Integration handbook for POS systems

(verze: 2.0.0) - last change 11. 6. 2021

source: https://qerko.com/pos-documentation/base.pdf

Introduction

Qerko aspires to be an open platform for comfortable payments by the table. Our effort is simplicity for all parties. Therefore we use standard technologies to make the implementation as easy as possible.

This document describes communication protocol between POS systems and Qerko servers. There are additional roles in this document. The first one is a restaurant where your POS is installed. We will call it the restaurant. The second one is a customer in the restaurant. We will call it the customer.

The communication uses HTTPS protocol to query REST API. Data is transferred in JSON format. You can use HTTP Long-Polling or websocket for communication.

Nothing interesting is happening until there is a customer with Qerko application in a restaurant. When a customer reads Qerko's QR code, which is sticked on a table, Qerko application displays a list of items which has been ordered so far. So a fresh list of ordered items is needed. In this particular moment Qerko asks POS for this list. The customer checks his items and then proceeds to the payment.

The payment is validated by Qerko, then by POS, then on payment gateway. If it passes all the validations and money has been successfully transferred, Qerko asks POS for receipt details. When POS sends required details to Qerko, receipt is generated and sent to the customer by email.

We wish you as easy integration as possible. You can get support at mila@qerko.com.

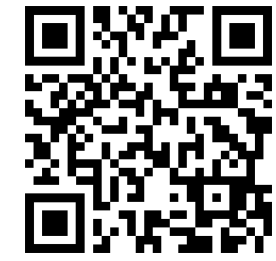
Development environment

API: https://sandbox.qerko.com/api/v2/pos/

Development environment is accessible publicly so nothing stands in your way to start integrating.

Development setup

1. Install Qerko app



2. The app works in production mode by default. To switch the app into development mode, you have to type `qerko-toggle-test` into the Email field on the login screen and press the Continue button. There will appear a `TEST` mark in the top part of the screen which indicates that the app has been switched to the development mode. (You can switch it back to production mode by repeating this procedure.)
3. When the app is successfully switched to development mode, you can register your development account and add test credit card:

Card number: 4125 0100 0100 0208
Expiration: 12/24 (You may enter whatever value you want)
CVC: 992

You may want to visit our payment gateway doc (in czech language) for more options.

Payments are simulated, so feel free to pay anything.

4. Register your development restaurant at https://sandbox.qerko.com/restaurant-admin/
5. In the restaurant administration you need to create an API key. There is a link “API keys admin” in the left column (accessible by the icon in the top left corner on a smartphone).
6. All set:

   You have an API key for POS authorization.
   You have set up the app on your smartphone.
   You have added the credit card into the app.
   You can use any non-empty string as a `Pos-Id` in the development environment.

   Let’s roll.

Development tools

Qerko provides a few goodies which can be used for development and testing purposes.

Demo POS

https://sandbox.qerko.com/demo/ (alias DEMO1)
https://sandbox.qerko.com/demo2/ (alias DEMO2)

This is a simple POS which serves as our testing environment. Feel free to use it to see how things works. DEMO1 uses long-polling and obsolete API v1. DEMO2 uses websocket and current API v2.

API Tester

https://sandbox.qerko.com/api/v2/pos/

This is a simple tool which sends requests to the API. Use it to test raw requests and responses.

Production environment

API: https://qerko.com/api/v2/pos/

Production environment works the same way as the development environment, but payments are not simulated any more. You have to make these steps to enter the production environment:

1. Switch URL to production: https://qerko.com/api/v2/pos/
2. Start using production `Pos-Id`. It will be in GUID v4 format and will be assigned to you by Míly at mila@qerko.com. We recommend to hardcode the `Pos-Id` into your POS or distribute the value with the POS. It is not supposed to be user configurable.

   In Development environment POS can use any non-empty string as stated in Development setup.

3. Switch Qerko app back to production mode by logging out and following the procedure at point 2 in Development setup.

Restaurant's API key

The restaurant's API key is a unique string which serves as an authorization for communication. POS must allow the restaurant to enter the API key. The restaurant generates the API key at Qerko's web administration and then it should be typed into the POS.

Typos in the API key can be checked like this:

```
function verifyApiKey(apiKey :string) :boolean {
	const KEY_LENGTH = 29;
	if (apiKey.length != KEY_LENGTH) {
		return false;
	}

	const HASH_LENGTH = 4
	let base = apiKey.substr(0, apiKey.length - HASH_LENGTH);
	let hash = apiKey.substr(apiKey.length - HASH_LENGTH, HASH_LENGTH);

	return hex(sha1(base)).toLowerCase().startsWith(hash);
}
```

