QOR Kiosk API — activating and loading QOR cards at a kiosk

Version 1.4 — 2026-11-11

This guide is for the kiosk operator's developers. It covers everything a kiosk needs to **activate** a QOR card from its stock, **load** money onto it, and **lock** it when the customer reports it lost or stolen, using the **card reference number** (CRN) printed on the card.

---

1. How it works

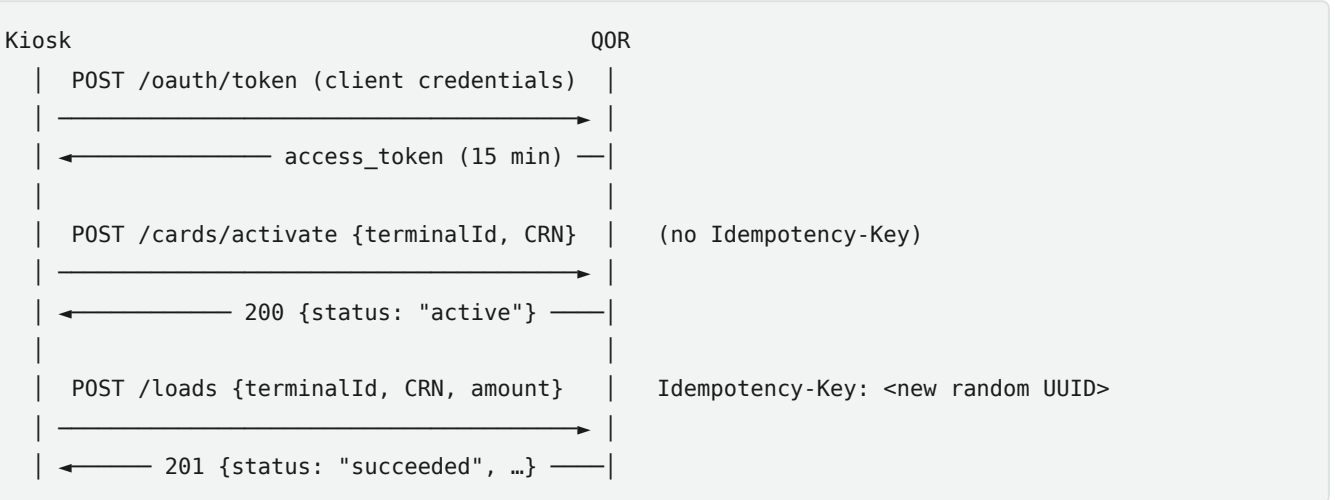
Each kiosk location receives a stock of QOR cards. The cards come **inactive, with a $0 balance**.

When a customer redeems a card at the kiosk:

1. The kiosk gets an access token (once every 15 minutes, not per sale).
2. The kiosk reads the **CRN** on the card.
3. **Activate**: `POST /cards/activate` with the CRN. This is the sale: the card leaves the location's stock and becomes active.
4. **Load**: `POST /loads` with the CRN and the amount the customer paid.
5. The kiosk dispenses the card and prints the receipt.

The customer can then check the balance and transactions at the QOR balance site using the CRN.

If the customer later reports the card lost or stolen, the kiosk can **lock** it: `POST /cards/lock` with the CRN (section 4.5).



A kiosk can only reach cards from **its own location's stock**. Any other card (another location's, unknown, closed) gets the same `CARD_NOT_ELIGIBLE` error.

---

2. Environments

Environment	Base URL
Sandbox	`https://kiosk.sandbox.qorteams.com/kiosk/v1`
Production	Provided at go-live

QOR gives each kiosk, through a secure channel:

Item	What it is
`client_id`	Identifies the kiosk, e.g. `ol_kiosk_01J…`
`client_secret`	Shown once. Store it in the OS credential store (e.g. Windows Credential Manager / DPAPI), never in a plain file or log.
`terminalId`	**Your** machine id for this kiosk. You tell QOR the id, QOR registers it on the kiosk. Every sale call must send it.

One set of credentials per kiosk. Never share credentials between kiosks.

Secret rotation: when QOR issues a new secret, the old one keeps working for 24 hours so you can update the machine.

