

ICD | LTU Fuel Card Interface
Doc No. 20009 | Version 4.0 | 16.09.2026 | Released

LTU Nationwide GNSS Tolling System

ICD

LTU Fuel Card Interface

Doc No. 20009

Version 4.0

Author	Juan Pablo Gottret
Reviewed:	Roman Trinko
Approved by:	Hans Gidoff

Overview of changes.

No.	Version	Status	Date	Contributor	Type of the change
1	0.1	DRAFT	18.03.2026	Juan Pablo Gottret	First Version
2	1.0	Released	25.03.2026	Roman Trinko	Minor change
3	2.0	Released	03.06.2026	Franco Di Stefano	Completely reworked based meetings with FCP and received documentation
4	3.0	Released	10.06.2026	Franco Di Stefano	Method for Payment Demands added
5	3.1	Draft	11.09.2026	Franco Di Stefano	Updates after technical alignment with FCPs
6	4.0	Released	16.09.2026	PMO	New released version

Table 1 Overview of changes

Reference to the status, versions and data classification.
Status:

Draft	the document is being processed
Released	the document has been checked and released, it can only be modified if the version number is updated.
Obsolete	the document is not valid anymore

Versions:

0.1, 0.2	draft versions
1.0	first released version with the status “Released“
1.1, 1.2, etc.	draft versions, that supplement version 1.0
2.0	second released version with the status “Released”

Data classification

Public	No restriction
Internal	Restricted to internal and external Kapsch employees
Confidential	Restricted to selected active directory and/or sharepoint groups (default)
Secret	Restricted to selected employees, server encryption needed

Table of Contents

1	General	8
1.1	Purpose of the document	8
2	Requirements for Fuel Card Providers	8
2.1	Integration Method	8
2.2	Fuel Card API Response Codes	8
2.3	IFrame or equivalent to enter Fuel Card details.....	9
2.4	Required API Methods	9
2.4.1	Tokenize	9
2.4.1.1	Purpose	9
2.4.1.2	Endpoint Definition	9
2.4.1.3	Request Headers	10
2.4.1.4	Request Parameters	10
2.4.1.4.1	Option A (Preferred – iFrame-based tokenization)	10
2.4.1.4.2	Option B (Direct API tokenization — only if PCI-compliant)	10
2.4.1.5	Processing Rules	10
2.4.1.6	Response Parameters.....	11
2.4.1.7	Response Status Interpretation.....	11
2.4.1.8	Example Request	12
2.4.1.9	Example Response	12
2.4.1.10	Error Handling	12
2.4.2	Authorization	12
2.4.2.1	Purpose	12
2.4.2.2	Endpoint Definition	13
2.4.2.3	Request Headers	13
2.4.2.4	Request Parameters	13
2.4.2.5	Processing Rules	13
2.4.2.6	Response Parameters.....	14
2.4.2.7	Response Status Values	14
2.4.2.8	Example Request	15

2.4.2.9	Example Response	15
2.4.2.10	Error Handling	15
2.4.3	Bin (optional)	15
2.4.3.1	Purpose	15
2.4.3.2	Endpoint Definition	16
2.4.3.3	Request Headers	16
2.4.3.4	Request Parameters	16
2.4.3.5	Processing Rules	16
2.4.3.6	Response Parameters.....	16
2.4.3.7	Response Status Interpretation.....	17
2.4.3.8	Example Request.....	17
2.4.3.9	Example Response	17
2.4.3.10	Error Handling	17
2.4.4	Capture.....	18
2.4.4.1	Purpose	18
2.4.4.2	Endpoint Definition	18
2.4.4.3	Request Headers	18
2.4.4.4	Request Parameters	18
2.4.4.5	Processing Rules	19
2.4.4.6	Response Parameters.....	19
2.4.4.7	Response Status Values	19
2.4.4.8	Example Request.....	20
2.4.4.9	Example Response	20
2.4.4.10	Error Handling	20
2.4.5	Refund.....	20
2.4.5.1	Purpose	20
2.4.5.2	Endpoint Definition	21
2.4.5.3	Request Headers	21
2.4.5.4	Request Parameters	21
2.4.5.5	Processing Rules	21
2.4.5.6	Response Parameters.....	22
2.4.5.7	Response Status Values.....	22
2.4.5.8	Example Request.....	23

2.4.5.9	Example Response	23
2.4.5.10	Error Handling	23
2.4.6	Void	23
2.4.6.1	Purpose	23
2.4.6.2	Endpoint Definition	24
2.4.6.3	Request Headers	24
2.4.6.4	Request Parameters	24
2.4.6.5	Processing Rules	24
2.4.6.6	Response Parameters.....	25
2.4.6.7	Response Status Values	25
2.4.6.8	Example Request	26
2.4.6.9	Example Response	26
2.4.6.10	Error Handling	26
2.4.7	Void by Order Id	26
2.4.7.1	Purpose	26
2.4.7.2	Endpoint Definition	27
2.4.7.3	Request Headers	27
2.4.7.4	Request Parameters	27
2.4.7.5	Processing Rules	27
2.4.7.6	Response Parameters.....	28
2.4.7.7	Response Status Values	28
2.4.7.8	Example Request	29
2.4.7.9	Example Response (Success)	29
2.4.7.10	Example Response (Not Found)	29
2.4.7.11	Error Handling	29
2.4.8	Query by Authorization Reference	29
2.4.8.1	Purpose	29
2.4.8.2	Endpoint Definition	30
2.4.8.3	Request Headers	30
2.4.8.4	Request Parameters	30
2.4.8.5	Response Parameters.....	30
2.4.8.6	Status Model	31
2.4.8.7	Example Request	31