Payment process

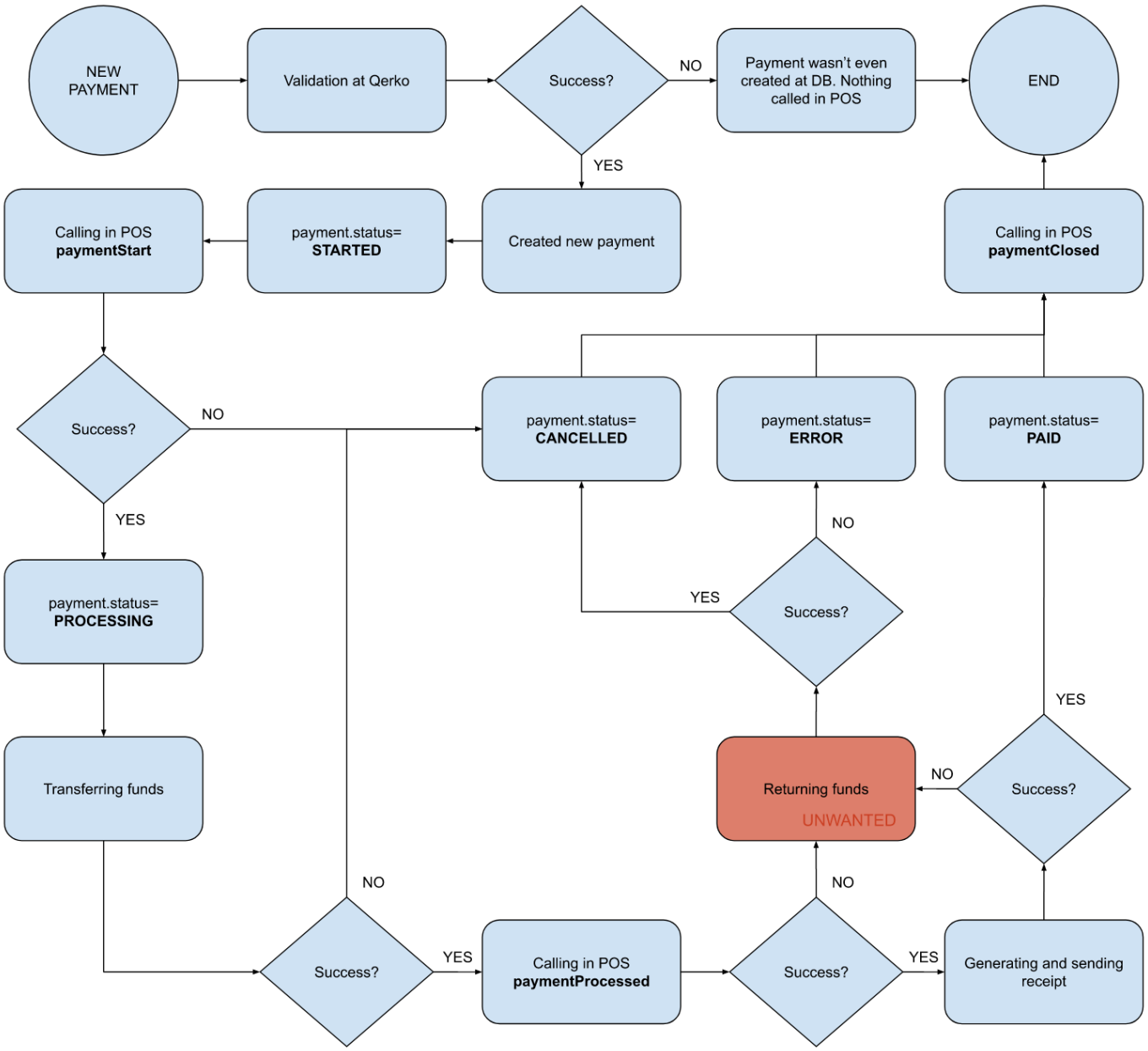
A payment is the most important thing of all. It is validated by Qerko, then by POS then by payment gateway. Any subject can cancel the payment.

1. Validation at Qerko
2. Qerko calls method `paymentStart(...)`. POS validates the payment as described by the method. This is the easiest phase for cancellation. We recommend to lock related items so the restaurant can not manipulate with them. Or at least visually mark them as there is a payment in progress.
   Any error or missing answer cancels the payment on Qerko side.
3. Qerko transfers funds and waits for payment gateway(s) confirmation.
4. Qerko calls method `paymentProcessed(...)`. POS returns to Qerko receipt details.
   Any error or missing answer cancels the payment on Qerko side. If the payment gets cancelled in this step, it takes some time to revoke the funds blocation. Exact revocation time depends on card issuer, bank, etc.
5. Qerko generates the receipt and sends it to the customer by email
6. Qerko calls method `paymentClosed(...)`. The payment has reached its final state. POS checks this final state and unlocks items locked in step 2.

   If the payment was successful, POS sends notification to waiters and makes internal steps to mark items as paid. Method `getTableContents(...)` nor `getBill(...)` must not return paid items any more.

   If the payment was not successful (payment is cancelled), POS must internally cancel the payment and cancel any tax related reports.

Qerko serializes concurrent payments, so there will never be multiple payments in progress on one bill in the same time.



REST API Interface

This section describes a set of endpoints for communicating between POS and Qerko using long-polling and REST API. You can choose to use websocket instead. It is still possible to use these endpoints while using websocket, except the long-polling endpoint.

Endpoints use standard HTTP protocol, version 1.1, using TLS (1.0 or newer). It is required to send SNI.

You can skip over the previous paragraph if you use a reasonably new HTTP client (usually a library).

All data are transferred in JSON format.

HTTP headers

POS must send these headers with every request.

Header name	Required?	Description
Accept-Language	no	Language for error messages. It doesn't influence anything else. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language Default: `"en"`
Authorization	yes	The API key with "Bearer" prefix. Example: `"Bearer abcd-efgh-ijkl-mnop-qrst"`
Content-Type	no	With value `"application/json"`. Required when sending payload to the server.
Pos-Id	yes	Pos-Id assigned to your POS. See Production environment.

GET /api/v2/pos/payment/<idPayment>

This request returns information about payment. This endpoint is provided to resolve various situations when POS missed the `paymentClosed(...)` call. This way POS can retrieve info about payment at any moment.

URL parameter	Description
idPayment	Identifier of payment

Response

- status: **200 OK** – response contains `Payment`
- status: **401 Unauthorized** – invalid API key, or Pos-Id – response contains `Error`
- status: **403 Forbidden** – payment is from another restaurant – response is empty
- status: **404 Not Found** – payment doesn't exist – response is empty

Example

Request

```
GET /api/v2/pos/payment/1234 HTTP/1.1
Host: sandbox.qerko.com
Pos-Id: my-pos-id
Authorization: Bearer my-api-key
User-Agent: MyPos/1.0.0
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
	"additionalData": null,
	"currency": "CZK",
	"discount": null,
	"id": "1234",
	"idBill": "1",
	"idCustomer": "aaf10f7b-730b-4283-9087-965060c6350a",
	"items": [
		{
			"id": "156",
			"minQuantity": "1",
			"name": "Item 5,-",
			"price": "5",
			"quantity": "1",
			"tags": ["Misc"]
		}
	],
	"state": "PAID",
	"parts": [
		{
			"total": "156",
			"provider": {
				"id": "qerko",
				"name": "Qerko",
				"type": "CARD"
			}
		}
	],
	"tipBrutto": "0",
	"tipNetto": "0"
}
```

DELETE /api/v2/pos/payment/<idPayment>

Cancels the payment. Use this request to cancel the payment and return funds to the customer. Limited to 3 hours from payment creation.

