_type>sale</transaction_type>
    <status>approved</status>
    <unique_id>130319cfb3bf65ff3c4a4045487b173f</unique_id>
    <transaction_id>BBD7945B-BE57-4A14-A7FB-47F7AE928D95</transaction_id>
    <technical_message>Transaction successful!</technical_message>
    <message>Transaction successful!</message>
    <mode>live</mode>
    <timestamp>2009-11-05T15:04:00Z</timestamp>
    <descriptor>descriptor one</descriptor>
    <amount>500</amount>
    <currency>USD</currency>
  </payment_response>
  ...
</payment_responses>
```

5 Asynchronous transactions and Notifications

5.1 Asynchronous transactions

3-D secure transactions and IdealSale transaction are always processed **asynchronously**. Some other transactions are processed **asynchronously** depending on the acquirer.
This means that the final result of the transaction will not be available immediately and the status is `pending_async`. Once the transaction has reached a final status, a Notification is sent to the merchant.

Note:

Whenever the status is `pending_async` the transaction gets processed **asynchronously**, and a Notification is sent to the merchant once the transaction has reached a final status.

Overview:

Transaction type	async?
AuthorizeTransaction	never
Authorize3dTransaction	always
SaleTransaction	depends on acquirer
Sale3dTransaction	always
CaptureTransaction	depends on acquirer
RefundTransaction	depends on acquirer
VoidTransaction	depends on acquirer
InitRecurringSale	never
RecurringSale	never
IdealSale	always
ReferencedFundTransfer	depends on acquirer
PayPal	always

5.2 Notifications

For asynchronous payments a notification is always send to the `notification_url` within the payment transaction.

The payment gateway can be configured to also send a notification after each synchronous payment transaction. The notification is send to the `notification_url` which is configured per merchant.

Also see 3-D Secure Transactions and Notification for asynchronous payments.

The format of the notification in both cases is the same.

```
transaction_id=82803B4C-70CC-43BD-8B21-FD0395285B40&unique_id=44177a21403427eb96664a6d7e5d5d48&
transaction_type=sale3d&channel_token=394f2ebc3646d3c017fa1e1cbc4a1e20&
status=approved&signature=088e16a1019277b15d58faf0541e11910eb756f6
```

Parameters:

name	type	description
`transaction_id`	`string`	merchant generated tranaction_id
`unique_id`	`string`	unique_id generated by hypercharge
`transaction_type`	`string`	transaction_type for the transaction eg: sale3d
`channel_token`	`string`	the channel_token as used in the processing url
`status`	`string`	status of the payment transaction
`signature`	`string`	the signature of the notification, should be used to verify the the notifcation was send by hypercharge

Status will be either "declined", "approved" or "error", like shown in the **states** table.

The signature is a mean of security to ensure that the gate is really the sender of the notification. It is generated by concatenating the unique_id of the transaction with your API password and generating a SHA1 Hash (Hex) of the string:

```
SHA1 Hash (Hex) of <unique_id><Your API password>
```

Notification signature examples

unique_id	API password	signature
`fc6c3c8c0219730c7a099eaa540f70dc`	`bogus`	08d01ae1ebdc22b6a1a764257819bb26e9e94e8d
`130319cfb3bf65ff3c4a4045487b173e`	`test123`	1b34dabed996788efcc049567809484454ee8b17

You can use the signature to verify the integrity of the notification, ensuring that it was really sent by the gate.

Note:

You **must** either use the signature to verify the notification's integrity or make a reconcile to check the final transaction status.

When receiving the notification, you are required to render an xml page containing the transaction's unique_id so that the gateway knows that you have accepted the notification. If the XML is not delivered, the notification is sent periodically until the XML is received.

```
<?xml version="1.0" encoding="UTF-8"?>
  <notification_echo>
    <unique_id>44177a21403427eb96664a6d7e5d5d48</unique_id>
  </notification_echo>
```

6 Chargebacks

Chargebacks are a special type of transactions as they cannot be triggered by the merchant. Chargebacks occur if a customer disputes to an item of his credit card bill at his issuing bank and the bank requests a chargeback. In this case, the amount is automatically refunded to the customers cc account and deducted from your merchant account.

Customers who initiate chargebacks will automatically be blocked for future transactions by the risk management.

You can see also see a chargeback overview in the admin interface.

6.1 Chargeback notifications

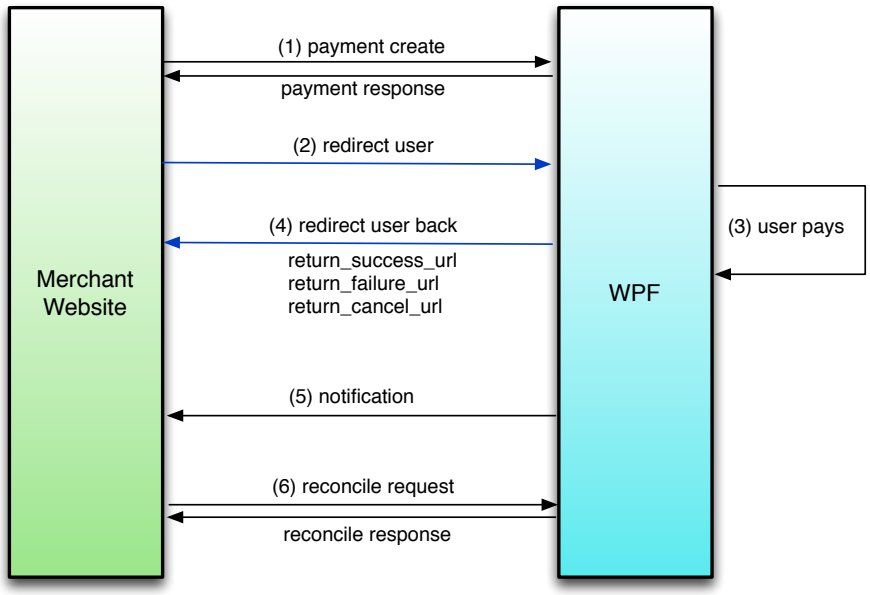
You now have the option to receive a notification for each chargeback that occurs. Enable this feature by emailing the Hypercharge support team with the chargeback notification URL.

These notifications are equal to Notifcation for asynchronous payments, please refer to the section Notifcation for asynchronous payments to understand how notifications work.

7 WPF

The WPF (Web Payment Form) is a customizable component of the hypercharge payment gateway. It provides merchants with an intuitive user interface to easily process their customers' payments. Through a single point of integration, the merchant can offer his customers multiple payment methods instantly and since the WPF is hosted on the secure hypercharge payment system server, it is already PCI-conform.

7.1 Workflow



In the example above the customer visits the merchant's website and does a checkout. Subsequently **(1)**, the merchant initiates the payment on the hypercharge payment gateway through a request to the WPF API, which carries the mandatory, inital payload (e.g. amount, currency, etc.). The response to this request (if successful) is a redirect URL, which the merchants hands over to the customer. Following this redirect URL **(2)**, the customer is then directed to the actual payment form (WPF), which gets served from the hypercharge servers. Because the merchant has previously transmitted most of the relevant payment information, the form is pre-filled with these values and the customer only needs to add personal data. The customer then selects one of the payment methods offered by the merchant and fills in his payment information **(3)** (e.g. credit card data). Upon completion the customer is redirected back to the merchant **(4)**[1]. After the payment has been processed and reached

[1] Particularly to the `return_success_url` defined by the merchant in his initial request. If the customer has selected an asynchronous payment method, he is redirected to the acquirer before this step

a final state the merchant is sent a notification **(5)** to the `notification_url` supplied in the initial `create` request (1). The merchant **must** either uses the notifcation's signature to verify the payment's integrity or make a reconcile **(6)** to check the final payment status. However, we urge all merchants to always do a reconcile.

Note:

As with all other asynchronous payment transactions, the return-, success- and cancel-URLs are **only** meant to display a useful page/message to the customer. A redirect of the customer to one of these URLs **never** gives any form of indication of the payment's state. To find out whether the payment has gone through or not the merchant must **always** wait for the notification or (even better) do a reconcile.

Please also note that there is no specific order in which notification and redirect will occur (that means that the notification may also arrive before the customer's redirect).

7.2 WPF API

7.2.1 URLs:

Create: The URL for the WPF API `create` method is:

```
https://payment.hypercharge.net/payment
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment
```

Reconcile: The URL for the WPF API `reconcile` method is:

```
https://payment.hypercharge.net/payment/reconcile
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment/reconcile
```

Cancel: The URL for the WPF API `cancel` method is:

```
https://payment.hypercharge.net/payment/cancel
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment/cancel
```

7.2.2 Create

Parameters:

parameter	req?	format	description
type	*required*	string	constant value "WpfPayment"
transaction_id	*required*	string(255)	unique transaction id defined by merchant
amount	*required*	integer > 0	Amount of the payment in cents
currency	*required*	string(3)	Currency code in ISO 4217
wire_reference_id	*optional*	string(10)	Wire reference id for the customer to use with the bankwire
sepa_mandate_id	*conditional*[1]	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*conditional*[1]	date	mandate signature date; Format: YYYY-MM-DD
company_name	*conditional*[2]	string(255)	Company name in case of B2B. Contact support for further information.
usage	*required*	string	statement, as it appears in the customer's bank statement
description	*required*	string	a text describing the reason of the payment (e.g. "you're buying concert tickets")
editable_by_user	*optional*	boolean	if set to false the payment form will skip step 1. However a billing_address musst be provided unless optional billing_address is configured
customer_email	*conditional*[3]	string	the customer's e-mail address
customer_phone	*optional*	string	the customer's phone number
notification_url	*required*	string	URL at merchant's site where gateway sends outcome of transaction after the payment has reached a final state. This should be an SSL secured page
return_success_url	*required*	string	URL where customer is sent to after successful payment
return_failure_url	*required*	string	URL where customer is sent to after unsuccessful payment
return_cancel_url	*required*	string	URL where customer is sent to when the customer cancels the payment process within the WPF
billing_address	*conditional*[3]		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
city	*required*	string(255)	City
zip_code	*required*	string(32)	Zip Code

state	*conditional*[4]	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
transaction_types	*optional*		the transaction types that the merchant is willing to accept payments for
transaction_type	*required*	string	one of the available transaction types (e.g. sale, sale3d, etc.)
risk_params	*optional*		list of risk params as described in the Advanced risk management with RiskParams section
user_id (example)	*required*	string	the customer's ID within the merchant's system (example)
retries	*optional*	integer > 0	The acceptable payment retries for the customers if a transaction failed.
expires_in	*optional*	integer > 300, integer < 86400	The acceptable "time-to-live" in seconds between the merchant's creation of a payment and the customer's interaction. This can be set to anything between 5 minutes and 24 hours

Example Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <transaction_id>wev238f328nc</transaction_id>
  <usage>Order ID 500, Shoes</usage>
  <description>You are about to buy 3 shoes at www.shoes.com!</description>
  <editable_by_user>true</editable_by_user>
  <notification_url>https://example.com/notification</notification_url>
  <return_success_url>https://example.com/return_success</return_success_url>
  <return_failure_url>https://example.com/return_failure</return_failure_url>
  <return_cancel_url>https://example.com/return_cancel</return_cancel_url>
  <amount>5000</amount>
  <currency>USD</currency>
  <sepa_mandate_id>1234567890abcdef</sepa_mandate_id>
  <sepa_mandate_signature_date>2014-06-24</sepa_mandate_signature_date>
  <company_name>Acme Inc.</company_name>
  <customer_email>john.doe@example.com</customer_email>
  <customer_phone>+11234567890</customer_phone>
  <billing_address>
    <first_name>John</first_name>
    <last_name>Doe</last_name>
    <address1>23, Doestreet</address1>
    <zip_code>11923</zip_code>
```

[1] required for SEPA.
[2] required for PurchaseOnAccount to initiate B2B transactions.
[3] is only required, if `editable_by_user is 0`.
[4] is required, if `country is either USA or Canada`.

```
    <city>New York City</city>
    <state>NY</state>
    <country>US</country>
  </billing_address>
  <transaction_types>
    <transaction_type>sale</transaction_type>
    <transaction_type>sale3d</transaction_type>
  </transaction_types>
  <risk_params>
    <user_id>123456</user_id>
  </risk_params>
  <retries>3</retries>
  <expires_in>300</expires_in>
</payment>
```

Successful response:

name	type	description
type	string	constant value "WpfPayment"
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if reconciling a payment)
transaction_id	string(255)	unique transaction id defined by merchant
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the payment, can be "live" or "test"
timestamp	string	time when the payment was created in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
amount	integer > 0	Amount of transaction in cents
currency	string	Currency code in ISO 4217
redirect_url	url	URL where user has to be redirected to complete payment process
payment_methods		list of available payment methods
payment_method	string	e.g. "credit_card", "direct_debit", etc.

