Rabobank
COO Operations NL

Software Development Kit Rabo OmniKassa 2.0

Developer's manual version 1.13 UK

Version:	1.13 UK July 2019
Contact e-mail address:	contact@omnikassa.rabobank.nl

Disclaimer: This manual is exclusively intended for third parties who take care of the technical link between Rabo OmniKassa and the customer's webshop for and on behalf of clients. Rabobank is not liable for possible damage resulting from errors in or incorrect use of the manual. For example, but not limited to, damage that occurs because the technical link between Rabo OmniKassa and the customer's webshop does not work (completely) correctly. Rabobank has the right to change this manual..

Inhoudsopgave

1. Vereisten en installatie instructies

1.1. PHP SDK

For the PHP SDK you need to have PHP version 7.1 or higher. To use the PHP SDK, extract the ZIP file into a directory on the system on which the webshop is installed and so that the files of the PHP SDK are accessible from the PHP files of the webshop. Then include the composer-generated autoloader.php file in your code using:

```
include('<installatiedirectory>/sdk/vendor/autoload.php');
```

1.2. Java SDK

If you are using the Java SDK, you should have Java Runtime Environment version 1.8 or 11 available. To use the Java SDK, extract the ZIP file to the system on which the webshop is installed. The actual SDK is a single jar and is located in the SDK subdirectory. The jar files that this SDK depends on are in the Lib subdirectory. Both the SDK jar file and the dependent jar files should be included in the classpath to be able to use the Java SDK.

1.3. .NET SDK

The .net SDK supports the following .net platforms:

- .net Framework (from version 4.5.2)
- .net Core 1. x/2. x (from .net Standard version 1.3)

If applicable, this document will display separate code examples for these platforms.

1.4. SSL configuratie

In order to establish an encrypted connection with Rabo OmniKassa, the Java Runtime environment, .net environment or PHP (depending on the chosen SDK) should be configured with a CA bundle that allows the validity of the certificate of https://betalen.rabobank.nl can be monitored. As a rule, this is already the case and you do not need to do anything.

If you are running PHP under Windows, you may need to configure the php.ini file with the property OpenSSL.Cafile with the path to the CA bundle. Consult the PHP documentation for more information.

2. Introduction

2.1. Purpose

This document describes how a webshop can be connected to the Rabo OmniKassa using the available SDKs. At this moment the Rabobank SDKs provides the following programming languages/platforms: PHP, Java and .NET.

It is possible that this manual already includes functionality that belongs to a payment method that is not yet available in the Rabo OmniKassa dashboard, this is not a problem for the proper operation of the SDK. In the Rabo OmniKassa Dashboard, the available payment methods can be selected.

2.2. Signingkeys and refresh tokens

In the code, the identifiers {Refresh_token} and {Signing_key} refer to the refresh token and the signing key respectively. These indications should not be taken literally but replaced with the values you find in the dashboard of Rabo Omnikassa 2.0.

3. Steps in processing payments

In order to obtain a global picture, this chapter describes the steps in which a payment is made via Rabo Omnikassa. In the remaining chapters each of these steps is described in more detail.

Each payment consists of the following three steps:

1. **Staging order**
   Before the consumer can meet the payment request, the Webshop first announces the order at Rabo Omnikassa. The order will include all the information that Rabo Omnikassa needs to lead the consumer through the payment steps. In a successful order announcement Rabo Omnikassa links a URL back to the payment pages.

2. **Consumer pays the order**
   The webshop redirects the consumer to the URL that Rabo Omnikassa linked to in the step above. The consumer is directed to Rabo OmniKassa Payment page and can fulfill the payment request. When this is done, the consumer is directed back to the webshop.

3. **Receive Updates about orders**
   In this step, Rabo OmniKassa sends a notification to the Webhook URL of the webshop when one or more orders have been processed. By using a unique token (key) in this notification, the webshop can return the final status of these orders to Rabo Omnikassa. This step is optional and will only run if the Webhook URL is configured in the Rabo Omnikassa dashboard.

The following chapters describe how these steps are implemented using the SDK.

3.1. Send payment request

3.1.1. Preparing order

In order to make a payment request, an order must first be prepared. The order contains all the information about the payment request that Rabo OmniKassa needs in order to guide the consumer through the payment steps. The following code blocks are examples for creating an order. The order is subject to requirements and if it does not meet certain criteria, certain payment methods are not available, the order will be cleaned or rejected.

PHP

```
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\Money;
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\OrderItem;
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\PaymentBrand;
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\PaymentBrandForc
e;
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\ProductType;
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\request\Merchant
Order;
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\Address;
```

```
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\CustomerInformat
ion;
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\VatCategory;

$orderItems = [OrderItem::createFrom([
    'id' => '1',
    'name' => 'Test product',
    'description' => 'Description',
    'quantity' => 1,
    'amount' => Money::fromDecimal('EUR', 99.99),
    'tax' => Money::fromDecimal('EUR', 21.00),
    'category' => ProductType::DIGITAL,
    'vatCategory' => VatCategory::HIGH
])];

$shippingDetail = Address::createFrom([
   'firstName' => 'Jan',
   'middleName' => 'van',
   'lastName' => 'Veen',
   'street' => 'Voorbeeldstraat',
   'postalCode' => '1234AB',
   'city' => 'Haarlem',
   'countryCode' => 'NL',
   'houseNumber' => '5',
   'houseNumberAddition' => 'a'
]);

$billingDetails = Address::createFrom([
   'firstName' => 'Jan',
   'middleName' => 'van',
   'lastName' => 'Veen',
   'street' => 'Factuurstraat',
   'postalCode' => '2314BA',
   'city' => 'Haarlem',
   'countryCode' => 'NL',
   'houseNumber' => '15'
]);

$customerInformation = CustomerInformation::createFrom([
    'emailAddress' => 'jan.van.veen@gmail.com',
    'dateOfBirth' => '20-03-1987',
    'gender' => 'M',
    'initials' => 'J.M.',
    'telephoneNumber' => '0204971111'
]);

$order = MerchantOrder::createFrom([
    'merchantOrderId' => '100',
    'description' => 'Order ID: 100',
    'orderItems' => $orderItems,
    'amount' => Money::fromDecimal('EUR', 99.99),
    'shippingDetail' => $shippingDetail,
    'billingDetail' => $billingDetail,
    'customerInformation' => $customerInformation,
    'language' => 'NL',
    'merchantReturnURL' => 'http://localhost/',
    'paymentBrand' => PaymentBrand::IDEAL,
    'paymentBrandForce' => PaymentBrandForce::FORCE_ONCE
]);
```