2.4.8.8	Example Response	31
2.4.9	Inquire by Order Id (generated by CBO)	31
2.4.9.1	Purpose	31
2.4.9.2	Endpoint Definition	32
2.4.9.3	Request Headers	32
2.4.9.4	Request Parameters	32
2.4.9.5	Response Parameters.....	32
2.4.9.6	Response Status Values	32
2.4.9.7	Example Request	33
2.4.9.8	Example Response	33
2.5	FCP Clearing Files generated by CBO	33
2.5.1	Purpose	33
2.5.2	Overview	34
2.5.3	Types of Clearing Files.....	34
2.5.4	File Naming Convention	35
2.5.5	File Formats	35
2.5.6	Field Formats	35
2.5.7	Date and Time Formats.....	36
2.5.8	Mandatory Fields.....	36
2.5.9	File Structure	37
2.5.9.1	Daily files	37
2.5.9.1.1	Header Record (R1)	37
2.5.9.1.2	Transaction Record (R2).....	37
2.5.9.1.3	Trailer Record (R4)	38
2.5.9.2	Payment Demand files	38
2.5.9.2.1	Header Record (R1)	38
2.5.9.2.2	Aggregated Record (R3)	39
2.5.9.2.3	Trailer Record (R4)	40
2.5.10	Transaction Classification	40
2.5.11	Delivery Mechanism	40
2.5.12	Acknowledgement Mechanism for Daily files.....	41
2.5.12.1	Request parameters.....	41
2.5.12.1.1	Acknowledgement	41

2.5.12.2	Response parameters	42
2.5.12.3	Example Request 1 (All records OK)	43
2.5.12.4	Example Request 2 (Mixed OK + Errors).....	43
2.5.12.5	Example Response 1 (No errors).....	44
2.5.12.6	Example Response 2 (Validation Errors in Request).....	44
3	Examples of Payment Flows.....	46
3.1	Fuel card payment without registered means of payment	46
3.2	Registration of a fuel card as a payment method for registered users	47
3.3	Payment of a Route Pass by a registered user with post-pay account	48
3.4	Start a trip by a registered user with post-pay account- Fee Reservation	49
3.5	End trip by a registered user with post-pay account.....	50
3.6	Posting toll for an user with OBU (and Fuel Card as an automatic payment method)	51

1 General

1.1 Purpose of the document

The purpose of this document is to describe the technical interfaces that support payments by **Fuel Cards**, in the course of the Lithuania Nationwide Tolling System.

The design supports **Online Operations** (via API) and **Clearing** (via File Exchange).

2 Requirements for Fuel Card Providers

2.1 Integration Method

- Integration must be implemented through a **RESTful API**.
- Fuel card providers must expose **API endpoints**.
 - **API Security:** All APIs must be secured using **OAuth2 authentication**. Before invoking any API endpoint, the Source System must first request an OAuth2 access token from the Fuel Card Provider authorization server and include the issued token in the Authorization: Bearer <access_token> request header.
- Fuel card APIs must support **tokenization**, meaning card details are tokenized by the Fuel Card Provider (FCP) and never stored or processed as raw data by source systems.
- Fuel card providers must **exchange files for clearing** via secure protocol.

2.2 Fuel Card API Response Codes

Fuel card providers must manage the behaviour of the API responses based on the next codes and results:

Fuel card API Response Codes

1. **HTTP 200 (OK):** Successful requests
2. **HTTP 409 (Conflict):** Business related request failures
3. **HTTP 500 (Internal Server Error):** Technical failures

For **failed requests** (HTTP 409 & HTTP 500), a common error response object is expected:

Parameter	Required	Type	Description
errorCode	Yes	String	ERROR / DECLINED
responseCode	Yes	String	Provider-specific code
responseMessage	Yes	String	Human-readable description

Example failed request error response:

```
{  
  "errorCode": "ERROR",  
  "responseCode": "string",  
  "responseMessage": "string"  
}
```

2.3 IFrame or equivalent to enter Fuel Card details

PCI-DSS compliance is the responsibility of the Payment Integrator. For this reason, the Fuel Card Provider (FCP) must provide a secure IFrame or equivalent to handle fuel card data entry.

When a user registers a fuel card for future use, the mobile/web application redirects the user to the FCP IFrame. All fuel card details are entered directly within this IFrame, ensuring that no sensitive card data is stored by source systems. The fuel card information remains exclusively within the Fuel Card Provider's secure environment.

Once the card details are entered and submitted, the FCP validates the data and generates a token. This token is then stored in the source system and used for all subsequent payment operations, allowing future transactions without re-entering fuel card details while maintaining PCI-DSS compliance.

When an unregistered user purchases a route pass, the mobile/web application also redirects the user to the Fuel Card Provider (FCP) IFrame to enter their fuel card details and obtain a token. Once the token is generated, the mobile application resumes the payment process using this token to complete the transaction.

2.4 Required API Methods

2.4.1 Tokenize

2.4.1.1 Purpose

The Tokenize method is used to convert sensitive fuel card data into a **secure token** that can be safely stored and reused by the **Source System**.

This operation:

- Ensures **PCI-DSS compliance** by preventing storage of raw card data
- Enables subsequent payment operations (Authorization, Capture, Refund, etc.)
- Is executed after card data is collected via the **FCP-provided secure iframe**

The Source System must never handle or persist raw card details.