URL parameter	Description
idPayment	Identifier of payment

Response

- status: **200 OK** – response contains `Payment`
- status: **401 Unauthorized** – invalid API key, or Pos-Id – response contains `Error`
- status: **403 Forbidden** – payment is from another restaurant – response is empty
- status: **404 Not Found** – payment doesn't exist – response is empty
- status: **410 Gone** – limit for cancellation has expired – response is empty

Example

Request

```
DELETE /api/v2/pos/payment/1234 HTTP/1.1
Host: sandbox.qerko.com
Pos-Id: my-pos-id
Authorization: Bearer my-api-key
User-Agent: MyPos/1.0.0
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
	"additionalData": null,
	"currency": "CZK",
	"discount": null,
	"id": "1234",
	"idBill": "1",
	"idCustomer": "aaf10f7b-730b-4283-9087-965060c6350a",
	"items": [
		{
			"id": "156",
			"minQuantity": "1",
			"name": "Item 5,-",
			"price": "5",
			"quantity": "1",
			"tags": ["Misc"]
		}
	],
	"state": "CANCELLED",
	"parts": [
		{
			"total": "156",
			"provider": {
				"id": "qerko",
				"name": "Qerko",
				"type": "CARD"
			}
		}
	],
	"tipBrutto": "0",
	"tipNetto": "0"
}
```

POST /api/v2/pos/poll

This request provides a way for Qerko to be able to call methods in POS. This technique is called long-polling. You can use websocket instead. Do not use websocket and long-polling at once. Choose only one approach.

This request is held open until Qerko needs to call one of the methods. Qerko responds with info required to call a method. POS internally calls the method and sends the result in another request to this endpoint putting the response into payload.

Payload

None or `MethodCallResponse` if response is available. Response isn't available if you are initiating a connection to Qerko. It is available when you have the return value from a method or you caught an error from a previous `MethodCallRequest`.

Response

- status: **200 OK** – response contains `MethodCallRequest`
- status: **400 Bad Request** – you have sent something wrong – response contains `Error`
- status: **401 Unauthorized** – invalid API key, or Pos-Id – response contains `Error`
- status: **409 Conflict** – server ends connection because another concurrent connection has been open to the same restaurant – response is empty
- status: **503 Service Unavailable** – server is about to be restarted. Try again in 2 minutes – response can contain various things and should be ignored

Example

Request #1

```
POST /api/v2/pos/poll HTTP/1.1
Host: sandbox.qerko.com
Pos-Id: my-pos-id
Authorization: Bearer my-api-key
User-Agent: MyPos/1.0.0

```

Response #1

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
      "uuid": "xyz-1234-5678",
      "method": "getTableContents",
      "args": ["foo-table", "bar-customer"]
}
```

Request #2

```
POST /api/v2/pos/poll HTTP/1.1
Host: sandbox.qerko.com
Pos-Id: my-pos-id
Authorization: Bearer my-api-key
User-Agent: MyPos/1.0.0
Content-Type: application/json; charset=utf-8

{
	"uuid":"xyz-1234-5678",
	"calledMethod": "getTableContents",
	"result": [{
		"id": "the-bill-id",
		"currency": "CZK",
		"items": [{
			"name": "Coca Cola",
			"price": "5",
			"quantity": "1"
		}]
	}]
}
```

Response #2

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
	"uuid": null,
	"method": "noop",
	"args": []
}
```

Websocket Interface

This section describes a protocol for communication between POS and Qerko using a websocket. It is an alternative for use of REST API. It is still possible to receive calls using websocket and use REST API for POS initiated requests (e.g. cancel payment endpoint).

Websockets are used according to RFC-6455 specification. We use websocket-over-TLS for security (handshake is made through HTTPS).

WEBSOCKET wss:// … /api/v2/pos/ws

Using this endpoint POS can open a websocket connection to Qerko. This creates a duplex connection so POS can send messages to Qerko and Qerko can send messages to POS. This request provides a way for Qerko to be able to call methods in POS. Duplex websocket connection is an alternative to long-polling. Do not use websocket and long-polling at once. Choose only one, which fits best to you.

Response HTTP statuses

During opening websocket connection you can receive these http status codes:

- status: **503 Service Unavailable** – server is about to be restarted. Try again in 2 minutes – response can contain various things and should be ignored

Websocket close codes

Websocket connection can be terminated by these socket codes (Don't confuse it with http statuses, see https://developer.mozilla.org/en-US/docs/Web/API/CloseEvent).

- status: **1001 Going Away** – Our server needs to restart the connection. You should try to reconnect immediately. It can happen, for example, when we are releasing an upgrade without downtime.
- status: **4503 Service Unavailable** – server is about to be restarted. Try again in 2 minutes – response can contain various things and should be ignored
- status: **4401 Unauthorized** – invalid API key, or Pos-Id. It can happen when API key is deleted while the connection is active.
- status: **4409 Conflict** – server ends connection because another concurrent connection has been open to the same restaurant – response is empty

Messages

Messages are small data structures which are sent through websocket. They are identified by `type` property.

Authorization request message

This message sends POS to Qerko to authenticate itself. Websockets don't support additional HTTP headers, so we need to authenticate the POS using something else.

Right after opening websocket awaits authorization message. If it receives anything else the websocket will be closed. If no authorization message is received in 5s after opening the connection, it will be closed also.