---

3. Conventions

- **HTTPS only**, TLS 1.2 or later.
- Request and response bodies are JSON (`Content-Type: application/json`), except the token request (section 4.1).
- **Money** is a string with two decimals, no thousands separator: `"25.00"`. Currency is ISO 4217: `"USD"`.
- **Success** responses: `{ "data": { … } }`.
- **Error** responses:

```
{
  "error": {
    "code": "CARD_NOT_ELIGIBLE",
    "message": "This card cannot be used here.",
    "retryable": false,
    "traceId": "4f1c2b7e9a0d4c51"
  }
}
```

  Branch on `code`, never on `message` (the wording can change). Log the `traceId` and quote it when you contact QOR support.

3.1 Idempotency-Key (loads only)

Only **`POST /loads`** needs an `Idempotency-Key` header. Activate and lock do not (a card can only be activated or locked once, so sending either twice is always safe — see 4.2 and 4.5).

Why: a load moves money. If the kiosk sends a load and the answer never comes back (timeout, network drop), the kiosk cannot tell whether the card was loaded. The key lets it send the load again safely: QOR loads the card **at most once per key**.

What the key is: a **random UUID that the kiosk generates itself** — no QOR or sale number is needed. Every platform has it built in:

Platform	Code
C# / .NET	`Guid.NewGuid().ToString()`
Java	`UUID.randomUUID().toString()`

Platform	Code
JavaScript / Node	`crypto.randomUUID()`
Python	`str(uuid.uuid4())`

Example: `Idempotency-Key: 3b9f6c1e-8d2a-4f57-a0c4-71e2d9b5a613` (any 8–100 letters, digits, `-` or `_` are accepted).

The one rule:

- **A new load → a new UUID.** Generate it when the customer starts the load, keep it in a variable.
- **Sending that same load again → the same UUID.** Reuse the variable on every retry of that load.

That is all. The kiosk does not need to store the key on disk; keeping it in memory while the sale is running is enough (section 4.3.1 shows the one case where storing it helps).

QOR keeps each key for **24 hours**, per kiosk (two kiosks using the same key do not affect each other).

3.2 The card reference number (CRN)

- Send the CRN **exactly as read** from the card. Spaces and dashes are ignored, letters are not case-sensitive.
- The CRN is a credential: the customer uses it to sign in to the balance site. **Do not** print the full CRN on receipts, write it to logs, or send it anywhere other than this API. If you need to show it, show the last 4 characters only (`•••• 4528`).

---

4. Endpoints

4.1 Get an access token

`POST /oauth/token` — OAuth 2.0 client credentials (RFC 6749 § 4.4).

Request (form-encoded):

```
POST /kiosk/v1/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=ol_kiosk_01J…&client_secret=…&scope=kiosk:load
```

Response `200`:

```
{
  "access_token": "eyJhbGciOi…",
  "token_type": "Bearer",
  "expires_in": 900,
  "scope": "kiosk:load"
}
```

- The token lives **15 minutes**. Cache it and reuse it for every call; ask for a new one shortly before it expires, or when a call answers `401 TOKEN_EXPIRED` .
- Send it on every other call: `Authorization: Bearer <access_token>` .
- Errors use the OAuth shape, not the envelope of section 3:

HTTP	`error`	Meaning	What to do
400	`invalid_request`	Missing or malformed field	Fix the request
400	`invalid_scope`	Scope other than `kiosk:load`	Send `kiosk:load` or no scope
401	`invalid_client`	Wrong id/secret, kiosk switched off, or kiosks switched off for the partner (`error_description` says so)	Stop activating; contact QOR
429	`slow_down`	More than 30 token requests in 15 minutes	Cache the token

4.2 Activate a card (the sale)

`POST /cards/activate`

```
POST /kiosk/v1/cards/activate
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "terminalId": "KIOSK-ATL-01",
  "cardReference": "0101000000000000"
}
```

Field	Type	Required	Notes
`terminalId`	string, ≤ 64	yes	The id registered for this kiosk
`cardReference`	string, ≤ 60	yes	The CRN read from the card

Response `200`:

```
{
  "data": {
    "status": "active",
    "last4": "3689",
    "message": "Card activated."
  }
}
```

- `last4` is the last four digits of the **card number** (for the receipt), not of the CRN.
- **No `Idempotency-Key`.** Activating is safe to repeat: a card is sold once. Sending activate again — after a timeout, a network drop, or by mistake — answers `200` with `"status": "active"` and `"message": "The card is already active."`, and nothing happens a second time. Two activations of the same card arriving at the same moment are handled one after the other: the card is sold once.
- If activate gets no answer, simply send it again.
- Only after a successful activate may the card be loaded.