2.4.1.2 Endpoint Definition

POST /cards/tokenize

2.4.1.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.1.4 Request Parameters

Two integration patterns are supported depending on FCP design.

2.4.1.4.1 Option A (Preferred – iFrame-based tokenization)

Parameter	Required	Type	Description
sessionId	Yes	String	Identifier linking the iframe session
customerId	No	String	Identifier of the end user in Source System

In this model:

- Card data is entered in the **FCP iframe**
- The FCP internally processes the data and returns a token tied to `sessionId`

2.4.1.4.2 Option B (Direct API tokenization — only if PCI-compliant)

Parameter	Required	Type	Description
cardNumber	Yes	String	Fuel card number
expirationDate	Yes	String	Card expiration date
cardHolderName	No	String	Cardholder name

This option must only be used if:

- The Source System is **fully PCI-DSS compliant**
- Secure transmission and storage controls are guaranteed

2.4.1.5 Processing Rules

4. PCI Compliance

- Tokenization must ensure that:
 - No sensitive data is stored in the Source System
 - Tokens cannot be reversed into original card data

5. Token Lifecycle

Internal

- Token must:
 - Uniquely represent a card
 - Be reusable across multiple transactions
 - Be scoped to the FCP
- 6. **Session Handling (iframe mode)**
 - sessionId must:
 - Be generated before iframe initialization
 - Expire after a configurable timeout
 - Be single-use (recommended)
- 7. **Validation**
 - FCP must validate:
 - Card format
 - Expiry validity
 - Card eligibility (optional)
- 8. **Idempotency**
 - Reusing the same session should not generate multiple tokens
 - Duplicate requests must return the same token (if already created)

2.4.1.6 Response Parameters

Parameter	Required	Type	Description
fuelCardToken	Yes	String	Token representing the fuel card
maskedCardNumber	Yes	String	Masked card number (e.g., *****1234)
expirationDate	Yes	String	Card expiration date (MMYY or ISO format depending on FCP)
cardType	No	String	Card type (FUEL, CREDIT, etc.)
issuerName	No	String	Issuing provider
responseCode	Yes	String	Provider-specific code
responseMessage	Yes	String	Human-readable description

2.4.1.7 Response Status Interpretation

Condition	Expected Result
Successful tokenization	Token returned
Invalid card data	ERROR
Expired card	DECLINED

Condition	Expected Result
Token already exists	Same token returned
Technical failure	ERROR

2.4.1.8 Example Request

```
{
  "sessionId": "SESSION-123456",
  "customerId": "CUST-001"
}
```

2.4.1.9 Example Response

```
{
  "fuelCardToken": "tok_abcl123xyz",
  "maskedCardNumber": "*****1234",
  "expirationDate": "1227",
  "cardType": "FUEL",
  "issuerName": "FCP1",
  "responseCode": "00",
  "responseMessage": "Tokenization successful",
}
```

2.4.1.10 Error Handling

Scenario	Expected Result
Invalid sessionId	ERROR
Card validation failure	DECLINED
Expired session	ERROR
Duplicate session usage	Same token returned
FCP unavailable	ERROR

2.4.2 Authorization

2.4.2.1 Purpose

The Authorization method is used to validate and reserve (or directly capture) funds on a fuel card for a specific transaction initiated by CBO.

This operation ensures that:

Internal

- The fuel card is valid and active
- The requested amount is available within the card limits
- A financial authorization is created in the Fuel Card Provider (FCP) system

Partial authorizations are **not supported**. If the requested amount cannot be fully authorized, the transaction must be declined.

2.4.2.2 Endpoint Definition

POST /payments/authorization

2.4.2.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.2.4 Request Parameters

Parameter	Required	Type	Description
orderId	Yes	String	Unique identifier generated by source system for idempotency and traceability
fuelCardToken	Yes	String	Token representing the fuel card
expirationDate	Yes	String	Card expiration date (MMYY or ISO format depending on FCP)
amount	Yes	String	Max Length: 14 Example: 106.00 Amount to authorize with decimal or without decimal in currency mayor units (e.g., EUR).
capture	No	String	Defines whether capture is performed immediately If the FCP does not support a reserve, the transaction is captured for settlement if approved. When supported, specify Y to capture the transaction for settlement, if approved. Defaults to N if not provided.
merchantId	No	String	Identifier of the merchant in the FCP system

2.4.2.5 Processing Rules

1. Idempotency

- The orderId must be unique per transaction.
- Repeated requests with the same orderId must return the same result.

2. Capture Behavior

- capture = N → Authorization hold only (reservation of funds)
- capture = Y → Immediate capture (sale transaction)
- If the FCP does **not support authorization-only**, the request must be treated as capture = Y

3. Authorization Strategy

- Full amount must be authorized or transaction is declined.
- No partial approvals allowed.