Java

```
import
nl.rabobank.gict.payments_savings.omnikassa_frontend.sdk.model.*;
import
nl.rabobank.gict.payments_savings.omnikassa_frontend.sdk.model.order_d
etails.*;
import
nl.rabobank.gict.payments_savings.omnikassa_frontend.sdk.model.enums.*
;


OrderItem orderItem = new OrderItem.Builder()
    .withId("1")
    .withQuantity(1)
    .withName("Test product")
    .withDescription("Description")
    .withAmount(Money.fromEuros(EUR, new BigDecimal("99.99")))
    .withTax(Money.fromEuros(EUR, new BigDecimal("21.00")))
    .withItemCategory(ItemCategory.DIGITAL)
    .withVatCategory(VatCategory.HIGH)
    .build();

CustomerInformation customerInformation = new
CustomerInformation.Builder()
    .withTelephoneNumber("0204971111')
    .withInitials("J.M.")
    .withGender("M")
    .withEmailAddress("jan.van.veen@gmail.com")
    .withDateOfBirth("20-03-1987")
    .build();

Address shippingDetails = new Address.Builder()
    .withFirstName("Jan")
    .withMiddleName("van")
    .withLastName("Veen")
    .withStreet("Voorbeeldstraat")
    .withHouseNumber("5")
    .withHouseNumberAddition("a")
    .withPostalCode("1234AB")
    .withCity("Haarlem")
    .withCountryCode(CountryCode.NL)
    .build()

Address billingDetails = new Address.Builder()
    .withFirstName("Jan")
    .withMiddleName("van")
    .withLastName("Veen")
    .withStreet("Factuurstraat")
    .withHouseNumber("5")
    .withHouseNumberAddition("a")
    .withPostalCode("1234AB")
    .withCity("Haarlem")
    .withCountryCode(CountryCode.NL)
    .build()

MerchantOrder order = new MerchantOrder.Builder()
    .withMerchantOrderId("ORDID123")
    .withDescription("An example description")
    .withOrderItems(Collections.singletonList(orderItem))
```

```
    .withAmount(Money.fromDecimal(Currency.EUR, new
BigDecimal("99.99")))
    .withCustomerInformation(customerInformation)
    .withShippingDetail(shippingDetails)
    .withBillingDetail(billingDetails)
    .withLanguage(Language.NL)
    .withMerchantReturnURL("http://localhost/")
    .withPaymentBrand(PaymentBrand.IDEAL)
    .withPaymentBrandForce(PaymentBrandForce.FORCE_ONCE)
    .build();
```

.NET

```
using System.Collections.Generic;
using OmniKassa.Model;
using OmniKassa.Model.Enums;
using OmniKassa.Model.Order;

OrderItem orderItem = new OrderItem.Builder()
   .WithId("1")
   .WithQuantity(1)
   .WithName("Test product")
   .WithDescription("Description")
   .WithAmount(Money.FromEuros(Currency.EUR, 99.99m))
   .WithTax(Money.FromEuros(Currency.EUR, 21.00m))
   .WithItemCategory(ItemCategory.PHYSICAL)
   .WithVatCategory(VatCategory.LOW)
   .Build();

CustomerInformation customerInformation = new
CustomerInformation.Builder()
   .WithTelephoneNumber("0204971111")
   .WithInitials("J.M.")
   .WithGender(Gender.M)
   .WithEmailAddress("jan.van.veen@gmail.com")
   .WithDateOfBirth("20-03-1987")
   .Build();

Address shippingDetails = new Address.Builder()
   .WithFirstName("Jan")
   .WithMiddleName("van")
   .WithLastName("Veen")
   .WithStreet("Voorbeeldstraat")
   .WithHouseNumber("5")
   .WithHouseNumberAddition("a")
   .WithPostalCode("1234AB")
   .WithCity("Haarlem")
   .WithCountryCode(CountryCode.NL)
   .Build();

 Address billingDetails = new Address.Builder()
   .WithFirstName("Jan")
   .WithMiddleName("van")
   .WithLastName("Veen")
   .WithStreet("Factuurstraat")
   .WithHouseNumber("5")
   .WithHouseNumberAddition("a")
   .WithPostalCode("1234AB")
   .WithCity("Haarlem")
   .WithCountryCode(CountryCode.NL)
   .Build();
```

```
MerchantOrder order = new MerchantOrder.Builder()
    .WithMerchantOrderId("ORDID123")
    .WithDescription("An example description")
    .WithOrderItems(new List<OrderItem>() { orderItem })
    .WithAmount(Money.FromEuros(Currency.EUR, 99.99m))
    .WithCustomerInformation(customerInformation)
    .WithShippingDetail(shippingDetails)
    .WithBillingDetail(billingDetails)
    .WithLanguage(Language.NL)
    .WithMerchantReturnURL("http://localhost/")
    .WithPaymentBrand(PaymentBrand.IDEAL)
    .WithPaymentBrandForce(PaymentBrandForce.FORCE_ALWAYS)
    .Build();
```

3.1.2. Field description

Below are all the fields with the name, a description, and the rules to which the field must comply.