Send this message right after opening websocket. It will authenticate the connection.

Property	Type	Required?	Description
type	string	Yes	Present in all messages. It defines purpose of the message. Value: `"auth-request"`
posId	string	Yes	Pos-Id assigned to your POS. See Production environment.
apiKey	string	Yes	The API key without any prefix. Example: `"abcd-efgh-ijkl-mnop-qrst"`
locale	string	No	Locale for error localization in ISO 639-1 format. Default: `"en"`

Authorization response message

This message is sent as a response to the authorization message.

Property	Type	Required?	Description
type	string	Yes	Present in all messages. It defines purpose of the message. Value: `"auth-response"`
authorized	boolean	Yes	`TRUE` – You are authorized and you can send and receive other messages. `FALSE` – You are not authorized. Reason is in the message. Qerko closes the socket after this message.

error	Error
	null
		Yes	If not authorized, here is the reason
			If authorized, `null` will be here.
			Example: `{`
			`  message: "Invalid Pos-Id",`
			`  code: "POS_UNKNOWN"`
			`}`

Method call request message

This message can go both ways. It means from Qerko to POS and vice versa.

After receiving this message the receiver executes the corresponding method and sends a Method call response message with a result.

Property	Type	Required?	Description
type	string	Yes	Present in all messages. It defines purpose of the message. Value: `"method-call-request"`
uuid	string null	Yes	Identifier of the call. It can be `null`. If it is `null` receiver doesn't have to send response message. Example: `"1d0de3e1-5d4c-11e9-bc19-064f106d678c"`
method	string	Yes	One of the method names. Example: `"paymentStart"`
args	object[]	Yes	Arguments for the method. Array will match method's signature. Example: `[{ id: "1", total: "255", ...}]`

Method call response message

This message can go both ways. It means from POS to Qerko and vice versa.

This message is sent when there is a result of a method call from a previous Method call request message. Answer is mandatory when a call request contains non-null uuid.

Property	Type	Required?	Description
type	string	Yes	Present in all messages. It defines purpose of the message. Value: `"method-call-response"`
uuid	string	Yes	Identifier of the call. You have received this value in the previous Method call request message. Example: `"1d0de3e1-5d4c-11e9-bc19-064f106d678c"`
result	any	No	It can be whatever the method returns including null. Default: `null` Example: `{ "foo": "bar" }`
error	Error null	No	Error which occured, `null` if there was no error. Default: `null`
calledMethod	string null	No	Method which produced this response. Qerko sends this every time. Provided for easier statelessness. Default: `null`

Methods

Method signatures use TypeScript syntax for documentation purposes.

POS Methods

Below is a list of methods which can be called by Qerko. POS must implement these methods. It is required for both communication techniques (long-polling and websocket). Both techniques call these methods.

getBill(id :string, idCustomer :string) :Bill

Qerko requests the contents of the opened bill. This will be called very often because it is the main Qerko functionality.

Parameter	Type	Description
id	string	Identifier of the bill.
idCustomer	string null	Identifier of the customer, who is at the table. For loyalty program on the POS side.

Special error codes

This section describes string values that can appear in potential `Error.code`, that have special meaning.

- `BILL_CLOSED` – In case Qerko has requested a closed bill. Qerko app will announce to the customer that the bill was closed and there is nothing to pay any more.
- `BILL_NOT_FOUND` – In case Qerko has sent invalid id. Qerko app will take the customer to the bill selection screen.

getTableList() :Table[]

Qerko requests a list of tables which are in the restaurant. All of them. It is used to get data for pairing QR codes and tables on Qerko's web.

getTableContents(idTable :string, idCustomer :string) :Bill[]

Qerko requests a list of opened bills on the table. This will be called when the customer scans Qerko's QR code, so we need to offer a list of bills, because the customer hasn't subscribed to a bill yet. This will be called very often because it is the main Qerko functionality.

Parameter	Type	Description
idTable	string	Identifier of the table, which contents is needed.
idCustomer	string null	Identifier of the customer, who is at the table. For loyalty program on the POS side.

error(error :Error, uuid :string) :null

This method is used to report error, which has occured while processing a response from a method. It is provided mainly for development. Qerko will call it when there was an error while processing a MethodCallResponse. Just write it into a log or process it as you want.

Parameter	Type	Description
error	Error	The error that has occured.
uuid	string null	UUID of the MethodCallResponse, which caused the error. `null` if not relevant or unknown

noop() :null

This is an empty method used to keep the connection between Qerko and POS alive. It is called after 45 seconds of quiet now, but the interval can change in the future. The method itself does nothing, but it is important to break the quiet and send the response with null as a result. Else the connection will be considered as stalled and closed.

pairCustomer(ident :string, idUser :string) :null

This is for pairing POS loyalty cards, vouchers, etc. with a Qerko's user. So POS can automatically offer a discount to the customer. This is a way to pair Qerko's users and POS customer database.

The customer types an identification code from his card into the Qerko app. Qerko calls this method with the typed value in `ident` parameter and Qerko's identifier in the `idUser` parameter.

Parameter	Type	Description
ident	string	Identifier of the customer entered in Qerko application by customer. It must identify customer in POS database. It can be basically whatever the restaurant provide to the customer.
idUser	string	Identifier of the Qerko's user. The same as in `getTableContents(...)`, `getBill(...)` and `Payment`.

Special error codes

This section describes string values that can appear in potential `Error.code`, that have special meaning.

- `IDENT_OCCUPIED` – In case the identifier is already paired with someone else and the POS doesn't allow multiple pairings.
- `IDENT_INVALID` – In case the identifier is invalid. I mean not found, malformed, etc.

paymentStart(payment :Payment) :null

Qerko requests POS to validate new payment. POS must verify the payment especially prices. POS must check that all of the items, which are about to get paid, are still on the bill and that everything is ready to be paid.

