Rabobank
COO Operations NL

Software Development Kit Rabo OmniKassa 2.0

Handleiding developers versie 1.13

Version:	1.13 NL Juli 2019
Contact e-mail address:	contact@omnikassa.rabobank.nl

Disclaimer: Deze Handleiding is uitsluitend bedoeld voor derden die voor en in opdracht van klanten zorgdragen voor de technische koppeling van Rabo OmniKassa met de webshop van de klant. Rabobank is niet aansprakelijk voor mogelijke schade die voortvloeit uit fouten in of verkeerd gebruik van de Handleiding. Bijvoorbeeld maar niet beperkt tot schade die ontstaat doordat de technische koppeling van Rabo OmniKassa met de webshop van de klant niet (geheel) correct werkt. Rabobank heeft het recht deze Handleiding te wijzigen.

.

Inhoudsopgave

1. Vereisten en installatie instructies

1.1. PHP SDK

Voor de PHP SDK dient u de beschikking te hebben over PHP versie 7.1 of hoger. Om de PHP SDK te gebruiken pakt u de ZIP file uit in een directory op het systeem waarop de webwinkel geïnstalleerd is en wel zodanig dat de bestanden van de PHP SDK bereikbaar zijn vanuit de PHP bestanden van de webwinkel. Include vervolgens het door Composer gegenereerde autoloader.php bestand in uw code middels:

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

1.2. Java SDK

Indien u gebruik maakt van de Java SDK dient u de beschikking te hebben over Java Runtime Environment versie 1.8 of 11. Om de Java SDK te gebruiken pakt u de ZIP file uit op het systeem waarop de webwinkel geïnstalleerd is. De feitelijke SDK is een enkele jar en bevindt zich in de sdk subdirectory. De jar files waarvan deze SDK afhankelijk is staan in de lib subdirectory. Zowel de SDK jar file als de afhankelijke jar files dienen allen in het classpath te worden opgenomen om de Java SDK te kunnen gebruiken.

1.3. .NET SDK

The .NET SDK ondersteunt de volgende .NET platformen:

- .NET Framework (vanaf versie 4.5.2)
- .NET Core 1.x / 2.x (vanaf .NET Standard versie 1.3)

Indien van toepassing worden in dit document aparte code voorbeelden gegeven voor deze platformen.

1.4. SSL configuratie

Om een versleutelde verbinding te kunnen opzetten met Rabobank OmniKassa dient de Java Runtime Environment, .NET omgeving of PHP (afhankelijk van de gekozen SDK) geconfigureerd te zijn met een CA bundle waarmee de geldigheid van het certificaat van https://betalen.rabobank.nl kan worden gecontroleerd. In de regel is dit al het geval en hoeft u niets te doen.

Indien u PHP onder Windows draait kan het nodig zijn dat u in het bestand php.ini de property openssl.cafile configureert met het pad naar de CA bundle. Raadpleeg de PHP documentatie voor meer informatie.

2. Introductie

2.1. Doel

Dit document beschrijft hoe een webshop aangesloten kan worden op de Rabo OmniKassa met behulp van de beschikbare SDKs. Op dit moment levert de Rabobank SDKs voor de volgende programmeertalen / platformen: PHP, Java en .NET.

Het is mogelijk dat in deze handleiding al functionaliteit is opgenomen die behoort bij een betaalmethode die nog niet beschikbaar is in het Rabo OmniKassa dashboard, dit is voor de goede werking van de SDK niet bezwaarlijk. In het Rabo OmniKassa dashboard kunnen de beschikbare betaalmethoden geselecteerd worden.

2.2. Signeersleutels en refresh tokens

. In de code wordt via de aanduidingen {refresh_token} en {signing_key} verwezen naar respectievelijk het refresh token en signeersleutel. Deze aanduidingen dient u niet letterlijk over te nemen maar te vervangen met de waarden die u aantreft in het dashboard van Rabo Omnikassa 2.0.

3. Betaalstappen

Om een globaal beeld te krijgen beschrijft dit hoofdstuk uit welke stappen een betaling via Rabo Omnikassa bestaat. In de resterende hoofdstukken worden elk van deze stappen in meer detail beschreven.