Example Successful Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>new</status>
  <unique_id>eabcb7a41044e764746b0c7e32c1e9d1</unique_id>
  <transaction_id>wev238f328nc</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-19T15:00:11Z</timestamp>
  <amount>5000</amount>
```

```
  <currency>USD</currency>
  <redirect_url>https://testpayment.hypercharge.net/pay/step1/eabcb7a41044e764746b0c7e32
c1e9d1</redirect_url>
  <payment_methods>
    <payment_method>credit_card</payment_method>
  </payment_methods>
</payment>
```

Error response:

name	type	description
status	string	status of the transaction, see **states**
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>Unknown system error. Please contact support.</technical_message>
  <message>Transaction failed, please contact support!</message>
</payment>
```

7.2.3 Notification

WPF Notifications are sent once the WPF payment has reached a final state and are transmitted via HTTP POST (application/x-www-form-urlencoded) with the following parameters:

Note:

Apart from the workflow described above, the hypercharge system will also use the `notification_url` endpoint to send notifications to the merchant if a WPF payment is chargebacked.

An example notification:

```
payment_transaction_id=mtid20110408144716113553696 2
&payment_status=approved
&payment_unique_id=26aa150ee68b1b2d6758a0e6c44fce4c
&notification_type=WpfPayment
&signature=c5219b3d385e74496b2b48a5497b347e102849f10eacd25b062f823b
```

In case a payment transaction was processed these additional parameters are returned:

```
&payment_transaction_channel_token=e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa
&payment_transaction_unique_id=bad08183a9ec545daf0f24c48361aa10
&payment_transaction_transaction_type=sale
```

In case a recurring schedule was created for the payment these additional parameters are returned:

```
&recurring_schedule_unique_id=bad08183a9ec545daf0f24c48361aa10
&recurring_schedule_end_date=2013-01-21 00:00:00
```

Parameters:

name	type	description
payment_transaction_id	string	merchant generated tranaction_id
payment_status	string	status of the payment transaction
payment_unique_id	string	unique_id generated by hypercharge, required for WPF payment reconciliation
payment_transaction_transaction_type	string	transaction_type for the transaction eg: sale3d
notification_type	string	constant value "WpfPayment"
signature	string	the signature of the notification, should be used to verify the the notifcation was send by hypercharge
payment_transaction_channel_token	string	the channel_token as used in the processing url
payment_transaction_unique_id	string	unique_id generated by hypercharge
recurring_schedule_unique_id	string	reference_id of the recurring schedule the payment was processed for
recurring_schedule_end_date	date	date of the final recurring event

Status will be one of the following: approved, declined, error, timeout, pending, pending_async, canceled, refunded, chargebacked, chargeback_reversed, pre_arbitrated, captured, and voided.

The signature is a mean of security to ensure that the gate is really the sender of the notification. It is generated by concatenating the unique_id of the payment with your API password and generating a SHA-512 Hash (Hex) of the string:

```
SHA-512 Hash (Hex) of <payment_unique_id><Your Merchant API password>
```

Notification signature examples (WPF)

unique_id	API password	signature
26aa150ee68b1b2d6758a0e6c44fce4c	b5af4c9cf497662e00b78550fd87e65eb415f42f	3d82fef85cb60e289d52c3854b97e832d4a2e95ad205e79d9cb4dc7439025cf0aabfd5b88e77d9260c5c4aa7434e01c5f00fc9c2487407c48efeddf6159ab526
3f760162ef57a829011e5e2379b3fa17	50fd87e65eb415f42fb5af4c9cf497662e00b785	14519d0db2f7f8f407efccc9b099c5303f55c0262e3b9132e5bcc97f7febf5f9ab19df03929c1ead271be79807b4086321a023743d2b6b1278c2082b61cf3ff0

Note:

You **must** either use the signature to verify the notification's integrity or make a reconcile to check the final transaction status.

When receiving the notification, you are required to render an xml page containing the payments's unique_id so that the gateway knows that you have accepted the notification. If the XML is not delivered, the notification is sent periodically until the XML is received.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification_echo>
  <unique_id>3f760162ef57a829011e5e2379b3fa17</unique_id>
</notification_echo>
```

7.2.4 Cancel

Cancel can be used to cancel a payment, as long as it hasn't been processed.

Cancel request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Cancel Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<cancel>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</cancel>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Cancel Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>canceled</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot cancel wpf_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

7.2.5 Capture

Capture can be used to capture a successfully authorized payment.

Capture request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Capture Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<capture>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</capture>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Capture Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>captured</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot capture wpf_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

7.2.6 Refund

Refund can be used to refund a successfully processed payment.

Refund request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Capture Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<refund>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</refund>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Refund Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>refunded</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot refund wpf_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

7.2.7 Void

Void can be used to void a successfully processed payment.

Void request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Capture Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<void>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</void>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Void Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>voided</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot voided wpf_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

7.2.8 Errors

Note:

WPF returns the same error codes/XML as the regular Server-to-Server API methods (e.g. sale). See the Error Section for more details.

7.2.9 Reconcile

Reconcile can be used to retrieve data about a payment. This can be useful if you want to retrieve information about a payment whose status is timeout, which returned an error or has changed eg. has beed chargebacked.

Reconcile request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Reconcile Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<reconcile>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</reconcile>
```