Although you can process the tax reports in this phase. It is not very advisable because the customer can be short of funds and you will have to cancel the reports. This can happen quite often and you know the tax office…

We recommend locking related items so the restaurant can not manipulate them. Or at least visually mark them as there is a payment in progress.

Qerko waits 15 seconds for the response. If the response isn't received, the payment will be cancelled.

POS can cancel the payment by throwing any error.

Any error or missing answer will cancel the payment. Feel free to cancel the payment in this phase. This is the best phase to cancel the payment, because if you cancel it in the next phase it will cause bad user-experience due to blocation of the customer's funds.

Parameter	Type	Description
payment	Payment	The payment to validate.

Payment.state MUST be "STARTED" at this moment.

Special error codes

This section describes string values that can appear in potential `Error.code`, that have special meaning.

- `BILL_LOCKED` – Qerko app will announce that there is a waiter operating with the bill right now and suggest to the customer to try the payment afterwards.
- `INVALID_DATA` – In case Qerko has sent invalid data (e.g. unknown bill, unknown item, price mismatch, etc.). Qerko app will try to refresh data and eventually try again.

paymentProcessed(idPayment :string) :Receipt

Qerko informs POS about finished payment and requests POS to generate info for receipt, this includes any tax related reports, etc.

If the payment fails in `paymentStart(...)` or on the payment gateway, this method can be skipped and `paymentClosed(...)` will be called without calling this method.

Qerko waits 15 seconds for the response. If the response isn't received, the payment will be cancelled.

POS can cancel the payment by throwing any error.

Any error or missing answer will cancel the payment. The customer's funds have already been blocked in this phase. So it is not very user-friendly to cancel the payment now because it can take a few hours to revoke the blocation, depending on the bank, card issuer, etc.

Parameter	Type	Description
idPayment	string	Identifier of payment. Corresponds to `Payment.id`.

paymentClosed(idPayment :string, state :string) :null

Qerko announces to POS that the payment has reached its final state and it won't change in the future. It is mandatory to check the state. If it has any other value than `"PAID"` the payment is cancelled.

POS can request cancellation, by REST endpoint or by calling `cancelPayment(...)`.

We recommend printing a short summary for the payment, display notification, play a sound, send notification to mobile waiters. So that the restaurant personnel know about the payment as soon as possible. It is recommended to show the related table, items summary, tip and a hint whether the bill is paid partially or completely.

Do not forget to unlock related items, which have been locked in the `paymentStart(...)` phase.

Parameter	Type	Description
idPayment	string	Identifier of payment. Corresponds to `Payment.id`.
state	string	The final state of the payment. Corresponds to `Payment.state`.

Qerko Methods

Below is a list of methods which can be called by POS using websocket. If you are not using websocket, there is an alternative in REST API.

cancelPayment(idPayment :string) :null

Cancels the payment. Be advised to specify `uuid` in the Method call request message so you can get response in case of error. Limited to 3 hours from payment creation.

Parameter	Type	Description
idPayment	string	Identifier of payment. Corresponds to `Payment.id`.

error(error :Error, uuid :string) :null

This method is used to report error, which has occured while processing response from a method. POS should call it when there was an error while processing a MethodCallResponse. This way you can alert Qerko team, that there might be something wrong.

Parameter	Type	Description
error	Error	The error that has occured.
uuid	string null	UUID of the MethodCallResponse, which caused the error. `null` if not relevant or unknown

getPayment(idPayment :string) :Payment

Returns details of a payment. This method is provided to resolve various situations when POS missed the `paymentClosed(...)` call. This way POS can retrieve info about payment at any moment.

Parameter	Type	Description
idPayment	string	Identifier of payment. Corresponds to `Payment.id`.

Data structures

Follows a list of data structures, which are used for data exchange between POS and Qerko. We provide JSON schemas for comfortable validation.

Bill

POS sends this structure when Qerko needs info about a bill. It carries a meta data and contents of the bill.

Property	Type	Required	Description
additionalData	any	No	POS can use this for any additional data. Qerko will not modify it and you can use it in the payment process. You will receive it in `Payment.additionalData`.
created	string	No	Timestamp of the bill creation in ISO 8601 extended format. Example: `"2019-04-13T00:22:57+01:00"`
currency	string	Yes	Used currency in ISO 4217 format. All prices in this structure must use the same currency. Example: `"CZK"`
denyDiscounts	boolean	No	Whether Qerko can apply another discount. Set it to `TRUE` when there already is an internally applied discount by POS and other discounts are unwanted. This is related to discounts provided by restaurant owners only. Discounts paid by Qerko or third parties (sponsors like breweries) aren't affected by this property. `FALSE` – Qerko can apply a discount. `TRUE` – Qerko can't apply a discount.
allowTip	boolean	No	Can the customer add a tip? Default: true
allowPartialPayment	boolean	No	Can the customer pay only a subset of the items? Default: true
discountOffer	BillDiscountOffer	No	Discount offered to the guest by the restaurant owner. Qerko has many sources of discounts, so Qerko's list will be extended by this offer.
id	string	Yes	Unique identifier of the bill from POS. This identifier is used in payment structure when Qerko returns the whole or part of the bill to be paid.
items	BillItem[]	Yes	Unpaid items on the bill.
name	string	No	Name of the bill to be shown to the customer. If not provided, Qerko will try to generate some.

BillDiscountOffer

POS sends this structure as a part of `Bill` structure. It carries info about a discount offer. It is used as a discount offer from POS, specifically from the restaurant. The customer will probably accept it, unless he has a better offer from another source (loyalty program, Qerko, etc.).