Elke betaling bestaat uit de volgende drie stappen:

1. **Klaarzetten order**
   Alvorens de consument het betaalverzoek kan voldoen kondigt de webwinkel eerst de order bij Rabo Omnikassa aan. In de order worden alle gegevens opgenomen die Rabo Omnikassa nodig heeft om de consument door de betaalstappen heen te leiden. BIj een succesvolle orderaankondiging koppelt Rabo Omnikassa een URL terug naar de betaalpagina's.

2. **Consument betaalt de order**
   De webwinkel redirect de consument naar de URL die door Rabo Omnikassa bij bovenstaande stap werd teruggekoppeld. De consument komt terecht op de betaalpagina's van Rabo Omnikassa van waaruit de consument het betaalverzoek voldoet. Wanneer dit gedaan is komt de consument terug bij de webwinkel.

3. **Updates ontvangen over orders**
   In deze stap stuurt Rabo Omnikassa een notificatie naar de webhook URL van de webwinkel wanneer één of meer orders zijn verwerkt. Aan de hand van een uniek token (sleutel) in deze notificatie kan de webwinkel de eindstatus van deze orders opvragen aan Rabo Omnikassa. Deze stap is optioneel en wordt alleen uitgevoerd als de webhook URL geconfigureerd is in het dashboard van Rabo Omnikassa.

In de volgende hoofdstukken beschrijven we hoe deze stappen aan de hand van de SDK worden geïmplementeerd.

3.1. Betaalverzoek versturen

3.1.1. Klaarzetten order

Om een betaalverzoek te doen moet er eerst een order klaargezet worden. In de order staat alle informatie over het betaalverzoek die de Rabo OmniKassa nodig heeft om de consument door de betaalstappen heen te leiden. In de volgende codeblokken staan voorbeelden om een order aan te maken. Aan de order zijn eisen gesteld en wanneer deze hier niet aan voldoet zijn bepaalde betaalmethodes niet beschikbaar, wordt de order opgeschoond of afgekeurd.

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. Veldbeschrijving

Hieronder staan alle velden beschreven met de naam, een omschrijving, en de regels waaraan het veld moet voldoen.

MerchantOrder

Veld	Omschrijving	Regels
`merchantOrderId`	De identiteit van de order	Verplicht.
		Mag alleen bestaan uit alfanumerieke karakters
		Dit veld wordt ingekort tot maximaal 24 karakters.
		Voor AfterPay moet dit veld uniek zijn
`description`	Een omschrijving van de order	Optioneel
		Heeft een maximale lengte van 35 karakters
`orderItems`	Alle items die besteld gaan worden door de consument	Optioneel
		Er mogen maximaal 99 items meegeleverd worden

Veld	Omschrijving	Regels
`amount`	Het totale orderbedrag in centen, inclusief BTW	Verplicht.
		Het bedrag mag niet hoger zijn dan 99.999,99
		Het bedrag moet gelijk zijn aan de som over alle order items van de stukprijs (inclusief BTW) vermenigvuldigd met het aantal exemplaren. Als gevolg van de manier waarop de BTW berekend wordt kan het door afrondingsverschillen voorkomen dat deze niet gelijk zijn. We adviseren daarom om het totale BTW bedrag te baseren op de BTW van de stukprijs in plaats van het totale orderbedrag exclusief BTW. Bijvoorbeeld: veronderstel dat de stukprijs van een order item (exclusief BTW) € 12,98 bedraagt en een BTW tarief van 21% gehanteerd wordt. Wanneer een consument 7 exemplaren bestelt dan is de stukprijs inclusief BTW € 12,98 + 21% = € 15,71 afgerond. Het totale orderbedrag (inclusief BTW) dat in dit veld wordt meegegeven is dan 7 x € 15,71 = € 109,97. **Let op**: Als het bedrag niet gelijk is aan de som van de bedragen van de order items dan 1. worden de order items uit de orderaankondiging gefilterd, en 2. is AfterPay niet als betaalmethode mogelijk
`shippingDetails`	Het afleveringsadres van deze order	Optioneel
`language`	De teksten op de betaalpagina's zullen in deze taal zijn	Optioneel, standaard hanteren we NL.