4. Timeout / Uncertain State

- If no response is received:
 - Transaction status must be verified using **Query by Order ID**
 - If unresolved → fallback to **Void by Order ID**

2.4.2.6 Response Parameters

Parameter	Required	Type	Description
authorizationCode	Yes	String	Unique reference assigned by the FCP
status	Yes	String	Transaction result (APPROVED / DECLINED / ERROR)
responseCode	Yes	String	Provider-specific response code
responseMessage	Yes	String	Human-readable explanation
authorizedAmount	Yes	String	Amount authorized (should match request)

2.4.2.7 Response Status Values

Status	Description
APPROVED	Authorization successful
DECLINED	Authorization rejected (e.g., insufficient funds)
ERROR	Technical or validation failure

2.4.2.8 Example Request

```
{
  "orderId": "ORD-0001",
  "fuelCardToken": "tok_fcp_abc123xyz",
  "expirationDate": "1227",
  "amount": 15.00,
  "capture": "N"
}
```

2.4.2.9 Example Response

```
{
  "authorizationCode": "FCP-987654321",
  "status": "APPROVED",
  "responseCode": "00",
  "responseMessage": "Authorization approved",
  "authorizedAmount": 15.00
}
```

2.4.2.10 Error Handling

Scenario	Expected Behavior
Invalid token	DECLINED / ERROR
Insufficient funds	DECLINED
Duplicate orderId	Return previous response
Format validation error	ERROR

2.4.3 Bin (optional)

2.4.3.1 Purpose

The BIN method is used to retrieve metadata associated with a fuel card based on the provided token. This operation enables the **Source System** to:

- Identify the card issuer and Fuel Card Provider
- Determine card type and service level
- Apply routing, eligibility, or business rules

This is an **enrichment operation** and does **not perform any financial transaction**.

2.4.3.2 Endpoint Definition

POST /cards/bin

2.4.3.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.3.4 Request Parameters

Parameter	Required	Type	Description
fuelCardToken	Yes	String	Token representing the fuel card
bin (optional)	No	String	First 4–6 digits of the card (if token not sufficient or fallback scenario)

2.4.3.5 Processing Rules

1. Primary Lookup

- The FCP should resolve BIN data using the provided fuelCardToken.

2. Fallback Behavior

- If token-based BIN lookup is not supported:
 - The optional bin parameter may be used

3. Non-Blocking Operation

- This method must **not block payment flows**
- If unavailable or failing, the Source System may continue with Authorization

4. Usage Timing

- Typically invoked:
 - During card registration (tokenization flow)
 - Prior to authorization for validation/routing

2.4.3.6 Response Parameters

Parameter	Required	Type	Description
issuerName	Yes	String	Name of issuing bank or provider
fuelCardProvider	Yes	String	Identifier of the Fuel Card Provider

Parameter	Required	Type	Description
cardType	Yes	String	Type of card (e.g., CREDIT, DEBIT, FUEL)
country	Yes	String	ISO country code of issuer
supported	Yes	String	Indicates if card is accepted by the FCP (Y / N values)
responseCode	Yes	String	Provider-specific response code
responseMessage	Yes	String	Human-readable description

2.4.3.7 Response Status Interpretation

Condition	Expected Behavior
Valid BIN	Full card metadata returned
Token not recognized	ERROR
Technical failure	ERROR

2.4.3.8 Example Request

```
{
  "fuelCardToken": "tok_fcp_abc123xyz"
}
```

2.4.3.9 Example Response

```
{
  "issuerName": "FCP1 Euro Service",
  "fuelCardProvider": "FCP1",
  "cardType": "FUEL",
  "country": "LT",
  "supported": "Y",
  "responseCode": "00",
  "responseMessage": "BIN details retrieved successfully"
}
```

2.4.3.10 Error Handling

Scenario	Expected Result
Invalid token	ERROR
BIN not recognized	supported = "N"

Scenario	Expected Result
Missing parameters	ERROR
Service unavailable	ERROR (non-blocking)

2.4.4 Capture

2.4.4.1 Purpose

The Capture method is used to finalize a previously authorized transaction by transferring (capturing) the reserved funds from the cardholder to the merchant.

This operation:

- Completes a prior **Authorization (capture = false)**
- Confirms the final amount to be settled
- Transitions the transaction from **AUTHORIZED** → **CAPTURED**

The captured amount may differ from the originally authorized amount, depending on the final charge calculation.

2.4.4.2 Endpoint Definition

POST /payments/capture

2.4.4.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.4.4 Request Parameters

Parameter	Required	Type	Description
authorizationCode	Yes	String	Reference returned from Authorization
orderId	Yes	String	Unique identifier from the Source System
amount	Yes	String	Amount to capture (mayor units)

2.4.4.5 Processing Rules

1. Precondition

- A valid, **APPROVED Authorization must exist**
- Authorization must not be expired, voided, or already fully captured

2. Capture Amount Rules

- Capture amount:
 - Can be **equal to** or **less than** the authorized amount
 - Must **not exceed** the authorized amount (unless FCP explicitly supports it)
- If less than authorized:
 - Remaining amount must be automatically released (or handled per FCP rules)

3. Idempotency

- The combination of orderId + authorizationCode must ensure idempotent behavior
- Duplicate requests must return the same result

4. Timeout / Uncertain State

- If no response is received:
 - Transaction must be verified using **Query Operations**
 - No retry without idempotency guarantee

2.4.4.6 Response Parameters

Parameter	Required	Type	Description
captureReference	Yes	String	Unique identifier of the capture transaction
authorizationCode	Yes	String	Original authorization reference
status	Yes	String	Result of capture
responseCode	Yes	String	Provider-specific response code
responseMessage	Yes	String	Human-readable description
capturedAmount	No	String	Amount successfully captured

2.4.4.7 Response Status Values

Status	Description
APPROVED	Capture successful
DECLINED	Capture rejected
ERROR	Technical or validation issue

2.4.4.8 Example Request

```
{
  "authorizationCode": "FCP-987654321",
  "orderId": "ORD-0001",
  "amount": 15.00
}
```

2.4.4.9 Example Response

```
{
  "captureReference": "CAP-123456789",
  "authorizationCode": "FCP-987654321",
  "status": "APPROVED",
  "responseCode": "00",
  "responseMessage": "Capture successful",
  "capturedAmount": 13500,
}
```

2.4.4.10 Error Handling

Scenario	Expected Result
Authorization not found	ERROR
Authorization already captured	DECLINED
Capture amount > authorized	DECLINED
Expired authorization	DECLINED
Duplicate capture request	Same response (idempotent)

2.4.5 Refund

2.4.5.1 Purpose

The Refund method is used to return funds to the cardholder for a previously **captured (settled) transaction**.