4.3 Load a card

`POST /loads`

```
POST /kiosk/v1/loads
Authorization: Bearer <access_token>
Idempotency-Key: 3b9f6c1e-8d2a-4f57-a0c4-71e2d9b5a613
Content-Type: application/json

{
  "terminalId": "KIOSK-ATL-01",
  "cardReference": "0101000000000000",
  "amount": "25.00",
  "fee": "1.00",
  "currency": "USD"
}
```

Field	Type	Required	Notes
`terminalId`	string, ≤ 64	yes	The id registered for this kiosk
`cardReference`	string, ≤ 60	yes	The CRN of a card this kiosk activated
`amount`	string	yes	Amount loaded onto the card, e.g. `"25.00"`. Must be greater than 0
`fee`	string	no	The fee the customer paid on top, recorded for reconciliation; it is **not** loaded onto the card
`currency`	string	yes	`"USD"`

Response `201`:

```
{
  "data": {
    "transactionId": "0b8f6a52-3c1e-4f7a-9d55-6a2f1d0c9e11",
    "terminalId": "KIOSK-ATL-01",
    "status": "succeeded",
    "last4": "3689",
    "amount": "25.00",
    "fee": "1.00",
    "currency": "USD",
    "balanceAfter": "25.00",
    "message": "Funds loaded."
  }
}
```

Always check `status`, a `201` alone does not mean the money is on the card:

`status`	Meaning	What to do
`succeeded`	The money is on the card	Dispense the card, print the receipt
`failed`	The load was refused (`message` says why)	Do not dispense as loaded; refund the customer
`unknown`	QOR sent the load to the card network but got no answer back: the money may or may not be on the card	**Never send it again with a new key.** Check `GET /loads/{transactionId}` a few minutes later; QOR staff settle it. Treat the sale as pending

- `balanceAfter` is the card's available balance after the load, or `null` if it could not be read at that moment (the load itself is still done).
- Keep `transactionId` with your sale: it is the reference for reconciliation and support.
- **Limits**: each kiosk has a maximum per load and may have a daily maximum; each card has a daily maximum. Over a limit → `422 LOAD_LIMIT_EXCEEDED` (`details.remainingToday` when a daily limit is hit).
- A load that would need a manual approval at QOR is refused at once (`422 LOAD_APPROVAL_REQUIRED`): a kiosk never waits for a person.
- **Reloads**: QOR sets, per partner, whether a card may be loaded again after its first load. When reloads are off, a card takes **one successful load**: a later load is refused (`422 RELOAD_NOT_ALLOWED`). A load that `failed` does not count, so it may be tried again. While a load is still `unknown`, a new load of that card is refused too.

4.3.1 Sending a load again: same key or new key

A kiosk sends a load again when it did not get an answer: a timeout, a dropped connection, or one of the retryable errors (section 5). What happens depends only on the `Idempotency-Key` it sends.

Case 1 — the load reached QOR, only the answer was lost. Sent again with the SAME key (correct):

```
10:00:00  Kiosk → POST /loads  key K  $25.00
          QOR loads $25.00 onto the card → transaction T1, answer "succeeded"
          ✗ the answer is lost on the way back (timeout)
10:00:30  Kiosk → POST /loads  key K  $25.00   (same key, same body)
          QOR: key K is already done → returns T1's answer again, nothing loaded
          ← 201 { transactionId: T1, status: "succeeded" }   header Idempotent-Replayed: true

Result: ONE transaction (T1). Card balance $25.00. Customer paid $25.00. ✓
```

Case 2 — the same situation, but sent again with a NEW key (wrong):

```
10:00:00  Kiosk → POST /loads  key K   $25.00
          QOR loads $25.00 → transaction T1   ✗ answer lost
10:00:30  Kiosk → POST /loads  key K2  $25.00   (new key)
          QOR: K2 is new → a new load → loads $25.00 again → transaction T2
          ← 201 { transactionId: T2, status: "succeeded" }

Result: TWO transactions (T1 and T2). Card balance $50.00. Customer paid $25.00. ✗
        $25.00 was given away; it can only be taken back by hand by QOR staff.
```

Case 3 — the load never reached QOR (the network dropped on the way out). Sent again with the SAME key:

```
10:00:00  Kiosk → POST /loads  key K  $25.00   ✗ never arrives
10:00:30  Kiosk → POST /loads  key K  $25.00
          QOR: key K is new → loads $25.00 → transaction T1
          ← 201 { transactionId: T1, status: "succeeded" }

Result: ONE transaction. ✓  (The same key is just as safe when the first try never arrived.)
```