Veld	Omschrijving	Regels
		ISO 3166-1 alpha-2, beperkt tot NL, EN, FR en DE.
		Als NL, EN, FR, of DE meegeleverd wordt: dan hanteren we NL, EN, FR, of DE respectievelijk
		Als een onbekende waarde wordt meegeleverd: dan hanteren we EN
`merchantReturnURL`	De URL waar de consument naar terug zal keren nadat de betaalstappen afgerond zijn	Mag niet `null` of leeg zijn
		Moet een valide `URL` zijn
`paymentBrand`	De betaalmethode waar de consument tot gelimiteerd wordt	Optioneel
		Moet één van de volgende waardes zijn: IDEAL, AFTERPAY, PAYPAL, MASTERCARD, VISA, BANCONTACT, MAESTRO, V_PAY Wanneer waarde `CARDS` meegeleverd wordt, dan worden alle kaart betaalmethoden aangeboden (MasterCard, Visa, Bancontact, Maestro, en V PAY)
`paymentBrandForce`	Manier waarop de betaalmethode geforceerd moet worden	Verplicht indien `paymentBrand` meegegeven is, anders optioneel
		Moet één van de volgende waardes zijn: `FORCE_ONCE` of `FORCE_ALWAYS`
`customerInformation`	Een beperkte set aan informatie van de consument	Optioneel
`billingDetails`	Het factuuradres van deze order	Optioneel

Veld	Omschrijving	Regels
`initiatingParty`	Een uniek ID dat de partij identificeert van waaruit de orderaankondiging afkomst is.	Optioneel. Dit veld dient leeg te blijven tenzij met de Rabobank anders is afgesproken.

De betaalmethode en de forceer opties werken als volgt. Wanneer de betaalmethode op `IDEAL` staat, en de forceer optie `FORCE_ONCE` is gekozen, dan betekent dit voor de consument dat zij bij aankomst op de Rabobank OmniKassa, gelijk een iDEAL betaling start en dus op het bank keuzen scherm aankomt. Zij heeft dan de mogelijkheid om de betaling af te ronden **of** voor een andere betaalmethode te kiezen door te klikken op `Kies andere betaalmethode`. Bij de optie `FORCE_ALWAYS` is het niet mogelijk voor de consument om een andere betaalmethode te kiezen en zijn de enige opties om het betaalverzoek goed te keuren of annuleren.

Money

Veld	Omschrijving	Regels
`currency`	De valuta	Mag niet `null` zijn
		Moet `EUR` zijn
		Andere valuta worden niet ondersteund
`amount`	Het bedrag	Mag niet `null` zijn
		Moet een natuurlijk getal zijn

Om een `Money` instantie aan te maken kan gebruik gemaakt worden van de volgende 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

Veld	Omschrijving	Regels
`firstName`	Voornaam van de consument	Mag niet `null` of leeg zijn
		Heeft een maximale lengte van `50` karakters
`middleName`	Tussenvoegsel van de consument	Optioneel
		Heeft een maximale lengte van `20` karakters
`lastName`	Achternaam van de consument	Mag niet `null` of leeg zijn
		Heeft een maximale lengte van `50` karakters
`street`	Straatnaam van het adres	Mag niet `null` of leeg zijn
		Heeft een maximale lengte van `100` karakters. **Let op**: In het geval van card betaling bij Worldline wordt dit veld ingekort naar maximaal 50 karakters.
`postalCode`	Postcode van het adres	Mag niet `null` of leeg zijn
		Moet overeenkomen met het postcode formaat* van de gekozen `countryCode`
`city`	Stad van het adres	Mag niet `null` of leeg zijn
		Heeft een maximale lengte van `40` karakters

Veld	Omschrijving	Regels
`countryCode`	Landcode van het adres	Mag niet `null` of leeg zijn
		Moet één van de volgende waardes zijn: `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, BV, 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, JE, 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`	Huisnummer van het adres	Optioneel
		Heeft een maximale lengte van 100 karakters. **Let op**: In het geval van card betaling bij Worldline wordt dit veld ingekort naar maximaal 50 karakters.
`houseNumberAddition`	Huisnummer toevoeging van het adres	Optioneel
		Heeft een maximale lengte van 6 karakters

De ondersteunde postcode formaten zijn als volgt. De overige landcodes hebben alleen een maximale lengte van 10 en staan in onderstaande tabel aangegeven als 'Anders'.