This operation:

- Reverses a financial transaction after settlement
- Supports **full or partial refunds**
- Creates a new financial record linked to the original transaction

Refunds can only be applied to successfully captured transactions.

Internal

2.4.5.2 Endpoint Definition

POST /payments/refund

2.4.5.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.5.4 Request Parameters

Parameter	Required	Type	Description
authorizationCode	Yes	String	Reference of the original transaction
orderId	Yes	String	Unique identifier of the refund request from the Source System
amount	Yes	String	Amount to refund (mayor units)
reason	No	String	Reason for refund (e.g., cancellation, dispute, adjustment)

2.4.5.5 Processing Rules

1. Precondition

- Refund can only be performed on:
 - Successfully **CAPTURED** transactions
- Authorization-only transactions must not be refunded (use **Void** instead)

2. Refund Amount Rules

- Refund amount:
 - Can be **partial or full**
 - Must not exceed the **total captured amount minus previous refunds**
- Multiple partial refunds are allowed until the full amount is refunded

3. Transaction Linking

- Each refund must be linked to:
 - The original authorizationCode
- The Source System must maintain traceability between:
 - Original transaction
 - All associated refunds

4. Idempotency

- orderId must be unique per refund attempt

- Duplicate requests must return the same result

5. Settlement Behavior

- Refunds are typically processed in the next clearing cycle
- The actual credit to the cardholder depends on FCP processing timelines

6. Timeout / Uncertain State

- If no response:
 - Must verify using **Query Operations**
 - Avoid blind retry without idempotency guarantee

2.4.5.6 Response Parameters

Parameter	Required	Type	Description
refundReference	Yes	String	Unique refund transaction identifier
authorizationCode	Yes	String	Original transaction reference
status	Yes	String	Result of the refund operation
responseCode	Yes	String	Provider-specific response code
responseMessage	Yes	String	Human-readable description
refundedAmount	No	String	Amount successfully refunded

2.4.5.7 Response Status Values

Status	Description
APPROVED	Refund successfully processed
DECLINED	Refund rejected
ERROR	Technical or validation issue

2.4.5.8 Example Request

```
{
  "authorizationCode": "FCP-987654321",
  "orderId": "REF-0001",
  "amount": 5.00,
  "reason": "Trip cancellation"
}
```

2.4.5.9 Example Response

```
{
  "refundReference": "REF-789456123",
  "authorizationCode": "FCP-987654321",
  "status": "APPROVED",
  "responseCode": "00",
  "responseMessage": "Refund successful",
  "refundedAmount": 5.00,
}
```

2.4.5.10 Error Handling

Scenario	Expected Result
Original transaction not found	ERROR
Transaction not captured	DECLINED
Refund amount exceeds available refundable	DECLINED
Duplicate refund request	Same response (idempotent)
Already fully refunded	DECLINED

2.4.6 Void

2.4.6.1 Purpose

The Void method is used to cancel a previously authorized or captured transaction **before it is settled (clearing stage)**.

This operation:

- Releases reserved funds (if only authorized)
- Prevents settlement (if already captured but not yet cleared)
- Ensures no financial posting occurs in clearing files

Void is the correct operation for cancelling transactions that have **not yet been settled**, whereas Refund applies after settlement.

2.4.6.2 Endpoint Definition

POST /payments/void

2.4.6.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.6.4 Request Parameters

Parameter	Required	Type	Description
authorizationCode	Yes	String	Reference of the original authorization
orderId	Yes	String	Unique identifier of the void request from the Source System
reason	No	String	Reason for cancellation

2.4.6.5 Processing Rules

1. Precondition

- Transaction must exist and be in one of the following states:
 - AUTHORIZED (not captured)**
 - CAPTURED but not yet settled**
- Transactions already settled must **not** be voided → use **Refund**

2. Void Behavior

- If transaction is **AUTHORIZED**:
 - The authorization hold must be released
- If transaction is **CAPTURED (pre-settlement)**:
 - The capture must be cancelled and excluded from settlement

3. Settlement Boundary

- The FCP must define the settlement cut-off:
 - After settlement → Void not allowed
 - Before settlement → Void allowed

4. Idempotency

- orderId must ensure idempotent behavior
- Duplicate requests must return the same result

5. Timeout / Uncertain State

- If no response is received:
 - Source System must verify using **Query Operations**
 - Avoid blind retries without idempotency

2.4.6.6 Response Parameters

Parameter	Required	Type	Description
voidReference	Yes	String	Unique identifier of the void operation
authorizationCode	Yes	String	Original transaction reference
status	Yes	String	Result of the void operation
responseCode	Yes	String	Provider-specific response code
responseMessage	Yes	String	Human-readable description

2.4.6.7 Response Status Values

Status	Description
APPROVED	Void successful
DECLINED	Void rejected (e.g., already settled)
ERROR	Technical or validation issue

2.4.6.8 Example Request

```
{
  "authorizationCode": "FCP-987654321",
  "orderId": "VOID-0001",
  "reason": "Trip cancelled before completion"
}
```

2.4.6.9 Example Response

```
{
  "voidReference": "VOID-456123789",
  "authorizationCode": "FCP-987654321",
  "status": "APPROVED",
  "responseCode": "00",
  "responseMessage": "Transaction voided successfully"
}
```

2.4.6.10 Error Handling

Scenario	Expected Result
Transaction not found	ERROR
Already settled transaction	DECLINED
Already voided transaction	DECLINED (idempotent-safe)
Invalid authorization reference	ERROR
Duplicate void request	Same response

2.4.7 Void by Order Id

2.4.7.1 Purpose

The Void by Order ID method is used to cancel a transaction using the **order identifier generated by the Source System**, instead of the FCP authorization reference.