Property	Type	Required	Description
additionalData	any	No	POS can use this for any additional data. Qerko will not modify it and you can use it in the payment process. If the customer decides to accept the offer, you will receive it in `Payment.discount.additionalData`.
name	string	Yes	Human readable name. It will be visible on the receipt and in a Qerko app.
percent	string	Yes	Discount value percentage. It is not possible to offer other types of discounts, because of difficulties with paying only a part of the bill. Example: `"5"` means 5% discount, `"7.5"` means 7.5% discount

BillItem

POS sends this structure as a part of `Bill` structure and receives it as a part of `Payment` structure. It carries info about an item on a bill or item related to a payment.

Property	Type	Required	Description
additionalData	any	No	POS can use this for any additional data. Qerko will not modify it and you can use it in the payment process. You will receive it in `Payment.items[i].additionalData`.
id	string	no	POS identifier of bill item. This value is just returned by Qerko when the same bill item is used (fully or partially). In other words POS sends it in bill structure and Qerko returns it in payment structure.
minQuantity	string	no	Minimum payable quantity. It defines quantity precision also. Qerko will allow only multiples of `minQuantity` as splitted `quantity`. If `minQuantity = 0`, then the item cannot be split. If `quantity` is divisible by `minQuantity` then Qerko will make it possible to split the item, else the item will be passed through Qerko with unchanged `quantity`. Example: If `minQuantity = 1` and `quantity` is integer, then the item can be split into lower integer quantities. If `minQuantity = 1` and `quantity` is decimal, then the item cannot be split. Example: `"0.001"`, `"1"`,... Default: `"1"`
name	string	Yes	Human readable name of bill item. It will be visible on the receipt and in a Qerko app. Example: `"The Two-Hand Burger"`
price	string	Yes	Total price for bill items including taxes. Price can be zero, or negative. Example: `"110.00"`
quantity	string	Yes	Sold amount. It can be decimal and negative. Example: `"5"` (5✕ beer), `"0.7"` (baby portion)
tags	string[]	No	Something identifying groups of products or categories of products, etc. categorization. Can be used for loyalty programs, discounts or something else. Used values depend on the relation between POS and Qerko configuration. Used in discount definition, ex: “10% off for coffee if a tag:dessert is ordered as well” Example: `["burger", "beef", "meat"]`

Discount

This structure is used as part of `Payment` structure. It carries info about a discount that was applied to the payment. Splitting the discount among bill items is completely up to POS. POS must include the discount into `Receipt.taxInfo`.

Property	Type	Required	Description
additionalData	any	No	If `source=pos` this property will contain whatever data POS has sent in BillDiscountOffer structure.
description	string	Yes	Human readable text for the receipt
source	string	Yes	The source of the discount: `POS` – Discount from POS, sent in `Bill.discountOffer` `QERKO` – Discount from Qerko, Qerko will pay the costs `PROMO` – Discount was set up in web administration `LOYALTY` – Discount was set up in web administration as a part of a loyalty program.
value	string	Yes	The absolute value of the discount. Currency is from the payment. Example: `"100"` when the discount was one hundred

Error

This structure is used in various situations to report an error.

Property	Type	Required	Description
message	string	Yes	Human readable text describing the error. It is never presented to the customer.
code	string null	No	When the error occurs in Qerko server it contains one of Error codes or `null`. When the error occurs in POS, it can have a special effect, which is defined by the respective method.

FiscalInfo

POS sends this structure as a part of the `Receipt` structure. It carries info about a fiscal report related to the receipt. This object is treated as template data. Feel free to add any relevant info that the restaurant might want to print on the receipt. It will be possible to have a per-restaurant template in the future. If you implement for a country which is not documented here, send us your requirements, we will make them the standard for the country in this document.

Property	Type	Required	Description
footerLines	string[]	No	Additional lines to print on the receipt below eet* parameters.
footerText	string	No	Additional text to print on the receipt below footerLines.
headerLines	string[]	No	Additional lines to print on the receipt above headerText.
headerText	string	No	Additional lines to print on the receipt above eet* parameters.
eetBkp	string	No	Czech EET BKP
eetPkp	string	No	Czech EET PKP
eetFik	string	No	Czech EET FIK
eetModeLine	string	No	Czech EET Mode, example: `"EET Režim: běžný"`

MethodCallRequest

POS receives this structure, although it is called request, in a response from the long-polling endpoint. It requests the POS to call a method.

Property	Type	Required	Description
uuid	string	No	Identifier of the method call. If not provided, then the sender does not expect the receiver to send a response.
method	string	Yes	The name of the method to call. See Methods.
args	any[]	Yes	Array of method arguments.

MethodCallResponse

POS sends this structure, in a request to the long-polling endpoint, although it is called response. It carries the result of a method call. It is sent as a reaction to receiving a `MethodCallRequest`.

Property	Type	Required	Description
uuid	string	Yes	Identifier of the method call which was in the `MethodCallRequest`.
error	Error	No	Error which has occured, if any.
result	any	No	Result of the corresponding method call.
calledMethod	string null	No	Method which produced this response. Qerko sends this every time. POS can ignore it. Provided for easier statelessness.

Payment

POS receives this structure for `paymentStart(...)` method or as a response to `getPayment(...)` or from the payment endpoint.

Total price is: `sum(items[i].price) + tipBrutto - discount.value`

Property	Type	Required	Description
additionalData	any	No	Additional data from the `Bill` structure.
currency	string	Yes	Currency code in ISO 4217 format.
discount	Discount	No	Discount applied to the payment.
id	string	Yes	Identifier of the payment. Qerko generates it for every
idBill	string	Yes	Identifier of the related bill.
idCustomer	string	Yes	Unique identifier of the customer who pays. For the loyalty program on the POS side. POS can identify the user when there was a previous call to method `pairCustomer`. Else the POS can track users activity, but the user remains anonymous.
items	BillItem[]	Yes	List of items, which are paid by the payment.
state	string	Yes	`STARTED` – The payment process has just started. `PROCESSING` – The payment is processed. `PAID` – The payment is paid. It is the final state. `CANCELLED` – The payment is cancelled. It is the final state. `ERROR` – The payment ended with an error. Manual action must be taken to solve this error and manually switch the transaction to CANCELED.
parts	PaymentPart[]	Yes	This is an array which holds info for splitting the payment among payment providers.
tipBrutto	string	Yes	Tip for the restaurant personnel.
tipNetto	string	Yes	Qerko might take a tip commission, so there will be a tip after the commission subtraction. This will be based on the agreement with the restaurant. POS should show this value to the restaurant personnel.