Successful response: Example Reconcile Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>approved</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
  <payment_transaction>
    <status>approved</status>
    <transaction_type>sale</transaction_type>
    <unique_id>bad08183a9ec545daf0f24c48361aa10</unique_id>
    <transaction_id>mtid20110408144716135536962</transaction_id>
    <channel_token>e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa</channel_token>
    <mode>test</mode>
    <timestamp>2011-04-08T14:46:40Z</timestamp>
```

```
    <descriptor>hypercharge.net/bogus +49123456789</descriptor>
    <amount>5000</amount>
    <currency>USD</currency>
    <customer_email>john.doe@example.com</customer_email>
    <customer_phone>+11234567890</customer_phone>
    <technical_message>TESTMODE: No real money will be transferred!</technical_message>
    <message>TESTMODE: No real money will be transferred!</message>
    <billing_address>
      <first_name>John</first_name>
      <last_name>Doe</last_name>
      <address1>32, Doestreet</address1>
      <address2></address2>
      <zip_code>12345</zip_code>
      <city>New York</city>
      <state>NY</state>
      <country>US</country>
    </billing_address>
  </payment_transaction>
</payment>
```

Example Response XML for voided payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>WpfPayment</type>
  <status>voided</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
  <payment_transaction>
    <status>voided</status>
    <transaction_type>sale</transaction_type>
    <unique_id>bad08183a9ec545daf0f24c48361aa10</unique_id>
    <transaction_id>mtid20110408144716135536962</transaction_id>
    <channel_token>e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa</channel_token>
    <mode>test</mode>
    <timestamp>2011-04-08T14:46:40Z</timestamp>
    <descriptor>hypercharge.net/bogus +49123456789</descriptor>
    <amount>5000</amount>
    <currency>USD</currency>
    <customer_email>john.doe@example.com</customer_email>
    <customer_phone>+11234567890</customer_phone>
    <billing_address>
      <first_name>John</first_name>
      <last_name>Doe</last_name>
      <address1>32, Doestreet</address1>
      <address2></address2>
      <zip_code>12345</zip_code>
      <city>New York</city>
      <state>NY</state>
      <country>US</country>
```

```
    </billing_address>
  </payment_transaction>
  <payment_transaction>
    <status>approved</status>
    <transaction_type>void</transaction_type>
    <unique_id>bad08183a9ec545daf0f24c48361aa10</unique_id>
    <transaction_id>mtid20110408144716113553696 2</transaction_id>
    <channel_token>e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa</channel_token>
    <mode>test</mode>
    <timestamp>2011-04-08T14:46:40Z</timestamp>
    <descriptor>hypercharge.net/bogus +49123456789</descriptor>
    <amount>5000</amount>
    <currency>USD</currency>
  </payment_transaction>
</payment>
```

Note:

Please note that the response may include multiple payment_transaction records, since a WPF payment is a container class for multiple payment transactions.

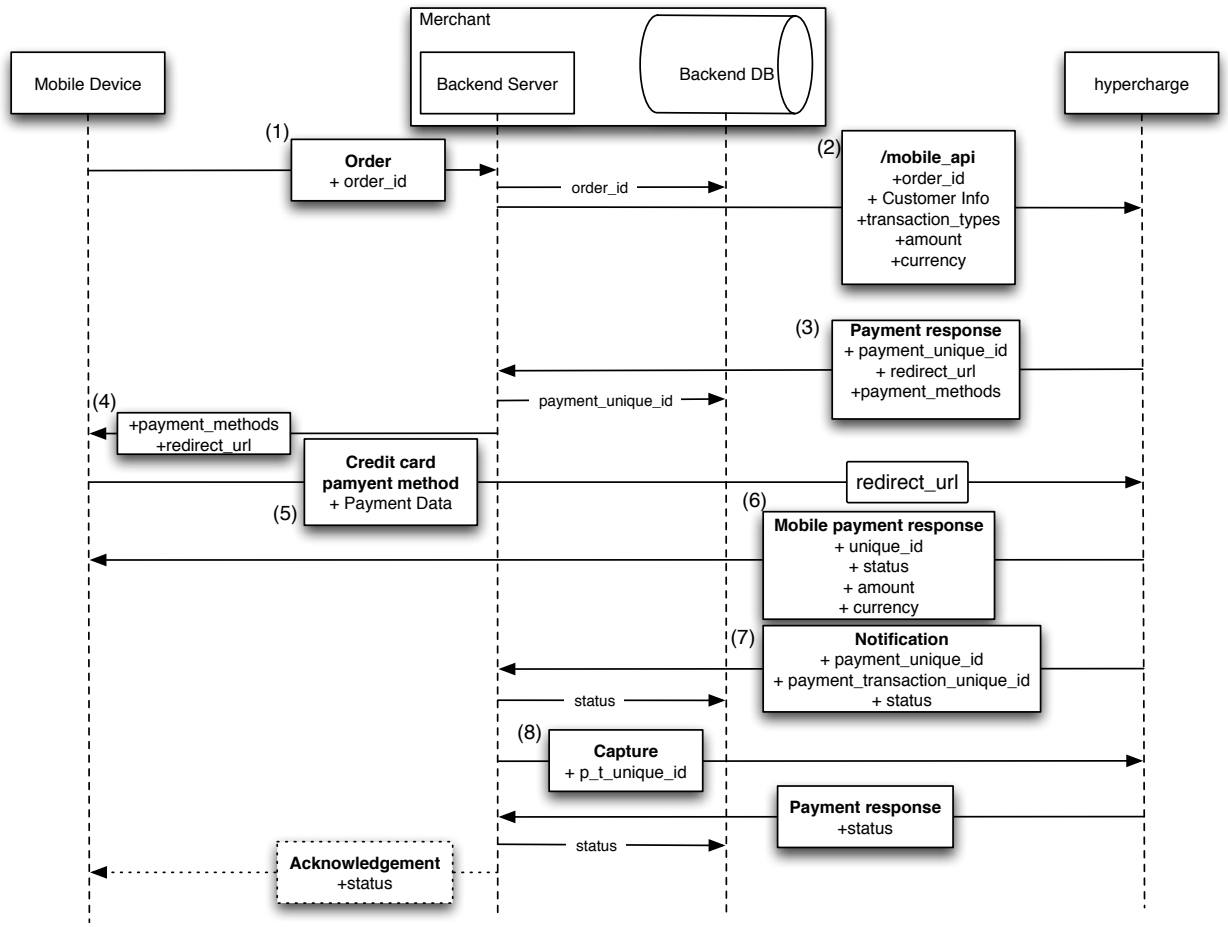
Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>Unknown system error. Please contact support.</technical_message>
  <message>Transaction failed, please contact support!</message>
</payment>
```

8 Mobile Payment

The Mobile Payment API is a component of the hypercharge payment gateway. It provides merchants with mobile solutions to easily process their customers' mobile payments. Through a single point of integration, the merchant can offer his customers multiple synchronous payment methods secure and instantly.

8.1 Workflow



In the example above the customer orders a product via mobile device **(1)**. Subsequently the merchant registers a payment **(2)** on the hypercharge payment gateway through a request to the Mobile

Payment API, which carries the mandatory, inital payload (e.g. amount, currency, etc.). The response to this request (if successful) is a redirect_url **(3)**, which the merchants hands over to the mobile application **(4)**. The mobile application then collects the remaining payment information like credit card number or bank account number and posts a XML request to the received payment URL **(5)**. The mobile Application receives a synchronous response **(6)** to the payment request **(5)**. After the payment has been processed and reached a final state the merchant is sent a notification **(7)** to the `notification_url` supplied in the initial `create` request. The merchant **must** either use the notifcation's signature to verify the payment's integrity or make a reconcile to check the final payment status. However, we urge all merchants to always do a reconcile. If necessary the merchant backend can capture the payment in case an authorize transaction type was used **(8)**.

Note:

The Mobile Payment API only supports synchronous transaction types like sale, authorize, debit_sale and their recurring variants. Asynchronous transaction types like sale3d or ideal_sale are not supported.

8.2 Mobile Payment API

8.2.1 URLs:

Create: The URL for the Mobile Payment API `create` method is:

```
https://payment.hypercharge.net/payment
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment
```

Reconcile: The URL for the Mobile Payment API `reconcile` method is:

```
https://payment.hypercharge.net/payment/reconcile
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment/reconcile
```

Cancel: The URL for the Mobile Payment API `cancel` method is:

```
https://payment.hypercharge.net/payment/cancel
```

For the test system the URL is:

```
https://testpayment.hypercharge.net/payment/cancel
```

8.2.2 Create

Parameters:

parameter	req?	format	description
type	*required*	string	constant value "MobilePayment"
transaction_id	*required*	string(255)	unique transaction id defined by merchant
amount	*required*	integer > 0	Amount of the payment in cents
currency	*required*	string(3)	Currency code in ISO 4217
wire_reference_id	*optional*	string(10)	Wire reference id for the customer to use with the bankwire
sepa_mandate_id	*conditional*[1]	string(35)	signed mandate ID between merchant and customer
sepa_mandate_ signature_date	*conditional*[1]	date	mandate signature date; Format: YYYY-MM-DD
company_name	*conditional*[2]	string(255)	Company name in case of B2B. Contact support for further information.
usage	*required*	string	statement, as it appears in the customer's bank statement
customer_email	*conditional*[3]	string	the customer's e-mail address
customer_phone	*conditional*[3]	string	the customer's phone number
notification_url	*required*	string	URL at merchant's site where gateway sends outcome of transaction after the payment has reached a final state. This should be an SSL secured page
billing_address	*conditional*[3]		
first_name	*required*	string(255)	Customer first name
last_name	*required*	string(255)	Customer last name
address1	*required*	string(255)	first line of address
address2	*optional*	string(255)	second line of address
city	*required*	string(255)	City
zip_code	*required*	string(32)	Zip Code
state	*conditional*[4]	string(2)	state code in ISO 3166-2, only needed for USA and Canada
country	*required*	string(2)	country code in ISO 3166
transaction_types	*optional*		the transaction types that the merchant is willing to accept payments for
transaction_type	*required*	string	one of the available synchronous transaction types (e.g. sale, authorize, etc.)
risk_params	*optional*		list of risk params as described in the Advanced risk management with RiskParams section
user_id (example)	*required*	string	the customer's ID within the merchant's system (example)
retries	*optional*	integer > 0	The acceptable payment retries for the customers if a transaction failed.
expires_in	*optional*	integer > 300, integer < 86400	The acceptable "time-to-live" in seconds between the merchant's creation of a payment and the customer's interaction. This can be set to anything between 5 minutes and 24 hours

Example Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <transaction_id>wev238f328nc</transaction_id>
  <usage>Order ID 500, Shoes</usage>
  <notification_url>https://example.com/notification</notification_url>
  <amount>5000</amount>
  <currency>USD</currency>
  <customer_email>john.doe@example.com</customer_email>
  <customer_phone>+11234567890</customer_phone>
  <billing_address>
    <first_name>John</first_name>
    <last_name>Doe</last_name>
    <address1>23, Doestreet</address1>
    <zip_code>11923</zip_code>
    <city>New York City</city>
    <state>NY</state>
    <country>US</country>
  </billing_address>
  <transaction_types>
    <transaction_type>sale</transaction_type>
    <transaction_type>debit_sale</transaction_type>
  </transaction_types>
  <risk_params>
    <user_id>123456</user_id>
  </risk_params>
  <retries>3</retries>
  <expires_in>300</expires_in>
</payment>
```

[1] required for SEPA.
[2] required for PurchaseOnAccount to initiate B2B transactions.
[3] is only required, if billing address is required by your account configuration.
[4] is required, if `country is either USA or Canada.`

Successful response:

name	type	description
type	string	constant value "MobilePayment"
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if reconciling a payment)
transaction_id	string(255)	unique transaction id defined by merchant
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the payment, can be "live" or "test"
timestamp	string	time when the payment was created in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
amount	integer > 0	Amount of transaction in cents
currency	string	Currency code in ISO 4217
redirect_url	url	URL where the mobile application submits the payment to
cancel_url	url	URL where the mobile application submits a cancel request to
cancel_url	url	URL where the mobile application can cancel the payment e.g. if user exits the mobile application
payment_methods		list of available payment methods
payment_method	string	e.g. "credit_card", "direct_debit", etc.