This method is designed as a **recovery and resiliency mechanism** when the transaction reference is not available or the transaction status is uncertain.

Typical use cases include:

- No response received from Authorization or Capture
- Loss of the FCP reference due to communication failure
- Inconclusive result after retry attempts

Internal

- Failure of **Query by Order ID** or inconsistent states

This method ensures that orphan or uncertain transactions can be safely cancelled.

2.4.7.2 Endpoint Definition

POST /payments/void-by-order-id

2.4.7.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	Auth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.7.4 Request Parameters

Parameter	Required	Type	Description
orderId	Yes	String	Unique identifier generated by the Source System for the original transaction
reason	No	String	Reason for cancellation

2.4.7.5 Processing Rules

1. Lookup Strategy

- The FCP must resolve the transaction internally using the provided orderId
- This may match:
 - Authorization
 - Capture
 - Pending transaction

2. Transaction State Handling

- If transaction is found:
 - Apply **Void logic** according to its current state:
 - ♦ AUTHORIZED → release funds
 - ♦ CAPTURED (pre-settlement) → cancel capture
- If transaction is already settled:
 - Void must be rejected → use **Refund**

3. Uncertain or Missing Transaction

- If no transaction is found for the given orderId:
 - Response must clearly indicate **NOT FOUND**

- Source System must treat this as:
 - Final state OR trigger reconciliation
- 4. **Idempotency**
 - orderId guarantees idempotency
 - Repeated requests must return consistent results
- 5. **Recovery Use Case**
 - Must be used only when:
 - authorizationCode is unavailable
 - Standard Void cannot be executed
 - Should not replace normal Void flow
- 6. **Timeout / Uncertain State**
 - If no response:
 - Source System must retry safely (idempotent)
 - Followed by **Query Operations**

2.4.7.6 Response Parameters

Parameter	Required	Type	Description
voidReference	No	String	Identifier of the void operation (if transaction found)
orderId	Yes	String	Original order identifier
status	Yes	String	Result of the operation
responseCode	Yes	String	Provider-specific response code
responseMessage	Yes	String	Human-readable description

2.4.7.7 Response Status Values

Status	Description
APPROVED	Transaction successfully voided
DECLINED	Transaction cannot be voided (e.g., already settled)
NOT_FOUND	No transaction found for given orderId
ERROR	Technical or validation issue

2.4.7.8 Example Request

```
{  
  "orderId": "ORD-20260603-0001",  
  "reason": "Authorization response not received"  
}
```

2.4.7.9 Example Response (Success)

```
{  
  "voidReference": "VOID-99887766",  
  "orderId": "ORD-20260603-0001",  
  "status": "APPROVED",  
  "responseCode": "00",  
  "responseMessage": "Transaction voided successfully",  
}
```

2.4.7.10 Example Response (Not Found)

```
{  
  "orderId": "ORD-20260603-0001",  
  "status": "NOT_FOUND",  
  "responseCode": "404",  
  "responseMessage": "No transaction found for given orderId",  
}
```

2.4.7.11 Error Handling

Scenario	Expected Result
Transaction not found	NOT_FOUND
Already settled transaction	DECLINED
Already voided	APPROVED / idempotent response
Invalid orderId format	ERROR
Duplicate request	Same response

2.4.8 Query by Authorization Reference

2.4.8.1 Purpose

Retrieves the status and details of a transaction using the **authorization reference** assigned by the Fuel Card Provider.

This method is typically used when:

- The Source System has received a response from the FCP
- A follow-up verification is required (e.g., reconciliation, audit)
- Clarification of settlement status is needed

This is the **primary query method** when the FCP reference is known.

2.4.8.2 Endpoint Definition

POST /payments/query/by-reference

2.4.8.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	Auth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.8.4 Request Parameters

Parameter	Required	Type	Description
authorizationCode	Yes	String	FCP transaction reference

2.4.8.5 Response Parameters

Parameter	Required	Type	Description
authorizationCode	Yes	String	FCP reference
orderId	Yes	String	Source System identifier
status	Yes	String	Current transaction status
transactionType	Yes	String	AUTHORIZATION / CAPTURE / REFUND / VOID
amount	Yes	Integer	Transaction amount
settlementStatus	No	String	SETTLED / NOT_SETTLED / PARTIALLY_SETTLED
responseCode	Yes	String	Provider code
responseMessage	Yes	String	Description

2.4.8.6 Status Model

Status	Description
AUTHORIZED	Funds reserved
CAPTURED	Funds captured
PARTIALLY_CAPTURED	Partial capture completed
REFUNDED	Fully refunded
PARTIALLY_REFUNDED	Partial refund
VOIDED	Transaction cancelled
DECLINED	Transaction rejected
NOT_FOUND	No transaction found

2.4.8.7 Example Request

```
{  
  "authorizationCode": "FCP-987654321"  
}
```

2.4.8.8 Example Response

```
{  
  "authorizationCode": "FCP-987654321",  
  "orderId": "ORD-0001",  
  "status": "CAPTURED",  
  "transactionType": "CAPTURE",  
  "amount": 15.00,  
  "settlementStatus": "NOT_SETTLED",  
  "responseCode": "00",  
  "responseMessage": "Transaction captured successfully",  
}
```

2.4.9 Inquire by Order Id (generated by CBO)

2.4.9.1 Purpose

Retrieves transaction details using the **orderId generated by the Source System**.