MerchantOrder

Field	Description	Line
`MerchantOrderId`	The identity of the order	Required
		May consist only of alphanumeric characters
		This field is shortened to a maximum of 24 characters.
		For AfterPay, this field must be unique
`Description`	A description of the order	Optional
		This field is shortened to a maximum of 35 characters.
`Order Items`	All items that will be ordered by the consumer	Optional
		A maximum of 99 items may be supplied
Amount	The total order amount in cents, including VAT	Required.
		The amount must not exceed 99,999.99

		The amount must be equal to the sum of all order items of the piece price (including VAT) multiplied by the number of copies. Due to the way VAT is calculated, rounding differences can prevent them from being equal. We therefore recommend that you base the total VAT amount on the VAT of the item price instead of the total order amount excluding VAT. For example: Suppose the price of an order item (excluding VAT) is €12.98 and a vat rate of 21% is used. When a consumer order 7 copies, the price including VAT €12.98 + 21% = €15.71 is completed. The total order amount (including VAT) that is given in this field is 7 x €15.71 = €109.97. **Note**– If the amount is not equal to the sum of the amounts of the order items then 1. The order items are filtered from the order announcement, and 2. AfterPay is not possible as a payment method
`ShippingDetails`	The delivery address of this order	Optional
`Language`	The texts on the payment pages will be in this language	Optional, standard we use en.
		ISO 3166-1 Alpha-2, limited to NL, en, FR and DE.
		If `En`, `And`, `Fr`Or `The` Supplied: Then we handle `En`, `And`, `Fr`Or `The` Respectively
		If an unknown value is included: then we handle `And`
`MerchantReturnURL`	The `Url` Where the consumer will return after the payment steps have been completed	Required.
		Must be a valid `Url` Are

PaymentBrand	The payment method to which the consumer is limited	Optional
		Must be one of the following values: IDEAL, AFTERPAY, PAYPAL, MasterCard, VISA, BANCONTACT, MAESTRO, V_PAY When value Cards Included, all card payment methods are offered (MasterCard, Visa, Bancontact, Maestro, and V PAY)
PaymentBrandForce	How to force the payment method	Obligatory if PaymentBrand Otherwise optional
		Must be one of the following values: FORCE_ONCE Or FORCE_ALWAYS
Customer Information	A limited set of consumer information	Optional
BillingDetails	The billing address of this order	Optional
InitiatingParty	An ID identifying the party from which the order announcement was initiated.	Optional. This field must be left empty unless agreed otherwise with Rabobank.

The payment method and the force options work as follows. When the payment method is iDEAL and the Force option FORCE_ONCE has been selected, this means that the consumer will immediatly start an iDEAL payment upon arrival at the Rabobank OmniKassa and thus arrives on the bank selecting screen. The customer then has the option to finalize the payment or to choose another payment method by clicking on <Choose Other Payment method>. With the FORCE_ALWAYS always It is not possible for the consumer to choose another payment method and are the only options to approve or cancel the payment request.

Money

Field	Description	Rules
Currency	The currency	should not be Null
		Must be EUR
		Other currencies are not supported
Amount	The amount	should not be Null

	Must be a natural number

To create a `Money Instance`, use the following code:

PHP

```
//1,75 Euro
$moneyFromCents = Money::fromCents('EUR', 175);
//75,99 Euro
$moneyFromDecimal = Money::fromDecimal('EUR', 75.99);

//Rounding examples
//1000 Euro cents
$amountInCents = Money::fromDecimal('EUR', 9.999)->getAmount();
//999 Euro cents
$amountInCents = Money::fromDecimal('EUR', 9.991)->getAmount();
//1000 Euro cents
$amountInCents = Money::fromDecimal('EUR',9.995)->getAmount();
```

Java

```
//1,75 Euro
Money moneyFromEuros = Money.fromEuros(Currency.EUR, BigDecimal.valueOf(175,
2));
Money moneyFromString = Money.fromEuros(Currency.EUR, new BigDecimal("1.75"));
```

.NET

```
using OmniKassa.Model;
using OmniKassa.Model.Enums;

Money moneyFromDecimal = Money.FromEuros(Currency.EUR, 1.75m);
```

Address / ShippingDetails

Field	Description	Rules
`FirstName`	Consumer's name	should not be `Null` or empty
		Has a maximum length of `50` Characters
`MiddleName`	Consumer Insert	Optional
		Has a maximum length of `20` Characters
`LastName`	Last name of the consumer	should not be `Null` or empty
		Has a maximum length of `50` Characters

`Street`	Street name of the address	should not be `Null` or empty
		Has a maximum length of `100` Characters. **Note**: In the case of card payment at Worldline, this field is shortened to a maximum of 50 characters.
`PostalCode`	Postal Code of the address	should not be `Null` or empty
		Must match the ZIP/Postal code format * of the selected `CountryCode`
`City`	City of Address	should not be `Null` or empty
		Has a maximum length of `40` Characters
`CountryCode`	Country Code of Address	should not be `Null` or empty