Case 4 — sent again while the first one is still running:

```
10:00:00  Kiosk → POST /loads  key K   (QOR still waiting for the card network)
10:00:05  Kiosk → POST /loads  key K
          ← 409 IDEMPOTENCY_IN_PROGRESS  (nothing done)
10:00:07  Kiosk → POST /loads  key K
          ← 201 { transactionId: T1, … }   Idempotent-Replayed: true

Result: ONE transaction. ✓
```

So the kiosk never needs to know which case it is in: **it sends the load again with the same key until it gets an answer.** QOR guarantees the money moves at most once per key.

What counts as "the same load": the same key **and** the same body (`cardReference`, `amount`, `fee`, `currency`, `terminalId`). Same key with a different body → `409 IDEMPOTENCY_KEY_REUSED`, nothing done. So:

Situation	Key
A new load (a new customer, or the same card loaded again later)	**New** UUID
No answer / timeout / dropped connection	**Same** UUID, same body
`409 IDEMPOTENCY_IN_PROGRESS`, `401 TOKEN_EXPIRED`, `429 TOO_MANY_REQUESTS`	**Same** UUID, same body (after waiting / a new token)
A clear answer came back (`succeeded`, `failed`, or any other error)	Done with this key. A different amount, or another try later, is a **new** load → **new** UUID
`status: "unknown"`	Do **not** send again. Follow up with `GET /loads/{transactionId}`

QOR remembers the **answer** of each key, errors included: sending the same key again returns the same answer, even an error. For example, a load refused with `LOAD_LIMIT_EXCEEDED` gives that same error every time it is sent again with its key; a smaller amount is a new load with a new key.

Optional — a kiosk that restarts in the middle of a load. If the kiosk crashes or reboots after sending a load but before getting its answer, the key in memory is lost and the kiosk cannot know whether the card was loaded. To cover this, a kiosk may write the key (with the CRN and amount) to disk before sending, and after a restart send that load again with the same key to get its real answer. Without this, call QOR support with the CRN's last 4 and the time: QOR staff can see whether the load went through.

4.4 Read a load again

`GET /loads/{transactionId}` — for a kiosk that lost the answer of a load, or got `status: "unknown"`.

```
GET /kiosk/v1/loads/0b8f6a52-3c1e-4f7a-9d55-6a2f1d0c9e11
Authorization: Bearer <access_token>
```

Response `200`:

```
{
  "data": {
    "transactionId": "0b8f6a52-3c1e-4f7a-9d55-6a2f1d0c9e11",
    "terminalId": "KIOSK-ATL-01",
    "status": "succeeded",
    "amount": "25.00",
    "fee": "1.00",
    "currency": "USD",
    "message": "Funds loaded."
  }
}
```

A kiosk only sees its own loads; anything else → `404 NOT_FOUND`.

If you never received the `transactionId` (the connection dropped), do not use this endpoint: **repeat the `POST /loads` with the same `Idempotency-Key`** and you get the original answer.

4.5 Lock a card

`POST /cards/lock` — for a customer who reports the card lost or stolen. The card stops working at once: no purchases and no loads.

```
POST /kiosk/v1/cards/lock
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "terminalId": "KIOSK-ATL-01",
  "cardReference": "010100000000000"
}
```

Field	Type	Required	Notes
`terminalId`	string, ≤ 64	yes	The id registered for this kiosk
`cardReference`	string, ≤ 60	yes	The CRN of a card sold at this kiosk's location

Response `200`:

```
{
  "data": {
    "status": "locked",
    "last4": "3689",
    "message": "Card locked."
  }
}
```

- Only a card **already activated** (sold) can be locked; a card still in stock → `422 CARD_NOT_ACTIVE`. A card from another location, unknown or closed → `422 CARD_NOT_ELIGIBLE`.
- **No `Idempotency-Key`.** Locking is safe to repeat: sending it again — after a timeout or by mistake — answers `200` with `"status": "locked"` and `"message": "The card is already locked."`. If lock gets no answer, simply send it again.

- **A kiosk cannot unlock a card.** The customer unlocks it on the QOR balance site, or through QOR support.
- A locked card cannot be loaded: a `POST /loads` on it is refused.