Landcode	Formaat	Maximale lengte
`BE`	`\p{Digit}+`	4
`DE`	`\p{Digit}+`	5
`NL`	`\p{Digit}{4}\p{Alpha}{2}`	6
Other	N.v.t.	10

CustomerInformation regels

Veld	Omschrijving	Regels
`emailAddress`	Het email adres van de consument	Optioneel
		Moet een valide email adres zijn
		Heeft een maximale lengte van 45 karakters
`dateOfBirth`	De geboortedatum van de consument	Optioneel
		Moet in het formaat: DD-MM-YYYY
`gender`	Het geslacht van de consument	Optioneel
		Moet één van de volgende waardes zijn: `M, F`
`initials`	De initialen van de consument	Optioneel
		Mag alleen bestaan uit alfabet karakters
		Heeft een maximale lengte van 256 karakters
`telephoneNumber`	Het telefoonnummer van de consument	Optioneel
		Mag bestaan uit cijfers en alfabet karakters
		Heeft een maximale lengte van 31 karakters

Bij betaalmethode AfterPay toont Rabobank Omnikassa na de orderaankondiging een pagina aan de consument waarin hij bovenstaande velden kan aanvullen of wijzigen.

Uw webshop wordt **niet** op de hoogte gebracht van de gegevens die tijdens dit proces ingevuld worden.

OrderItem Regels

Veld	Omschrijving	Regels	Toelichting
`id`	Het id van het item	Optioneel	
		Heeft een maximale lengte van `25` karakters	
`name`	Naam van het item	Mag niet `null` of leeg zijn	
		Mag alleen bestaan uit alfabet karakters	
		Heeft een maximale lengte van `50` karakters	
`description`	Een omschrijving van het item	Optioneel	
		Heeft een maximale lengte van `100` karakters	
`quantity`	Aantal exemplaren	Mag niet `null` of leeg zijn	
		Moet een natuurlijk getal zijn groter dan 0	
`amount`	Het bedrag per stuk in centen, inclusief BTW	Mag niet `null` zijn	
`tax`	BTW per stuk in centen	Optioneel	
`category`	Categorie van dit item	Mag niet `null` zijn	
		Moet één van de volgende waardes zijn: `DIGITAL`, `PHYSICAL`	
`vatCategory`	BTW categorie van het item	Optioneel	

	Moet één van de volgende waardes zijn: 1, 2, 3, 4	De waarden verwijzen naar de verschillende tarieven die in Nederland worden gehanteerd: 1 = Hoog (momenteel 21%), 2 = Laag (tot en met 31 december 2018 is dit 6%, vanaf 1 januari 2019 is dit 9%), 3 = Nul (0%), en 4 = Geen (vrijgesteld van BTW)

De quantity beschrijft hoeveel de consument van het product wil hebben, bijvoorbeeld 2 appels, of 3 peren. Het amount geeft aan hoeveel één product kost, dus 1 appel kost €1,50, of 1 peer kost €1,75, en is inclusief BTW. Als we de voorbeelden uitschrijven dan wordt het:

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();
```

Korting

Een korting in de order wordt eveneens aangeleverd middels een OrderItem maar waarbij de velden amount en (indien van toepassing) en tax een negatieve waarde hebben. In onderstaande voorbeelden wordt een korting vastgelegd van 10 euro:

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();
```

Het `amount` van de `MerchantOrder` moet, indien de `orderItems` gevuld worden, overeenkomen met de som van de `orderItems`. Als geen `orderItems` meegeleverd worden mag elk bedrag, binnen de opgestelde regels, ingevuld worden. Het totaalbedrag kan als volgt worden berekend:

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);
```

Mocht het zijn dat er toch een incorrecte order wordt opgestuurd en de Rabo OmniKassa het kan repareren, dan wordt er gepoogd om de order toch te corrigeren door gegevens weg te halen of in te korten. Dit is alleen in de uiterste gevallen van toepassing en kan ertoe leiden dat specifieke betaalmethodes niet beschikbaar zijn voor deze order.

Verplichte velden per betaalmethode

Ongeacht de betaalmethode dient in elke order tenminste de merchantOrderId, amount en de merchantReturnUrl te zijn opgenomen. Afhankelijk van de betaalmethode dienen extra gegevens in de order te worden opgenomen. Onderstaande tabel toont welke gegevens dit zijn, uitgesplitst naar betaalmethode:

Betaalmethode	**Aanvullende gegevens in de order**
iDEAL	Voor deze betaalmethode zijn geen aanvullende gegevens nodig.
PayPal	Hoewel niet verplicht adviseren we om ook de order items in de order op te nemen voor extra zekerheid over de betaling van PayPal aan de eigenaar van de webwinkel.
Bancontact VISA MasterCard V PAY Maestro	Hoewel niet verplicht adviseren we om het afleveradres (of indien niet bekend het factuuradres) in de order op te nemen voor extra zekerheid over een succesvolle betaling.
AfterPay	Voor AfterPay zijn de volgende aanvullende gegevens verplicht: • order items met per order item het ID, de description en tax of vatCategory. • factuur- of afleveradres (indien afwijkend dan zijn beide adressen verplicht). Daarnaast vereist AfterPay dat het veld merchantOrderId uniek is. Het orderbedrag dient minimaal 5 euro te zijn.

Indien voor een betaalmethode de verplichte aanvullende gegevens ontbreken in de order dan kan de consument deze methode niet gebruiken om het betaalverzoek te voldoen. Als de betaalmethode middels het veld paymentBrand in de order was opgenomen dan zal de aankondiging worden geweigerd door Rabo Omnikassa.

3.2. Endpoint aanmaken

Naast een order is ook een instantie van een `Endpoint` nodig. Met deze `Endpoint` is het mogelijk om alle aanroepen richting de Rabo OmniKassa uit te voeren. Een `Endpoint` wordt geïnstantieerd met een drietal parameters: een `URL,` een `Base64` geëncodeerde signeersleutel, en een `TokenProvider` implementatie.

De URL bepaalt of tegen de Productie omgeving of tegen de Sandbox omgeving gekoppeld wordt:

Omgeving	Omschrijving
PRODUCTION	In deze omgeving worden alle betalingen afgeschreven van consumenten en overgemaakt naar de eigenaar van de webshop
SANDBOX	Dit is een testomgeving. Betalingen op deze omgeving leiden niet tot af- of bijschrijvingen op uw rekening en zijn slechts bedoeld om de verbindingen met uw webwinkel te testen.

De omgevingen zijn te configureren in het Rabo OmniKassa Dashboard - bijvoorbeeld welke betaalmethodes aangeboden moeten worden.

De signeersleutel, is een geheime sleutel die terug te vinden is in het Rabo OmniDashboard. Deze signeersleutel is Base64 geëncodeerd. De signeersleutel wordt gebruikt om alle berichten van de aanroepen te versleutelen van de Rabo OmniKassa. Dit is een extra beveiligingslaag om aan te tonen dat berichten wel degelijk van de webshop afkomen en niet van eventuele hackers.

Als laatste hebben we een implementatie van de `TokenProvider` nodig. Deze implementatie is verantwoordelijk voor het aanleveren en opslaan van token informatie. Een implementatie kan bijvoorbeeld de gegevens opslaan in een database, op een file systeem, of in het geheugen gezet worden. Het is de bedoeling dat de eigen `TokenProvider` standaard een refresh token aanlevert - deze moet op te halen zijn voordat de eerste aanroep gedaan wordt. Dit token is te vinden in het Rabo OmniKassa Dashboard en is een uniek token, met lange houdbaarheid, die gebruikt wordt om een access token aan te vragen bij de Rabo OmniKassa. Het access token is een token met een korte houdbaarheid waarmee alle aanroepen worden geauthenticeerd. Dit token wordt aangeleverd bij de `TokenProvider` voor opslag en eveneens om uit te lezen.

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);
            }
        }
    }
}
```

Met deze gegevens kan dan als volgt een `Endpoint` aangemaakt worden.