PaymentPart

POS receives this structure as a part of `Payment` structure. It holds information about a part of payment which has been paid using a payment provider.

Property	Type	Required	Description
provider	PaymentProvider	Yes	This is the payment provider that was used to fulfill this payment part.
total	string	Yes	This is how much was paid using this payment provider.

PaymentProvider

POS receives this structure as a part of `PaymentPart` structure. It holds information about used payment provider. Each provider can be enabled/disabled per restaurant on the Qerko web page. There might be different rules for different types according to local laws.

Property	Type	Required	Description
id	string	Yes	A unique identifier of this payment provider.
name	string	Yes	A human readable provider name Example: “Qerko”, “A meal ticket, Ltd.”, “Some smart card, Ltd.”, etc.
type	string	Yes	This is for description of the type of the transaction, so POS can apply correct rules according to local laws. `CARD` - This means online payment using card `BANK_TRANSFER` - This means payment by bank transfer `INVOICE` - This means exchange of funds based on an invoice `VOUCHER` - This includes vouchers, coupons and other pre-paid types. `MEAL_TICKET` - This means payment by meal tickets

Receipt

POS sends this structure as a response to the `paymentProcessed(...)` method. It carries info for generating the receipt. This object is treated as template data. Feel free to add any relevant info that the restaurant might want to print on the receipt. It will be possible to have a per-restaurant template in the future.

Property	Type	Required	Description
receiptInfo	ReceiptInfo	No	Object with properties containing information about the bill or the receipt.
items	ReceiptItem[]	Yes	Items printed on the receipt. If relevant, include the `Payment.tipBrutto`, or any discount. The total amount on the receipt will be the sum of items prices.
taxInfo	TaxInfo[]	Yes	Calculated tax. Tax info is verified. Sum of all `TaxInfo.base` plus sum of all `TaxInfo.vat` must be equal to related `Payment.total`. If not equal, the related payment is cancelled.
fiscalInfo	FiscalInfo[]	No	Fiscal reports such as czech EET and others

ReceiptInfo

POS sends this structure as a part of the `Receipt` structure. It carries general info about the receipt. This object is treated as template data. Feel free to add any relevant info that the restaurant might want to print on the receipt. It will be possible to have a per-restaurant template in the future.

Property	Type	Required	Description
billName	string	No	The name of the bill
businessPremise	string	No	An identifier of the business premise
cashBox	string	No	An identifier of the cashbox
footerLines	string[]	No	Additional lines to print on the receipt below items.
headerLines	string[]	No	Additional lines to print on the receipt above items.
serial	string	No	Serial number or another identifier
timestamp	string	No	Timestamp in ISO 8601 extended format
waiter	Waiter	No	The waiter who was serving the customer

ReceiptItem

POS sends this structure as a part of the `Receipt` structure. It carries info about an item to be printed on the receipt. This object is treated as template data. Feel free to add any relevant info that the restaurant might want to print on the receipt. It will be possible to have a per-restaurant template in the future.

Property	Type	Required	Description
name	string	Yes	The name of the item
quantity	string	No	The quantity of the item
price	string	Yes	The total price for the item(s)
taxName	string	No	The tax in which is this item included. Related to `TaxInfo.name`. As suggested here: vzorova_uctenka.pdf

Table

POS sends this structure in the list of tables as a response to the `getTableList(...)` method. It carries info about a table.

Property	Type	Required	Description
id	string	Yes	Identifier of the table.
name	string	No	Human readable name of the table. if not provided, `id` will be used instead. Example: `"The table by the door"`, or `"Garden 1"`

TaxInfo

This structure is used as part of `Receipt` structure. It carries info about a tax.

Property	Type	Required	Description
name	string	No	Tax name. Related to `ReceiptItem.taxName`. As suggested here: vzorova_uctenka.pdf Example: `"A"`
rate	string	Yes	Tax rate percentage. Example: `"15"` means 15%, `"12.5"` means 12.5%, etc.
base	string	Yes	Tax base. Example: `"100.00"`
tax	string	Yes	Tax value. Example: `"15.00"`

Waiter

This structure is used as part of `Receipt` structure. It carries info about a waiter, who has served the customer.

Property	Type	Required	Description
id	string	Yes	Identifier of the waiter. Unique in the restaurant.
name	string	Yes	Name of the waiter Example: `"John Doe"`

Error codes

This is a list of possible error codes. Error code is a machine readable identification of the error. This list is not exhaustive and you may encounter a code that is not mentioned.

Error code	Description
`null`	Unknown error.
ACCESS_DENIED	Access to the respective item was denied. Ex: when you try to cancel payment which belongs to another restaurant.
ALREADY_AUTHORIZED	When POS sends auth-request when already authorized
BAD_PARAMETER	A parameter in the incoming payload is invalid.
BAD_PAYLOAD	Server can not parse the incoming payload.
NO_SUCH_METHOD	Received `MethodCallRequest.method` is unknown.
NOT_FOUND	Requested item was not found. Ex: POS sent invalid idPayment to the getPayment(...) method
POS_BLOCKED	Pos-Id is valid, but access is denied. Usually to prevent further damage, because we have identified an error in your implementation.
POS_UNKNOWN	Pos-Id is invalid.
TIME_LIMIT_EXPIRED	When you try to cancel a payment after the allowed time limit.
UNAUTHORIZED	Api-Key is invalid.

Extensions

This is a list of available extensions, which can be implemented by POS. This is optional. Decision to implement these is up to you.