This method is **critical for resiliency and recovery**, especially when:

- No response was received from Authorization/Capture
- The FCP reference is missing

- The transaction outcome is uncertain

This is the **fallback query method** for transaction recovery scenarios.

2.4.9.2 Endpoint Definition

POST /payments/query/by-order-id

2.4.9.3 Request Headers

Header	Required	Value	Description
Content-Type	Yes	application/json	Payload format
Authorization	Yes	Bearer <access_token>	OAuth2 access token obtained from the FCP OAuth2 token endpoint before calling this API.

2.4.9.4 Request Parameters

Parameter	Required	Type	Description
orderId	Yes	String	Unique identifier generated by the Source System

2.4.9.5 Response Parameters

Parameter	Required	Type	Description
authorizationCode	No	String	FCP reference (if exists)
orderId	Yes	String	Source System identifier
status	Yes	String	Current transaction status
transactionType	No	String	AUTHORIZATION / CAPTURE / REFUND / VOID
amount	No	String	Transaction amount
settlementStatus	No	String	SETTLED / NOT_SETTLED / PARTIALLY_SETTLED
responseCode	Yes	String	Provider code
responseMessage	Yes	String	Description

2.4.9.6 Response Status Values

Status	Description
AUTHORIZED	Funds reserved

Status	Description
CAPTURED	Funds captured
PARTIALLY_CAPTURED	Partial capture completed
REFUNDED	Fully refunded
PARTIALLY_REFUNDED	Partial refund
VOIDED	Transaction cancelled
DECLINED	Transaction rejected
NOT_FOUND	No transaction found

2.4.9.7 Example Request

```
{  
  "orderId": "ORD-0001"  
}
```

2.4.9.8 Example Response

```
{  
  "authorizationCode": "FCP-987654321",  
  "orderId": "ORD-0001",  
  "status": "AUTHORIZED",  
  "transactionType": "AUTHORIZATION",  
  "amount": 15.00,  
  "settlementStatus": "NOT_SETTLED",  
  "responseCode": "00",  
  "responseMessage": "Authorization exists"  
}
```

2.5 FCP Clearing Files generated by CBO

2.5.1 Purpose

The objective of the FCP Clearing Files is to provide a **reconciliation mechanism** between the Central Back Office (CBO) and each Fuel Card Provider (FCP).

This includes the generation of:

- **Daily** files with details
- **Payment Demand** files with aggregated values

Clearing files are **informational and accounting artifacts**, not authorization mechanisms. All transactions included in the files have already been processed via the FCP APIs (authorization, capture, refund).

The clearing process ensures that:

- All fuel card transactions processed in CBO are reported to the corresponding FCP
- Fuel Card Providers can reconcile transactions against their internal systems
- Financial settlement processes can be triggered on the FCP side

The generation of daily clearing files is executed as a **daily batch process**, for the transactions of the **previous day**.

The generation of payment demand files is executed as a **batch process 2 times per month**, and each file includes the records that have not been informed previously.

2.5.2 Overview

The solution consists of:

- A **data service** that collects all fuel card transactions processed in CBO
- A **file generation component** that builds clearing files per FCP
- A **delivery mechanism** to send files to FCPs (e.g., SFTP or other agreed channel)

For each Fuel Card Provider:

- One or more clearing files are generated per day
- Files are grouped by Fuel Card Provider
- If the number of transactions exceeds the maximum file size, **multiple files will be generated for the same day**.
- If there are no FCP transactions for the daily or biweekly files, the files will be generated with no transaction records (empty files).

2.5.3 Types of Clearing Files

In this implementation, **a single clearing file type is defined per Fuel Card Provider (FCP)**.

File Type	Description
FCP1	Contains all daily fuel card transactions processed in CBO
FCP2	Payment demand: contains aggregated information per FCP.

Key principles:

- All transactions are included in the same file regardless of origin
- There is no separation by contract type
- Transaction semantics (debit, credit, refund, adjustment) are defined at record level

2.5.4 File Naming Convention

All clearing files must follow the naming convention:

FCP1_<FCP_ID>_<TIMESTAMP>_<SEQUENCE>.fcc

Example:

Daily files:

FCP1_XYZ_20260603123000_000001.fcc

FCP1_XYZ_20260603123000_000002.fcc

Aggregated files:

FCP2_XYZ_20260603123000_000001.fcc

Where:

- **FCP1** = fixed identifier for daily clearing files
- **FCP_ID** = Fuel Card Provider identifier
- **TIMESTAMP** = file generation date and time
- **SEQUENCE** = sequential number
- **FCP2** = fixed identifier for Payment Demand files

Sequence rules:

- Starts at 000001
- Incremented for each generated file
- Sequence is separated by FCP and per file type (2 sequences per FCP)

2.5.5 File Formats

The files exchanged between the Central Back Office (CBO) and the Fuel Card Providers (FCPs) are **flat files (text files)** with **variable record length**.

Variable record length means that the length of each record may differ depending on the record type (e.g., header, transaction record, trailer).