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);
```

Om met de Sandbox omgeving te koppelen dient de constante SANDBOX gebruikt te worden als eerste parameter. Vergeet daarbij niet om ook het refresh token en de signeersleutel te vervangen.

3.3. Versturen order

Met dit `Endpoint` kunnen we de order aankondigen en wordt de URL van de Rabo OmniKassa, waar de gebruiker de order kan betalen, terug gestuurd.

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);
```

Het object van het type MerchantOrderResponse dat door de Java SDK wordt geretourneerd bevat de volgende velden:

Field	Description
redirectUrl	De URL naar de betaalpagina's van Rabo Omnikassa. De gebruiker dient naar deze URL te worden omgeleid.
omnikassaOrderId	Een uniek ID dat Rabo Omnikassa aan de order heeft toegekend. Dit ID is nodig om het resultaat van een betaling zoals uitgewisseld via de webhook aan de juiste order te koppelen (zie ook hoofdstuk 5).

Opmerking: De andere SDKs (PHP and .NET) zullen binnenkort worden aangepast zodat deze ook een MerchantOrderResponse object retourneren met dezelfde velden.

4. Consument betaalt de order

In de Rabo OmniKassa kan de consument de order betalen. Wanneer dit gedaan is komt de consument terug bij de webwinkel via de `merchantReturnUrl` die opgegeven is tijdens het klaarzetten van de order. Met deze `URL` worden een aantal `GET` parameters meegestuurd die over de order gaan namelijk:

Parameter	Omschrijving
order_id	Dit is het bestelnummer van de order.
status	De huidig bekende status van de order.
signature	Met de handtekening is het mogelijk om te controleren dat de huidige aanroep authentiek is.

Om te controleren dat de handtekening authentiek is kan de volgende code gebruikt worden. Het is eveneens aan te raden om gebruik te maken van de `PaymentCompletedResponse` class zodat de `order_id`, `status`, en `signature` geschoond worden als zij invalide / onveilige waardes bevatten.

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;
```

Mocht het antwoord invalide zijn, dan is het raadzaam om de gebruiker door te verwijzen naar een foutpagina maar de order als 'open' te beschouwen. In een later stadium kan door middel van notificaties de daadwerkelijke status van de order opgevraagd worden.

5. Updates ontvangen over orders

Alle status overgangen van een order worden bijgehouden door de Rabo OmniKassa zodat deze aangeboden kunnen worden aan de webwinkel met behulp van notificaties. De Rabo OmniKassa doet dit door een notificatie te sturen naar de `webhookUrl` via een `POST` aanvraag met de notificatie als JSON meegestuurd. De `webhookUrl` kan geconfigureerd worden in het Dashboard van Rabo OmniKassa.

Een notificatie ziet er als volgt uit:

Notification

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

Met deze JSON is het mogelijk om een `AnnouncementResponse` op te bouwen waarmee de daadwerkelijke informatie van de orders opgehaald kan worden.

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);
}
```

Ook bij deze stap wordt de handtekening van de notificatie gecontroleerd. Als dit fout gaat kan het zijn dat bijvoorbeeld een nieuwe signeersleutel gegenereerd is in het Rabo OmniKassa Dashboard en deze nog niet verwerkt is door alle systemen. De Rabo OmniKassa probeert het dan op een later moment nog een keer.