		Must be one of the following values: AF, AX, AL, DZ, VI, AS, AD, AO, AI, AQ, AG, AR, AM, AW, AU, AZ, BS, BH, BD, BB, BE, BZ, BJ, BM, BT, BO, BQ, BA, BW, eg, BR, VG, IO, BN, BG, BF, BI, KH, CA, CF, CL, CN, CX, CC, CO, KM , CG, CD, CK, CR, CU, CW, CY, DK, DJ, DM, DO, DE, EC, EG, SV, GQ, ER, EE, ET, FO, FK, FJ, PH, FI, FR, TF, GF, PF, GA, GM, GE, GH, GI, GD, GR, GL, GP, GU, GT, GG, GN, GW, GY, HT, HM, HN , HU, HK, IE, IS, IN, ID, IQ, IR, IL, IT, CI, JM, JP, YE, you, JO, KY, CV, CM, KZ, KE, KG, KI, UM, KW, HR, LA, LS, LV, LB, LR, LY, LI, LT, LU, MO, MK, MG, MW, MV, MY, ML, MT, IM, MA, MH , MQ, MR, MU, YT, MX, FM, MD, MC, MN, ME, MS, MZ, MM, NA, NR, NL, NP, NI, NC, NZ, NE, NG, NU, MP, KP, NO, NF, UG, UA, UZ, OM, at, TL, PK, PW, PS, PA, PG, PY, PE, PN, PL, PT, PR, QA, RE , RO, RU, RW, BL, KN, LC, PM, VC, SB, WS, SM, SA, ST, SN, RS, SC, SL, SG, SH, MF, SX, SI, SK, SD, SO, ES, SJ, LK, SR, SZ, SY, TJ, TW, TZ, TH, TG, TK, TO, TT, TD, CZ, TN, TR, TM, TC, TV , UY, VU, VA, VE, AE, US, GB, VN, WF, EH, BY, ZM, ZW, ZA, GS, KR, SS, SE, CH
`HouseNumber`	House number of the address	Optional
		Has a maximum length of 100 characters. **Note**: In the case of card payment at Worldline, this field is shortened to a maximum of 50 characters.
`HouseNumberAddition`	House number addition of address	Optional
		Has a maximum length of 6 characters

The supported ZIP code formats are as follows. The remaining country codes only have a maximum length of 10 and are indicated in the table below as ' different '.

Country code	Format	Maximum length
Be	\p{Digit} +	4
The	\p{Digit} +	5
En	\p{Digit}{4}\p{Alpha}{2}	6
Else	N/a.	10

CustomerInformation regels

Field	Description	Rules
Emailaddress	The email address of the consumer	Optional
		Must be a valid email address
		has a maximum length of 45 characters.
DateOfBirth	The date of birth of the consumer	Optional
		Must be in the format: DD-MM-YYYY
Gender	Consumer Sex	Optional
		Must be one of the following values: M, F
Initials	Consumer Initials	Optional
		May consist only of alphabet characters
		has a maximum length of 256 characters.
TelephoneNumber	The telephone number of the consumer	Optional
		May consist of numbers and alphabet characters
		has a maximum length of 31 characters.

For payment method AfterPay, Rabo OmniKassa shows a page to the consumer in which he can complete or modify the above fields after the order announcement. Your webshop will **not** be informed of the details that are filled in during this process.

OrderItem Regels

Field	Description	Rules	Notes
`Id`	The ID of the item	Optional	
		has a maximum length of `25` Characters	
`Particular`	Name of the item	should not be `Null` or empty	
		May consist only of alphabet characters	
		has a maximum length of `50` Characters	
`Description`	A description of the item	Optional	
		has a maximum length of `100` Characters	
`Quantity`	Number of copies	should not be `Null` or empty	
		Must be a natural number greater than 0	
`Amount`	The amount per piece in cents, including VAT	should not be `Null` Are	
Tax	VAT per piece in cents	Optional	
`Category`	Category of this item	should not be `Null` Are	
		Must be one of the following values: `DIGITAL`, `Physical`	
`VatCategory`	VAT Category of the item	Optional	

	Must be one of the following values: `1, 2, 3, 4`	The values refer to the different rates used in the Netherlands: 1 = high (currently 21%), 2 = Low (until 31 December 2018 this is 6%, from 1 January 2019 this is 9%), 3 = zero (0%), and 4 = none (exempt from VAT)

The quantity describes how much the consumer wants to have the product, for example 2 apples, or 3 pears. The amount indicates how much one product costs, so 1 apple costs €1.50, or 1 peer costs €1.75, and includes VAT.

If we detail the examples then it becomes:

PHP

```
$apples = OrderItem::createFrom([
    'id' => '1',
    'name' => 'Apple',
    'description' => 'A delicious apple',
    'quantity' => 1,
    'amount' => Money::fromDecimal('EUR', 1.50),
    'tax' => Money::fromDecimal('EUR', 0.14),
    'category' => ProductType::PHYSICAL,
    'vatCategory' => VatCategory::LOW
]);

$pears = OrderItem::createFrom([
    'id' => '2',
    'name' => 'Pear',
    'description' => 'A delicious pear',
    'quantity' => 3,
    'amount' => Money::fromDecimal('EUR', 1.75),
    'tax' => Money::fromDecimal('EUR', 0.16),
    'category' => ProductType::PHYSICAL,
    'vatCategory' => VatCategory::LOW
]);
```

Java

```
OrderItem apples = new OrderItem.Builder()
    .withId("1")
    .withQuantity(2)
    .withName("Apple")
    .withDescription("A delicious apple")
    .withAmount(Money.fromEuros(Currency.EUR,
BigDecimal.valueOf("1.50")))
    .withTax(Money.fromEuros(Currency.EUR,
BigDecimal.valueOf("0.14")))
    .withItemCategory(ItemCategory.PHYSICAL)
    .withVatCategory(VatCategory.LOW)
    .build();

OrderItem pears = new OrderItem.Builder()
    .withId("2")
    .withQuantity(3)
```

```
    .withName("Pear")
    .withDescription("A delicious pear")
    .withAmount(Money.fromEuros(Currency.EUR,
BigDecimal.valueOf("1.75")))
    .withTax(Money.fromEuros(Currency.EUR,
BigDecimal.valueOf("0.16")))
    .withItemCategory(ItemCategory.PHYSICAL)
    .withVatCategory(VatCategory.LOW)
    .build();
```