Example Successful Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>new</status>
  <unique_id>eabcb7a41044e764746b0c7e32c1e9d1</unique_id>
  <transaction_id>wev238f328nc</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-19T15:00:11Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
  <redirect_url>https://testpayment.hypercharge.net/mobile/submit/eabcb7a41044e764746b0c7e32
c1e9d1</redirect_url>
  <cancel_url>https://testpayment.hypercharge.net/mobile/cancel/eabcb7a41044e764746b0c7e32c1
e9d1</cancel_url>
  <payment_methods>
    <payment_method>credit_card</payment_method>
    <payment_method>direct_debit</payment_method>
  </payment_methods>
</payment>
```

Error response:

name	type	description
type	string[1]	constant value "MobilePayment"
status	string	status of the transaction, see **states**
unique_id	string(32)[1]	unique id defined by gate (must later be used if reconciling a payment)
transaction_id	string(255)[1]	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <unique_id>eabcb7a41044e764746b0c7e32c1e9d1</unique_id>
  <transaction_id>wev238f328nc</transaction_id>
  <status>error</status>
  <code>100</code>
  <technical_message>Unknown system error. Please contact support.</technical_message>
  <message>Transaction failed, please contact support!</message>
</payment>
```

[1] is only available, if payment is persistent within hypercharge system.

8.2.3 Submit payment

The mobile application posts the payment data to the redirect_url to process the payment. The unique_id of the payment session is within the redirect_url.

Credit card

Request:

parameter	req?	format	description
card_holder	*required*	string(255)	Full name of customer as printed on cc (first name and lastname at least)
card_number	*required*	string(13..21)	credit card number
cvv	*required**	3 to 4 digits	cvv of cc, requirement is based on channel configuration
expiration_month	*required*	MM	Expiration month as printed on credit card
expiration_year	*required*	YYYY	Expiration year as printed on credit card

Example Request XML:

```
<payment>
  <payment_method>credit_card</payment_method>
  <card_holder>Manfred Mann</card_holder>
  <card_number>4200000000000000</card_number>
  <cvv>123</cvv>
  <expiration_year>2015</expiration_year>
  <expiration_month>12</expiration_month>
</payment>
```

Direct debit

Request:

parameter	req?	format	description
bank_account_holder	*required*	string(255)	name of the bank account holder
bank_account_number	*required*	string(16)	customers bank account number
bank_number	*required*	string(16)	customers bank number

Example Request XML:

```
<payment>
  <payment_method>direct_debit</payment_method>
  <bank_account_holder>Manfred Test</bank_account_holder>
  <bank_account_number>9290701</bank_account_number>
  <bank_number>20050550</bank_number>
</payment>
```

Successful response:

name	type	description
type	string	constant value "MobilePayment"
status	string	status of the transaction, see **states**
unique_id	string(32)	unique id defined by gate (must later be used if reconciling a payment)
transaction_id	string(255)	unique transaction id defined by merchant
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.
mode	string	Mode of the payment, can be "live" or "test"
timestamp	string	time when the payment was created in ISO 8601 Combined date and time, e.g. 2007-08-30T17:46:11Z
amount	integer > 0	Amount of transaction in cents
currency	string	Currency code in ISO 4217

Example Successful Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>approved</status>
  <unique_id>eabcb7a41044e764746b0c7e32c1e9d1</unique_id>
  <transaction_id>wev238f328nc</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-19T15:00:11Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Error response:

name	type	description
type	string[1]	constant value "MobilePayment"
status	string	status of the transaction, see **states**
unique_id	string(32)[1]	unique id defined by gate (must later be used if reconciling a payment)
transaction_id	string(255)[1]	unique transaction id defined by merchant
code	integer	Error code according to Error code table
technical_message	string(255)	Technical error message (for internal use only, not to be displayed to users).
message	string(255)	Human readable error message which can be displayed to users.

[1] is only available, if payment is persistent within hypercharge system.

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>error</status>
  <unique_id>eabcb7a41044e764746b0c7e32c1e9d1</unique_id>
  <transaction_id>wev238f328nc</transaction_id>
  <code>100</code>
  <technical_message>Unknown system error. Please contact support.</technical_message>
  <message>Transaction failed, please contact support!</message>
</payment>
```

8.2.4 Notification

Mobile Payment Notifications are sent once the payment has reached a final state and are transmitted via HTTP POST (application/x-www-form-urlencoded) with the following parameters:

Note:

Apart from the workflow described above, the hypercharge system will also use the `notification_url` endpoint to send notifications to the merchant if a payment is chargebacked.

An example notification:

```
payment_transaction_id=mtid2011040814471611355369 62
&payment_status=approved
&payment_unique_id=26aa150ee68b1b2d6758a0e6c44fce4c
&notification_type=MobilePayment
&signature=c5219b3d385e74496b2b48a5497b347e102849f10eacd25b062f823b
```

In case a payment transaction was processed these additional parameters are returned:

```
&payment_transaction_channel_token=e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa
&payment_transaction_unique_id=bad08183a9ec545daf0f24c48361aa10
&payment_transaction_transaction_type=sale
```

In case a recurring schedule was created for the payment these additional parameters are returned:

```
&recurring_schedule_unique_id=bad08183a9ec545daf0f24c48361aa10
&recurring_schedule_end_date=2013-01-21 00:00:00
```

Parameters:

name	type	description
payment_transaction_id	string	merchant generated tranaction_id
payment_status	string	status of the payment transaction
payment_unique_id	string	unique_id generated by hypercharge, required for mobile payment reconciliation
payment_transaction_transaction_type	string	transaction_type for the transaction eg: sale3d
notification_type	string	constant value "MobilePayment"
signature	string	the signature of the notification, should be used to verify the the notifcation was send by hypercharge
payment_transaction_channel_token	string	the channel_token as used in the processing url
payment_transaction_unique_id	string	unique_id generated by hypercharge
recurring_schedule_unique_id	string	reference_id of the recurring schedule the payment was processed for
recurring_schedule_end_date	date	date of the final recurring event

Status will be one of the following: approved, declined, error, timeout, pending, pending_async, canceled, refunded, chargebacked, chargeback_reversed, pre_arbitrated, captured, and voided.

The signature is a mean of security to ensure that the gate is really the sender of the notification. It is generated by concatenating the unique_id of the payment with your API password and generating a SHA-512 Hash (Hex) of the string:

```
SHA-512 Hash (Hex) of <payment_unique_id><Your Merchant API password>
```

Notification signature examples (Mobile)

unique_id	API password	signature
26aa150ee68b1b2d 6758a0e6c44fce4c	b5af4c9cf497662e00b7 8550fd87e65eb415f42f	3d82fef85cb60e289d52c3854b97e832d4a2e95a d205e79d9cb4dc7439025cf0aabfd5b88e77d926 0c5c4aa7434e01c5f00fc9c2487407c48efeddf6 159ab526
3f760162ef57a829 011e5e2379b3fa17	50fd87e65eb415f42fb5 af4c9cf497662e00b785	14519d0db2f7f8f407efccc9b099c5303f55c026 2e3b9132e5bcc97f7febf5f9ab19df03929c1ead 271be79807b4086321a023743d2b6b1278c2082b 61cf3ff0

Note:

You **must** either use the signature to verify the notification's integrity or make a reconcile to check the final transaction status.

When receiving the notification, you are required to render an xml page containing the payments's unique_id so that the gateway knows that you have accepted the notification. If the XML is not delivered, the notification is sent periodically until the XML is received.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification_echo>
  <unique_id>3f760162ef57a829011e5e2379b3fa17</unique_id>
</notification_echo>
```

8.2.5 Cancel by merchant

Cancel can be used to cancel a payment, as long as it hasn't been processed.

Cancel request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Cancel Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<cancel>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</cancel>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Cancel Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>canceled</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716113553696 2</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>Invalid workflow: Cannot cancel payment at this point.
</technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

8.2.6 Cancel by customer

Cancel by customer can be used to cancel a payment from the mobile device, as long as it hasn't been processed.

Cancel request is sent via POST to the cancel_url received by create request.

The URL for the Mobile Payment API `cancel` method for the customer is:

```
https://payment.hypercharge.net/mobile/cancel/[unique_id]
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Cancel Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>canceled</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>Invalid workflow: Cannot cancel payment at this point.
</technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

8.2.7 Capture

Capture can be used to capture a successfully authorized payment.

Capture request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Capture Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<capture>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</capture>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Capture Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>captured</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot capture mobile_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

8.2.8 Refund

Refund can be used to refund a successfully processed payment.

Refund request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Capture Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<refund>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</refund>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Refund Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>refunded</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot refund mobile_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

8.2.9 Void

Void can be used to void a successfully processed payment.

Void request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Capture Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<void>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</void>
```

Successful response: The response is almost identical to the Reconcile response, except in most cases it will not include payment_transation records. Example Void Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>voided</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
</payment>
```

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>
    Invalid workflow: Cannot voided mobile_payment at this point.
  </technical_message>
  <message>Something went wrong, please contact support!</message>
</payment>
```

8.2.10 Errors

Note:

The mobile payment returns the same error codes/XML as the regular Server-to-Server API methods (e.g. sale). See the Error Section for more details.

8.2.11 Reconcile

Reconcile can be used to retrieve data about a payment. This can be useful if you want to retrieve information about a payment whose status is timeout, which returned an error or has changed eg. has beed chargebacked.

Reconcile request are handled exactly like transaction requests via XML.

Parameters:

parameter	req?	format	description
unique_id	*required*	string(32)	unique id as returned by the create request

Example Reconcile Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<reconcile>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
</reconcile>
```

Successful response: Example Reconcile Response XML for successful payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>approved</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716135536962</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
  <payment_transaction>
    <status>approved</status>
    <transaction_type>sale</transaction_type>
    <unique_id>bad08183a9ec545daf0f24c48361aa10</unique_id>
    <transaction_id>mtid20110408144716135536962</transaction_id>
    <channel_token>e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa</channel_token>
    <mode>test</mode>
    <timestamp>2011-04-08T14:46:40Z</timestamp>
```

```
    <descriptor>hypercharge.net/bogus +49123456789</descriptor>
    <amount>5000</amount>
    <currency>USD</currency>
    <customer_email>john.doe@example.com</customer_email>
    <customer_phone>+11234567890</customer_phone>
    <technical_message>TESTMODE: No real money will be transferred!</technical_message>
    <message>TESTMODE: No real money will be transferred!</message>
    <billing_address>
      <first_name>John</first_name>
      <last_name>Doe</last_name>
      <address1>32, Doestreet</address1>
      <address2></address2>
      <zip_code>12345</zip_code>
      <city>New York</city>
      <state>NY</state>
      <country>US</country>
    </billing_address>
  </payment_transaction>
</payment>
```

Example Response XML for voided payment:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <type>MobilePayment</type>
  <status>voided</status>
  <unique_id>26aa150ee68b1b2d6758a0e6c44fce4c</unique_id>
  <transaction_id>mtid20110408144716113553696 2</transaction_id>
  <technical_message>TESTMODE: No real money will be transferred!</technical_message>
  <message>TESTMODE: No real money will be transferred!</message>
  <mode>test</mode>
  <timestamp>2011-04-08T14:46:27Z</timestamp>
  <amount>5000</amount>
  <currency>USD</currency>
  <payment_transaction>
    <status>voided</status>
    <transaction_type>sale</transaction_type>
    <unique_id>bad08183a9ec545daf0f24c48361aa10</unique_id>
    <transaction_id>mtid20110408144716113553696 2</transaction_id>
    <channel_token>e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa</channel_token>
    <mode>test</mode>
    <timestamp>2011-04-08T14:46:40Z</timestamp>
    <descriptor>hypercharge.net/bogus +49123456789</descriptor>
    <amount>5000</amount>
    <currency>USD</currency>
    <customer_email>john.doe@example.com</customer_email>
    <customer_phone>+11234567890</customer_phone>
    <billing_address>
      <first_name>John</first_name>
      <last_name>Doe</last_name>
      <address1>32, Doestreet</address1>
      <address2></address2>
      <zip_code>12345</zip_code>
      <city>New York</city>
      <state>NY</state>
      <country>US</country>
```