Het MerchantOrderStatusResponse object dat geretourneerd wordt bij de aanroep op de retrieveAnnouncement methode bestaat uit de volgende properties:

Veld	Omschrijving
orderResults	Dit is een array bestaande uit 0 of meer objecten van het type MerchantOrderResult. In elk object is het resultaat van een order opgenomen. De samenstelling van dit object wordt verder in deze paragraaf nader toegelicht.
moreOrderResultsAvailable	Bij de aanroep van retrieveAnnouncement worden de gegevens van maximaal 100 objecten teruggegeven in orderResults, dit om de omvang van de response beheersbaar te houden. Indien er meer als 100 orders verwerkt zijn dan heeft het veld moreOrderResultsAvailable de waarde true hebben en kan met een nieuwe aanroep van retrieveAnnouncement de resultaten van de volgende verzameling van maximaal 100 orders worden opgevraagd. Dit proces kan net zolang herhaald worden totdat moreOrderResultsAvailable de waarde false heeft.

Zoals boven aangegeven bevat een object van het type MerchantOrderResult de gegevens van een enkele order. In onderstaande tabel worden de velden van dit object nader toegelicht.

Veld	Omschrijving
merchantOrderId	Het ID van de order zoals door de webshop is opgegeven bij de aankondiging van de order.
omnikassaOrderId	Het unieke ID dat door Omnikassa aan de order werd toegekend op het moment van de aankondiging.
poiId	Het unieke ID dat de webshop identificeert.
orderStatus	De status van de order. De volgende waarden zijn mogelijk: COMPLETED: het volledige bedrag van de order is betaald door de consument. CANCELLED: de order werd geannuleerd door de consument. EXPIRED: de order is verlopen zonder dat de consument betaald heeft.

orderStatusDateTime	De meest actuele timestamp van de order. Indien er geen betaalpoging is geweest zal dit veld het tijdstip bevatten waarop de order werd aangekondigd bij Omnikassa. Anders bevat dit veld het tijdstip van de meest recente betaalpoging.
errorCode	Dit veld is gereserveerd voor toekomstig gebruik. Op dit moment bevat dit veld geen waarde.
paidAmount	Het bedrag dat door de consument is betaald. In geval van COMPLETED is dat gelijk aan het orderbedrag zoals opgegeven door de webshop bij de aankondiging. Bij CANCELLED en EXPIRED zal dit veld de waarde 0 hebben.
totalAmount	Het originele orderbedrag zoals opgegeven door de webshop bij de aankondiging van de order.

6. Beschikbare betaalmethodes opvragen

Opmerking: de functionaliteit zoal beschreven in dit hoofdstuk is op dit moment alleen beschikbaar in de Java SDK. De andere SDKs (PHP en .NET) zullen binnenkort worden voorzien van dezelfde uitbreiding.

De SDK biedt ook functionaliteit aan om de betaalmethodes op te vragen die voor een webshop in het dashboard van Rabo Omnikassa zijn geconfigureerd. Deze informatie kan gebruikt worden om te bepalen welke betaalmethodes aan de consument worden aangeboden.

Gegeven een `Endpoint` worden de betaalmethodes als volgt opgevraagd.

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
}
```

Zoals bovenstaand voorbeeld laat zien retourneert de `retrievePaymentBrands()` methode van de `Endpoint` class een instantie van `PaymentBrandsResponse`. Deze class heeft een methode `getPaymentBrands()` dat een lijst retourneerd van alle betaalmethodes die voor de webshop geconfigureerd zijn. Elk element van deze lijst is een object van het type `PaymentBrandInfo` en bevat de details van een enkele betaalmethode, zie ook onderstaande tabel.

Field	Description
name	De naam van de payment brand. Deze string kan één van de volgende waarden bevatten: • IDEAL • PAYPAL • AFTERPAY • MASTERCARD • VISA • BANCONTACT • MAESTRO • V_PAY **Opmerking**: Wanneer Rabo Omnikassa in de toekomst wordt uitgebreid met nieuwe betaalmethodes dan kan dit veld ook andere waarden bevatten.
active	Een Boolean dat aangeeft of de betaalmethode actief (`true`) of inactief (`false`) is.

Alleen betaalmethodes die in deze lijst zijn opgenomen en die actief zijn kunnen gebruikt worden voor betalingen.