.NET

```
OrderItem apples = new OrderItem.Builder()
   .WithId("1")
   .WithQuantity(2)
   .WithName("Apple")
   .WithDescription("A delicious apple")
   .WithAmount(Money.FromEuros(Currency.EUR, 1.50m))
   .WithTax(Money.FromEuros(Currency.EUR, 0.14m))
   .WithItemCategory(ItemCategory.PHYSICAL)
   .WithVatCategory(VatCategory.LOW)
   .Build();

OrderItem pears = new OrderItem.Builder()
   .WithId("2")
   .WithQuantity(3)
   .WithName("Pear")
   .WithDescription("A delicious pear")
   .WithAmount(Money.FromEuros(Currency.EUR, 1.75m))
   .WithTax(Money.FromEuros(Currency.EUR, 0.16m))
   .WithItemCategory(ItemCategory.PHYSICAL)
   .WithVatCategory(VatCategory.LOW)
   .Build();
```

Discount

A discount in the order is also provided by an order item but the amount and (if applicable) the tax fields have a negative value. The following examples describe discount of 10 euros:

PHP

```
$discount = OrderItem::createFrom([
    'id' => '1',
    'name' => 'Discount',
    'description' => 'Frequent buyer',
    'quantity' => 1,
    'amount' => Money::fromDecimal('EUR', -10),
    'tax' => Money::fromDecimal('EUR', -0.9),
    'category' => ProductType::PHYSICAL,
    'vatCategory' => VatCategory::LOW
]);
```

Java

```
OrderItem discount = new OrderItem.Builder()
    .withId("1")
    .withQuantity(1)
    .withName("Discount")
    .withDescription("Frequent buyer")
    .withAmount(Money.fromEuros(Currency.EUR, BigDecimal.valueOf("-
```

```
10")))
    .withTax(Money.fromEuros(Currency.EUR, BigDecimal.valueOf("-
0.9")))
    .withItemCategory(ItemCategory.PHYSICAL)
    .withVatCategory(VatCategory.LOW)
    .build();
```

.NET

```
OrderItem discount = new OrderItem.Builder()
   .WithId("1")
   .WithQuantity(1)
   .WithName("Discount")
   .WithDescription("Frequent buyer")
   .WithAmount(Money.FromEuros(Currency.EUR, -10m))
   .WithTax(Money.FromEuros(Currency.EUR, -0.9m))
   .WithItemCategory(ItemCategory.PHYSICAL)
   .WithVatCategory(VatCategory.LOW)
   .Build();
```

The Amount of the MerchantOrder must, if the OrderItems are filled, match the sum of the OrderItems. If no OrderItems are included, any amount may be entered within the set rules. The total amount can be calculated as follows:

PHP

```
$orderTotalInCents = 0;
foreach ($orderItems as $orderItem) {
    $priceTaxInclusiveInCents = $orderItem->getAmount()->getAmount();
    $orderTotalInCents += $priceTaxInclusiveInCents * $orderItem-
>getQuantity();
}
$orderAmount = Money::fromCents('EUR', $orderTotalInCents);
```

Java

```
BigDecimal orderTotal = BigDecimal.ZERO;
for (OrderItem orderItem : orderItems) {
    BigDecimal amount = orderItem.getAmount().getAmount();
    BigDecimal quantity = new BigDecimal(orderItem.getQuantity());
    orderTotal = orderTotal.add(amount.multiply(quantity));
}
Money orderAmount = Money::fromEuros(Currency.EUR, orderTotal);
```

C#

```
decimal totalAmount = 0.0m;

foreach (var orderItem in orderItems)
{
    decimal amount = orderItem.Amount.Amount;
    int quantity = orderItem.Quantity;
    totalAmount += quantity * amount;
}
Money orderAmount = Money.FromEuros(Currency.EUR, totalAmount);
```

Should there be an incorrect order which was sent and the Rabo OmniKassa and the Rabo OmniKassa can repair it, then an attempt is made to correct the order by removing

or shortening any data. This applies only in the extreme cases and may result in specific payment methods not being available for this order.

Required fields per payment method

Regardless of the payment method, at least the merchantOrderId, amount and the MERCHANTRETURNURL must be included in each order. Depending on the payment method, additional information must be included in the order. The table below shows what data this is, broken down by payment method:

Payment method	Additional information in the order
iDEAL	No additional information is required for this payment method.
PayPal	Although not mandatory, we also recommend that you include order items in the order for additional security regarding PayPal's payment to the Web store owner.
Bancontact Visa Mastercard V PAY Maestro	Although not mandatory, we advise you to include the delivery address (or, if not known, the billing address) in the order for additional certainty about a successful payment.
AfterPay	For AfterPay, the following additional information is required: • Order items with per order item the ID, the description and the sales tax amount or the sales tax category. • Billing or delivery address (if different then both addresses are required). In addition, AfterPay requires that the MerchantOrderId field be unique. The order amount must be at least 5 euro.

If for a payment method The mandatory additional information is missing in the order then the consumer cannot use this method to fulfill the payment request. If the payment method was included in the order using the PaymentBrand field, the announcement will be refused by Rabo OmniKassa.

3.2. Creating endpoint

Besides an order is also an instance of an `Endpoint` needed. With this `Endpoint` It is possible to perform all calls towards the Rabo OmniKassa. An `Endpoint` is instantiated with three parameters: an Url, A `Base64` encoded signing key, and a `Token provider` implementation.

The URL determines whether the production environment or the Sandbox environment is linked:

Environment	Description
Production	In This environment all payments are depreciated from consumers and transferred to the owner of the Webshop
Sandbox	This is a test environment. Payments in this environment do not result in the payment of your account and are only intended to test the connections with your webshop.