---

5. Errors

HTTP	`code`	`retryable`	Meaning	What the kiosk should do
400	`VALIDATION_ERROR`	no	A field is missing or malformed (`details` lists them)	Fix the request
400	`IDEMPOTENCY_KEY_REQUIRED`	no	A load without a valid `Idempotency-Key`	Send a UUID (section 3.1)
401	`AUTH_REQUIRED`	no	No or invalid token	Get a token
401	`TOKEN_EXPIRED`	**yes**	The token expired	Get a new token, send again (a load: same key)
403	`DEVICE_DISABLED`	no	QOR switched this kiosk off	Stop activating, contact QOR
403	`KIOSKS_DISABLED`	no	Kiosks are switched off for the partner	Stop activating, contact QOR
403	`TERMINAL_MISMATCH`	no	`terminalId` is not the one registered for this kiosk	Fix the kiosk's configuration
409	`IDEMPOTENCY_KEY_REUSED`	no	A load key already used for a different load	Bug: a new load needs a new UUID (4.3.1)
409	`IDEMPOTENCY_IN_PROGRESS`	**yes**	The same load is still running	Wait 1–2 s, send again with the same key
422	`CARD_NOT_ELIGIBLE`	no	Card unknown, not in this location's stock, or closed	Do not activate this card; set it aside and tell QOR
422	`CARD_NOT_ACTIVE`	no	Load or lock before activate	Activate first (a card in stock needs no lock)
422	`LOAD_LIMIT_EXCEEDED`	no	Over a kiosk or card limit	Ask for a smaller amount
422	`LOAD_APPROVAL_REQUIRED`	no	Amount needs a manual approval	Ask for a smaller amount
422	`RELOAD_NOT_ALLOWED`	no	Reloads are off for the partner and this card already has a load	Do not load this card again; refund the customer
422	`LOAD_CURRENCY_MISMATCH`	no	Wrong currency for this card	Send `USD`
422	`INSUFFICIENT_FUNDS`	no	The partner's prefunded balance at QOR is too low	Stop loads, tell the partner to add funds
422	`PROCESSOR_REFUSED`	no	The card network refused the action (`message` says why)	Do not activate as loaded
429	`TOO_MANY_REQUESTS`	**yes**	Too many calls (e.g. more than 60 card look-ups a minute per kiosk)	Slow down, send again (a load: same key)

HTTP	`code`	`retryable`	Meaning	What the kiosk should do
502	`PROCESSOR_UNAVAILABLE`	**yes**	The card network did not answer an activate or a lock	Send it again after a few seconds (a load never answers this: it answers `status: "unknown"`)
503	`LOADS_SUSPENDED`	no	Loads paused by QOR	Try later
500	`INTERNAL_ERROR`	no	Unexpected	Do not send a load again with a new key; contact QOR with the `traceId`

Retry rule: send again only when `retryable` is `true` or on a network error / timeout, with a back-off (e.g. 1 s, 2 s, 4 s, at most 5 attempts). An activate or a lock is simply sent again; a load is sent again with its **same** **`Idempotency-Key`** and the same body (4.3.1).

---

6. Recommended sale sequence

```
1. token := cached token, or POST /oauth/token
2. read CRN from the card
3. POST /cards/activate                    (no key)
     200                      → go on
     CARD_NOT_ELIGIBLE        → reject the card, offer another one
     retryable / no answer    → send 3 again
4. take the customer's money
5. loadKey := new random UUID              (kept in memory for this load)
   POST /loads   Idempotency-Key: loadKey
     succeeded                → dispense, receipt with transactionId and last4
     failed / 4xx not retry   → refund
     unknown                  → receipt "pending", follow up with GET /loads/{transactionId}
     retryable / no answer    → send 5 again with the SAME loadKey and the same body
```

Taking the money **before** activating is also fine. In that case, if activate fails, refund and offer another card. A card that was activated but never loaded can be loaded later: that is a new load, with a new UUID.

---

7. Security checklist

- [ ] `client_secret` kept in the OS credential store, never in files, logs or screenshots.
- [ ] Access token kept in memory only.
- [ ] Full CRN never printed, logged or stored. Receipts show `last4` and `transactionId` only.
- [ ] HTTPS with certificate validation on (no "accept any certificate").
- [ ] A new random UUID for each load, reused (with the same body) on every retry of that load.
- [ ] The kiosk clock synchronised (NTP).