Call the waiter

https://qerko.com/pos-documentation/extension-call-the-waiter.pdf

The customer can call the waiter any time. Like on the plane :)

Direct bills

https://qerko.com/pos-documentation/extension-direct-bills.pdf

Direct bills are mapped to QR codes by POS, not by Qerko, so there is no table between bills and QR codes.

Thank for your patience. If you encounter any trouble contact Mila at mila@qerko.com for help.
Qerko team

It Is Paid, s.r.o. company reserves the right to change.

Changelog

- 11.6.2021
  - Replaced testing credit card due to payment gateway switch
  - Improved documentation of Websocket close codes
- 1.6.2021
  - Fixed some grammar mistakes and typos
- 31.12.2020
  - Updated test credit card
- 4.11.2019
  - Changed the magic keyword for switch to development environment to `qerko-toggle-test`
  - Added examples
  - Added `headerLines`, `headerText`, `footerLines`, `footerText` to `FiscalInfo` structure
- 27.6.2019
  - Added optional Extensions
  - Corrected a few typos
  - Corrected a few mistakes in examples
- 12.6.2019
  - Added payment diagram
  - Added optional `MethodCallResponse.calledMethod`
  - Added missing info about time limit of method cancelPayment(...)
  - Added error codes: ACCESS_DENIED, ALREADY_AUTHORIZED, NO_SUCH_METHOD, NOT_FOUND, TIME_LIMIT_EXPIRED
- 1.6.2019
  - Changed `Error.reason` to `Error.code`
  - Added list of Error Codes
  - Removed `Bill.roundingMode`
  - Added `Bill.additionalData`
  - `BillItem.tags` are no longer required
  - `BillItem.minQuantity` default value is 1
  - Removed `BillDiscountOffer.roundingMode`
  - Renamed `BillDiscountOffer.description` to `BillDiscountOffer.name`
  - Added `Payment.additionalData`
  - Renamed `Payment.tip` to `Payment.tipBrutto`
  - Removed `Payment.total`
  - Renamed `Payment.idUser` to `Payment.idCustomer`
  - Added `Payment.state` STARTED
  - Added `PaymentProvider.id`
  - Removed `Receipt.additionalData`
  - Added `Receipt.receiptInfo` and structure `ReceiptInfo`
  - Moved `Receipt.footerLines` to changes in `ReceiptInfo.footerLines`
  - Moved `Receipt.headerLines` to changes in `ReceiptInfo.headerLines`
  - Moved `Receipt.waiter` in `ReceiptInfo.waiter`
  - `Receipt.items` are required now
  - Added `Receipt.fiscalInfo` and structure `FiscalInfo`
- 24.5.2019
  - Removed `BillItem.splittable`
  - Added `BillItem.minQuantity`
  - Added `Bill.roundingMode`
  - Added `BillDiscountOffer.roundingMode`
  - Added `Payment.tipNetto`
- 17.5.2019
  - Renamed `BillItem.amount` to `BillItem.quantity`
  - Added optional property `Receipt.items`
  - Minor changes in descriptions
- 13.5.2019
  - Added `locale` into Authorization request message
- 9.5.2019
  - `Payment.paymentProvider` was replaced by `Payment.parts`

        - Added `PaymentPart` structure
        - Added `BillItem.amount`
        - Added `BillItem.id`
        - Added `BillItem.splittable`
        - Changed `BillItem.price`
- 24.4.2019
    - Added `headerLines` a `footerLines` to the `Receipt` structure
    - Removed `lines` form the `Receipt` structure
- 23.4.2019
    - Allow to skip nullable properties in data structures
    - Renamed `pairUser(...)` to `pairCustomer(...)`
- 19.4.2019
    - Added `Bill.denyDiscounts`
        - This can prevent Qerko from applying a discount
- 18. 4. 2019
    - It Is Paid is renamed to Qerko
    - This document is in English now
    - Introducing API v2.
        - Added websocket support.
        - Added `Bill` structure.
            - This brings support for multiple bills on a table.
        - Added `BillDiscountOffer` structure.
            - This brings support for POS discount offers.
        - Added `BillItem` structure.
            - This brings support for `additionalData` that can be used by POS.
            - This brings the all new `tags` set for item categorization or recognition.
        - Added `Discount` structure.
            - Successor of the `Sale` structure.
            - Support for various discount sources.
        - Added endpoint for cancelling the payment.
            - For the ability to cancel the payment in case of a customer issue.
        - Added `getBill(...)` method.
            - For the ability to get contents of exact bill.
        - Added `pairUser(...)` method.
            - For the ability to pair Qerko's users and POS loyalty DB.
        - Added `TaxInfo` structure.
            - Taxes are completely calculated by POS now.
        - Added `Receipt` structure.
            - Successor of `BillInfo`.
            - Many changes due international differences.
        - Added `MethodCallRequest`.
            - Just for convenience. This has introduced no functional changes.
        - Added `MethodCallResponse`.
            - Type of `MethodCallResponse.error` has changed.
        - Changed method `error(...)`.
            - Error is be reported using Error structure, not just string.
        - Changed method `getTableContents(...)`.
            - Added `idUser` argument, so POS can offer discounts.
        - Changed methods `paymentProcessed(...)` and `paymentClosed(...)`.
            - To prevent sending the same data more than once and the validation related with it.
        - Structure `Payment`.
            - Added `Payment.paymentProvider`
            - Removed `Payment.idTable`.
                - Replaced by `Payment.idBill`.
            - Removed `Payment.totalItems`.
                - It was confusing and useless.
            - Removed `Payment.timestamp`.
                - It was useless.
        - Removed `BillInfo` structure.

    - Replaced by `Receipt`.
- Removed `Sale` structure.
    - Replaced by `Discount`.
- Removed `TableContents` structure.
    - Replaced by `Bill`.
- Removed `TableItem` structure.
    - Replaced by `BillItem`.