The environments can be configured in Rabo OmniKassa Dashboard-for example, which payment methods should be offered.

The signing key is a secret key that can be found in the Rabo OmniKassa Dashboard. This signing key is BASE64 encoded. The signing key is used to encrypt all messages from the calls of Rabo OmniKassa. This is an extra layer to show that messages do come from the webshop and not from any hackers.

Finally, we need the implemention of the `Token provider`. This implementation is responsible for providing and storing token information. For example, an implementation can store the data in a database, on a file system, or in memory. The aim is that your `Token provider` provides a refresh token by default -it must be retrieved before the first call is made. This token can be found in the Rabo OmniKassa Dashboard and is an unique token, with long shelf life, which is used to request an access token from the Rabo OmniKassa. The access token is a short shelf life token that authenticates all calls. This token is delivered to the `Token provider` For storage and also for reading.

PHP

```
<?php

use
nl\rabobank\gict\payments_savings\omnikassa_sdk\connector\TokenProvide
r;

class InMemoryTokenProvider extends TokenProvider
{
    private $map = array();

    /**
     * Construct the in memory token provider with the given refresh
token.
     * @param string $refreshToken The refresh token used to retrieve
the access tokens with.
     */
    public function __construct($refreshToken)
    {
        $this->setValue(static::REFRESH_TOKEN, $refreshToken);
    }

    /**
     * Retrieve the value for the given key.
```

```
     *
     * @param string $key
     * @return string Value of the given key or null if it does not
exists.
     */
    protected function getValue($key)
    {
        return array_key_exists($key, $this->map) ? $this->map[$key] :
null;
    }

    /**
     * Store the value by the given key.
     *
     * @param string $key
     * @param string $value
     */
    protected function setValue($key, $value)
    {
        $this->map[$key] = $value;
    }

    /**
     * Optional functionality to flush your systems.
     * It is called after storing all the values of the access token
and can be used for example to clean caches or reload changes from the
database.
     */
    protected function flush()
    {
    }
}
```

Java

```
public class InMemoryTokenProvider extends TokenProvider {
    private Map<FieldName, String> map = new HashMap<>();

    public InMemoryTokenProvider(String refreshToken) {
        setValue(FieldName.REFRESH_TOKEN, refreshToken);
    }

    @Override
    public String getValue(FieldName key) {
        return map.get(key);
    }

    @Override
    public void setValue(FieldName key, String value) {
        map.put(key, value);
    }
}
```

.NET

```
using System;
using System.Collections.Generic;
using OmniKassa;
```

```
namespace MyNamespace
{
    public sealed class InMemoryTokenProvider : TokenProvider
    {
        private Dictionary<FieldName, String> mMap = new
Dictionary<FieldName, String>();

        public InMemoryTokenProvider(String refreshToken)
        {
            SetValue(FieldName.REFRESH_TOKEN, refreshToken);
        }

        protected override String GetValue(FieldName key)
        {
            mMap.TryGetValue(key, out string value);
            return value;
        }

       protected override void SetValue(FieldName key, String value)
        {
            if (mMap.ContainsKey(key))
            {
                mMap[key] = value;
            }
            else
            {
                mMap.Add(key, value);
            }
        }
    }
}
```

This data can then be used to create an `Endpoint` as follows:.

PHP

```
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\Environment;
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\signing\SigningK
ey;
use nl\rabobank\gict\payments_savings\omnikassa_sdk\endpoint\Endpoint;

$signingKey = new SigningKey(base64_decode('{signing_key}'));
$inMemoryTokenProvider = new InMemoryTokenProvider('{refresh_token}');
$endpoint = Endpoint::createInstance(ENVIRONMENT::PRODUCTION,
$signingKey, $inMemoryTokenProvider);
```

Java

```
import nl.rabobank.gict.payments_savings.omnikassa_frontend.sdk.*;
import
nl.rabobank.gict.payments_savings.omnikassa_frontend.sdk.connector.Tok
enProvider;

TokenProvider inMemoryTokenProvider = new
InMemoryTokenProvider("{refresh_token}");
Endpoint endpoint = Endpoint.createInstance(ENVIRONMENT.PRODUCTION,
"{signing_key}", inMemoryTokenProvider);
```

.NET

```
using OmniKassa;
using MyNamespace;

TokenProvider inMemoryTokenProvider = new
InMemoryTokenProvider("{refresh_token}");
Endpoint endpoint = Endpoint.Create(Environment.PRODUCTION,
"{signing_key}", inMemoryTokenProvider);
```

To link to the sandbox environment, the constant sandbox must be Used as the first parameter. Don't forget to also replace the refresh token and the signing key.

3.3. Sending order

With this `Endpoint` We can announce the order and the URL of the Rabo OmniKassa, where the user can pay the order, will be returned..

PHP

```
$redirectUrl = $endpoint->announceMerchantOrder($order);
//Redirect user to Rabo OmniKassa
header('Location: ' . $redirectUrl);
die();
```

Java

```
MerchantOrderResponse response = endpoint.announce(order);
// Redirect user to Rabo OmniKassa
return "redirect:" + response.getRedirectUrl();
```

.NET Standard 1.3

```
using Microsoft.AspNetCore.Mvc;

MerchantOrderResponse merchantOrderResponse = await
endpoint.AnnounceMerchantOrder(order);
String redirectUrl = response.RedirectUrl;
return new RedirectResult(redirectUrl);
```

.NET Framework

```
using System.Web.Mvc;

String redirectUrl = omniKassa.AnnounceMerchantOrder(order);
return new RedirectResult(redirectUrl);
```

The object of type MerchantOrderResponse as returned by the Java SDK contains the following fields:

Field	Description
redirectUrl	The URL to which the consumer must be redirected to in order to pay for the order.
omnikassaOrderId	A unique ID that Rabo Omnikassa will assign to the order. This ID is needed to link the payment result as returned by the webhook notification mechanism to the correct order (see chapter 5).