```
    </billing_address>
  </payment_transaction>
  <payment_transaction>
    <status>approved</status>
    <transaction_type>void</transaction_type>
    <unique_id>bad08183a9ec545daf0f24c48361aa10</unique_id>
    <transaction_id>mtid20110408144716113553696 2</transaction_id>
    <channel_token>e9fd7a957845450fb7ab9dccb498b6e1f6e1e3aa</channel_token>
    <mode>test</mode>
    <timestamp>2011-04-08T14:46:40Z</timestamp>
    <descriptor>hypercharge.net/bogus +49123456789</descriptor>
    <amount>5000</amount>
    <currency>USD</currency>
  </payment_transaction>
</payment>
```

Note:

Please note that the response may include multiple payment_transaction records, since a payment is a container class for multiple payment transactions.

Example Error Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
  <status>error</status>
  <code>100</code>
  <technical_message>Unknown system error. Please contact support.</technical_message>
  <message>Transaction failed, please contact support!</message>
</payment>
```

9 Errors

9.1 Error classes

9.1.1 Systems errors (100..199)

Transaction could not be processed and was not passed to acquirer.

9.1.2 Communication errors (200..299)

Transaction could not be processed properly. Acquirer could not be reached or returned invalid data. Errors 230 - 250 need to be reconciled as they might have been processed properly acquirer-wise.

9.1.3 Input data errors (300..399)

Transactions cannot be processed due to invalid incoming data in your request.

9.1.4 Workflow errors (400..499)

Workflow errors will occur if you trigger a transaction that is not possible at this time in the workflow, e.g. a refund on a declined transaction.

9.1.5 Processing errors (500..599)

These errors occur when a transaction was declined by the acquirer.

9.1.6 Risk errors (600..699)

Risk errors occur when any of the risk management systems will not let the transaction pass through.

9.1.7 Acquirer errors (900..999)

Acquirer errors occur when the acquirer is unreachable or has other technical problems. If you experience this kind of errors, contact support.

9.2 Error code table

code	name	description
1	Error	Undefined error.
100	SystemError	A general system error occured
101	MaintenanceError	System is undergoing maintenance, request could not be handled.
110	AuthenticationError	Login failed. Check your API credentials.
120	ConfigurationError	Configuration error occured, e.g. channel not configured properly. Check channel settings.
200	CommunicationError	Communication with acquirer failed, please contact support.
210	ConnectionError	Connection to acquirer could not be established, please contact support.
220	AccountError	Acquirer account data invalid, please contact support.
230	TimeoutError	Acquirer does not respond within given timeframe - please reconcile
240	ResponseError	Acquirer returned invalid response - please reconcile and contact support
250	ParsingError	Acquirer response could not be parsed - please reconcile and contact support.
260	CustomerTimeoutError	DOCUMENTATION MISSING
300	InputDataError	Invalid were data sent to the API.
310	InvalidTransactionTypeError	Invalid transaction type was passed to API. See **transaction types**.
320	InputDataMissingError	Required argument is missing. Check parameters.
330	InputDataFormatError	Argument passed in invalid format. Check parameters.
340	InputDataInvalidError	Argument passed in valid format but makes no sense (e.g. incorrect country code or currency). Check parameters.
350	InvalidXmlError	The input XML could not be parsed due to invalid code. Please check XML data.
360	InvalidConentTypeError	DOCUMENTATION MISSING
400	WorkflowError	A transaction was triggered that is not possible at this time in the workflow, e.g. a refund on a declined transaction.
410	ReferenceNotFoundError	Reference transaction was not found.
420	ReferenceWorkflowError	Wrong Workflow specified.
430	ReferenceInvalidatedError	Reference transaction already invalidated!
440	ReferenceMismatchError	Data mismatch with reference, e.g. amount exceeds reference
450	DoubletTransactionError	Transaction doublet was detected, transaction was blocked. This happens if several transactions with same amount, card holder, cc number, cvv and expiry date are sent within 5 minutes.
460	TransactionNotFoundError	The referenced transaction could not be found.

code	name	description
500	ProcessingError	Transaction declined by acquirer
510	InvalidCardError	Transaction declined, Credit card number is invalid.
520	ExpiredCardError	Transaction declined, expiration date not in the future or date invalid.
530	TransactionPendingError	Transaction pending.
540	CreditExceededError	Amount excceds credit card limit.
600	RiskError	Transaction declined by risk management
610	CardBlacklistError	Card is blacklisted
611	BinBlacklistError	BIN blacklisted.
612	CountryBlacklistError	Country blacklisted.
613	IpBlacklistError	IP address blacklisted.
614	BlacklistError	value from payment_transaction or risk_params is blacklisted.
620	CardLimitExceededError	Card limit exceeded configured limits.
621	ChannelLimitExceededError	Channel limits exceeded.
622	ContractLimitExceededError	Contract limits exceeded.
623	CardVelocityExceededError	Velocity by unknown card exceeded!
624	CardTicketSizeExceededError	Ticketsize by unknown card exceeded!
625	UserLimitExceededError	User limit exceeded configured limits.
626	MultipleFailureDetectionError	Found user transaction declined by acquirer. Try again later!
627	CSDetectionError	DOCUMENTATION MISSING
628	RecurringLimitExceededError	DOCUMENTATION MISSING
690	AvsError	Address Verification failed.
900	RemoteError	Some error occured on the acquirer. Contact support.
910	RemoteSystemError	Some error occured on the acquirer
920	RemoteConfigurationError	Acquirer configuration error
930	RemoteDataError	Some passed data caused an error on the acquirer
940	RemoteWorkflowError	Remote workflow error
950	RemoteTimeoutError	Acquirer has timeouted with clearing network
960	RemoteConnectionError	Acquirer could not reach clearing network