Remark: The other SDKs (PHP and .NET) will be updated in the near future to return a MerchantOrderResponse with the same fields as well.

4. Consumer pays order

In Rabo OmniKassa The consumer can pay the order. When this is done, the consumer comes back to the webshop via the `MerchantReturnUrl` Specified during the staging of the order. With this `Url` some `Get` Parameters will be delivered that relate to the order, namely:

Parameter	Description
order_id	This is the order number.
Status	The current known status of the order.
Signature	The signature makes it possible to verify that the current call is authentic.

To verify that the signature is authentic, the following code can be used. It is also advisable to make use of the `PaymentCompletedResponse` class so that the `order_id`, `Status a`nd `Signature` will be cleaned if they contain invalid/unsafe values.

PHP

```
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\signing\SigningK
ey;
use
nl\rabobank\gict\payments_savings\omnikassa_sdk\model\response\Payment
CompletedResponse;

$orderId = $_GET['order_id'];
$status = $_GET['status'];
$signature = $_GET['signature'];
$signingKey = new SigningKey(base64_decode('{signing_key}'));
$paymentCompletedResponse =
PaymentCompletedResponse::createInstance($orderId, $status,
$signature, $signingKey);
if (!$paymentCompletedResponse) {
    throw new Exception('The payment completed response was
invalid.');
}

// Use these variables instead of using the URL parameters ($orderId
and $status). Input validation has been performed on these values.
$validatedMerchantOrderId = $paymentCompletedResponse->getOrderID();
$validatedStatus = $paymentCompletedResponse->getStatus();

// ... complete payment
```

Java

```
void paymentCompleted(HttpServletRequest request) {
    byte[] signingKey = Base64Utils.decodeFromString("{signing_key}");
    String orderId = request.getParameter("order_id");
    String status = request.getParameter("status");
    String signature = request.getParameter("signature");
    PaymentCompletedResponse paymentCompletedResponse;
    try {
            paymentCompletedResponse =
```

```
PaymentCompletedResponse.newPaymentCompletedResponse(orderId, status,
signature, signingKey);
    } catch (RabobankSdkException invalidSignatureException){
            throw new IllegalStateException("The payment completed
response was invalid.", invalidSignatureException);
    }

    // Use these variables instead of using the URL parameters
(orderId and status). Input validation has been performed on these
values.
    String validatedOrderId = paymentCompletedResponse.getOrderID();
    String validatedStatus = paymentCompletedResponse.getStatus();

        //... complete payment
}
```

.NET Standard 1.3

```
using System.Collections.Generic;
using Microsoft.Extensions.Primitives;
using Microsoft.AspNetCore.WebUtilities;
using OmniKassa.Model.Response;
using OmniKassa.Model.Enums;

Dictionary<String, StringValues> query =
QueryHelpers.ParseQuery(Request.QueryString.Value);
Dictionary<String, String> dictionary = GetDictionary(query);

PaymentCompletedResponse response =
PaymentCompletedResponse.Create(dictionary, "{signing_key}");

String validatedOrderId = paymentCompletedResponse.OrderId;
PaymentStatus validatedStatus = paymentCompletedResponse.Status;
```

.NET Framework

```
using OmniKassa.Model.Response;
using OmniKassa.Model.Enums;

PaymentCompletedResponse response =
PaymentCompletedResponse.Create(Request.QueryString, "{signing_key}");

String validatedOrderId = paymentCompletedResponse.OrderId;
PaymentStatus validatedStatus = paymentCompletedResponse.Status;
```

If the answer is invalid, it is advisable to refer the user to an error page but to consider the order as ' open '. At a later stage, the actual status of the order can be requested by means of notifications.

5. Receive updates about orders

All the status transitions of an order are tracked by the Rabo OmniKassa so that they can be offered to the webshop with the help of notifications. The Rabo OmniKassa does this by sending a notification to the `WebhookUrl` through a `Post` request with the notification as JSON. The `WebhookUrl` can be configured on the dashboard of Rabo OmniKassa.

A notification looks like this:

Notification

```
{
  "authentication":
"eyJraWQiOiJHS0wiLCJhbGciOiJFUzI1NiJ9.eyJwayMiOjUwMiwiY2lkIjoiYzhjYy1mMThjIiwi
ZXhwIjoxNDc5MTIyODc2fQ.MEUCIQC2Z5WUVTAKcBHISsOVMJIJE8PAbVe5x1ior4bgrTcgCwIgLNo
VIWEmSbQekJTccM89sosAY-8JzN47DGjvdPGdF0w",
  "expiry": "2016-11-25T09:53:46.765+01:00",
  "eventName": "merchant.order.status.changed",
  "poiId": "123"
}
```

With this JSON it is possible to build up `AnnouncementResponse` with which the actual information of the orders can be retrieved.

PHP

```
use nl\rabobank\gict\payments_savings\omnikassa_sdk\endpoint\Endpoint;
use nl\rabobank\gict\payments_savings\omnikassa_sdk\model\Environment;

$signingKey = new SigningKey(base64_decode('{signing_key}'));
$inMemoryTokenProvider = new InMemoryTokenProvider('{refresh_token}');
$endpoint = Endpoint::createInstance(ENVIRONMENT::PRODUCTION,
$signingKey, $inMemoryTokenProvider);

$json = file_get_contents('php://input');
$announcementResponse = new AnnouncementResponse($json, $signingKey);

do {
    $response = $endpoint-
>retrieveAnnouncement($announcementResponse);
    $results = $response->getOrderResults();
    foreach($results as $result) {
        //... Update the order status using the properties in $result
    }
} while ($response->isMoreOrderResultsAvailable());
```

Java

```
@RequestMapping(value = "/webhook", method = RequestMethod.POST)
ResponseEntity webhook(@RequestBody ApiNotification apiNotification)
throws RabobankSdkException {
    byte[] signingKey = Base64Utils.decodeFromString("{signing_key}");
    apiNotification.validateSignature(signingKey);
    Endpoint endpoint = Endpoint.createInstance(...);

    MerchantOrderStatusResponse merchantOrderStatusResponse;
    do {
```

```
        merchantOrderStatusResponse =
endpoint.retrieveAnnouncement(apiNotification);
        for (MerchantOrderResult result :
merchantOrderStatusResponse:getOrderResults()) {
            // Update the order status using the properties in result
        }
    } while (merchantOrderStatusResponse.moreOrderResultsAvailable());

    return new ResponseEntity(HttpStatus.OK);
}
```

.NET Standard 1.3

```
using Microsoft.AspNetCore.Mvc;
using OmniKassa.Model.Response;
using OmniKassa.Model.Response.Notification;

[HttpPost]
public ActionResult Webhook([FromBody] ApiNotification notification)
{
    MerchantOrderStatusResponse response;
    do
    {
            response = await
omniKassa.RetrieveAnnouncement(notification);
        foreach(MerchantOrderResult result in response.OrderResults)
        {
            // Update the order status using the properties in result
        }
    }
    while (response.MoreOrderResultsAvailable);

        return new OkObjectResult("");
}
```

.NET Framework

```
using System.Web.Mvc;
using OmniKassa.Model.Response;
using OmniKassa.Model.Response.Notification;

[HttpPost]
public ActionResult Webhook(ApiNotification notification)
{
    MerchantOrderStatusResponse response;
    do
    {
        response = omniKassa.RetrieveAnnouncement(notification);
        foreach(MerchantOrderResult result in response.OrderResults)
        {
            // Update the order status using the properties in result
        }
    }
    while (response.MoreOrderResultsAvailable);

    return new HttpStatusCodeResult(HttpStatusCode.OK);
}
```

This step also verifies the signature of the notification. If this goes wrong, for example, a new signing key may have been generated in the Rabo OmniKassa Dashboard and it has not yet been processed by all systems. The Rabo OmniKassa tries it again at a later time.

The MerchantOrderStatusResponse Object returned at the call to the RetrieveAnnouncement method consists of the following properties:

Field	Description
OrderResults	This is an array consisting of 0 or more objects of type MerchantOrderResult. The result of an order is included In each object. The composition of this item is further explained in further detail in this section.
MoreOrderResultsAvailable	When calling RetrieveAnnouncement, the data of up to 100 objects is returned in OrderResults, to keep the volume of the response manageable. If more than 100 orders have been processed, the MoreOrderResultsAvailable field has a value of true and a new call to RetrieveAnnouncement can query the results of the next set of up to 100 orders. This process can be repeated as long as moreOrderResultsAvailable has a value of false.

As shown above, an object of type MerchantOrderResult contains the data of a single order. The following table explains the fields of this item.

Field	Description
MerchantOrderId	The ID of the order as specified by the webshop when the order is announced.
OmniKassaOrderId	The unique identifier assigned by OmniKassa to the order at the time of the announcement.
PoiId	The unique ID that identifies the webshop.
Order Status	The status of the order. The following values are possible: COMPLETED: The full amount of the order is paid by the consumer. CANCELLED: The order was canceled by the consumer. EXPIRED: The order has expired without the consumer having paid.

OrderStatusDateTime	The most current time stamp of the order. If there is no payment attempt, this field will contain the time the order was announced at OmniKassa. Otherwise, this field contains the time of the most recent payment attempt.
Error code	This field is reserved for future use. Currently, this field does not contain a value.
PaidAmount	The amount paid by the consumer. In case of COMPLETED, that is equal to the order amount as given by the webshop at the announcement. When CANCELLED and EXPIRED, this field will have a value of 0.
TotalAmount	The original order amount as specified by the webshop when announcing the order.

6. Request available payment brands

Remark: The functionality described in this chapter is currently only available in the Java SDK. In the near future the other SDKs (PHP and .NET) will be updated to include this functionality as well.

The SDK also provides functionality to look up all payment brands of a web shop as configured in the dashboard of Rabo Omnikassa. This information can be used to determine which payment brands are available to the consumer. It is typically used in conjunction with the paymentBrand and paymentBrandForce fields of an order announcement (see also section 3.1).

Given an `Endpoint` (see section 3.2) we can look up the payment brands as followings.

Java

```
PaymentBrandsReponse response = endpoint.retrievePaymentBrands();
for (PaymentBrandInfo paymentBrand : response.getPaymentBrands()) {
  String name = paymentBrand.getName();
  PaymentBrandStatus status = paymentBrand.getStatus();

  // use the name and status variables for further processing
}
```

As shown above the `retrievePaymentBrands()` method of the `Endpoint` class returns an instance of `PaymentBrandsResponse`. This class provides a method `getPaymentBrands()` that returns a list containing all payment brands that are configured for the web shop. Each element of this list is an object of type `PaymentBrandInfo` containing the details of a single payment brand, as shown in the following table.

Field	Description
name	A string containing the name of the payment brand. This string may contain one of the following values: • IDEAL • PAYPAL • AFTERPAY • MASTERCARD • VISA • BANCONTACT • MAESTRO • V_PAY Note: Keep in mind that other values can be expected as well whenever Rabo Omnikassa is extended with new payment brands.
active	A boolean indicating if the payment brand is active (`true`) or inactive (`false`).

Only the payment brands that are returned in this list and are active can be used for payments.