Each record is separated by a line break and terminated by:

- **CR LF** (Carriage Return + Line Feed)

2.5.6 Field Formats

All files are encoded using **Unicode UTF-8**.

The solution must support:

- International characters in textual fields (e.g., names, license plates)
- Extended Latin characters required in European contexts

The following notation is used to define field formats:

Format	Description
Cn	Character field with length n . Fields are left-aligned and padded with spaces. Empty fields contain only spaces.
Nn	Numeric field with length n . Fields are right-aligned and padded with leading zeroes. Empty fields contain only zeroes.
Nn.d	Numeric field with decimals, where n is the total number of digits and d is the number of decimal places.

Additional rules:

- Decimal values do not include explicit separators (no comma or dot)
- The last d digits represent the decimal fraction

Example:

- Format: **N9.2**
- Value: 000120050 → interpreted as **1,200.50**

2.5.7 Date and Time Formats

Unless otherwise specified, the following formats apply:

- **Timestamp:**

YYYYMMDDHHMMSS

- **Date:**

YYYYMMDD

- **Time:**

HHMM

All timestamps and dates provided by CBO are expressed in:

- **UTC timezone**, unless explicitly agreed otherwise with the Fuel Card Provider

2.5.8 Mandatory Fields

Fields marked as **Required = Yes** must always contain a valid value.

- Valid values are defined in the corresponding field descriptions
- Empty or invalid values are not allowed for mandatory fields

Fields marked as **Required = No**:

- May be left empty
- Must follow formatting rules (spaces or zeroes depending on type)

Important:

- Even if a field is optional, it may become **conditionally required** depending on the transaction context
- Such conditions are defined in the business rules and validation logic of the system

Internal

2.5.9 File Structure

Each clearing file consists of three types of records:

1. **Header Record (R1)**
2. **Detail**
 - a. **For Daily files: Transaction Records (R2)**
 - b. **For Payment Demand files: Aggregated Records (R3)**
3. **Trailer Record (R4)**

2.5.9.1 Daily files

2.5.9.1.1 Header Record (R1)

The header record contains metadata about the file.

Parameter	Required	Type	Description
RECORD_TYPE	Yes	String (C2)	Fixed value: R1
FILE_TYPE	Yes	String (C4)	Fixed value: FCP1
SENDER_ID	Yes	String (C10)	Identifier of CBO
RECIPIENT_ID	Yes	String (C10)	Identifier of the FCP
FILE_CREATION_DATE	Yes	String (C8)	Format: YYYYMMDD
FILE_CREATION_TIME	Yes	String (C4)	Format: HHMM
SEQUENTIAL_NUMBER	Yes	Numeric (N6)	Incremental file counter
CURRENCY	Yes	String (C3)	ISO currency code

2.5.9.1.2 Transaction Record (R2)

Each **transaction** processed during the previous day is represented as a transaction record.

Parameter	Required	Type	Description
RECORD_TYPE	Yes	String (C2)	Fixed value: R2
CARD_IDENTIFIER	Yes	String (C25)	Tokenized or masked fuel card number
CARD_EXPIRY_DATE	Yes	String (C6)	Expiration date of the fuel card (format: YYYYMM)
TRANSACTION_DATE	Yes	String (C8)	Date of the transaction (format: YYYYMMDD)
TRANSACTION_TIME	Yes	String (C4)	Time of the transaction (format: HHMM)

Parameter	Required	Type	Description
TRANSACTION_AMOUNT	Yes	Numeric (N17.2)	Total amount of the transaction
AUTHORIZATION_CODE	No	String (C10)	Reference code returned by the FCP for approved transactions
DEBIT_CREDIT_INDICATOR	Yes	String (C1)	Indicates whether the transaction is Debit (D) or Credit (C)
COLLECTION_ORDER_ID	Yes	Numeric (N25)	Reference number associated with the transaction
ACCOUNTING_DATE	No	String (C8)	Accounting date of the transaction (format: YYYYMMDD)

Important considerations:

- Both **debit and credit transactions** must be included
- Refunds are represented as credit transactions

2.5.9.1.3 Trailer Record (R4)

The trailer record ensures file completeness and integrity.

Parameter	Required	Type	Description
RECORD_TYPE	Yes	String (C2)	Fixed value: R4
RECORD_COUNTER	Yes	Numeric (N9)	Number of records in the file
TOTAL_AMOUNT	Yes	Numeric (N17.2)	For Daily files: • Total TRANSACTION_AMOUNT for all the records.
CHECKSUM	Yes	Numeric (N16.2)	Sum of all transaction amounts

Checksum rules:

- Calculated as the **sum of absolute transaction amounts**
- Debit and credit transactions are treated equally for the calculation

2.5.9.2 Payment Demand files

2.5.9.2.1 Header Record (R1)

The header record contains metadata about the file.

Parameter	Required	Type	Description
RECORD_TYPE	Yes	String (C2)	Fixed value: R1

Internal

Parameter	Required	Type	Description
FILE_TYPE	Yes	String (C4)	Fixed value: FCP2
SENDER_ID	Yes	String (C10)	Identifier of CBO
RECIPIENT_ID	Yes	String (C10)	Identifier of the FCP
FILE_CREATION_DATE	Yes	String (C8)	Format: YYYYMMDD
FILE_CREATION_TIME	Yes	String (C4)	Format: HHMM
SEQUENTIAL_NUMBER	Yes	Numeric (N6)	Incremental file counter
CURRENCY	Yes	String (C3)	ISO currency code

2.5.9.2.2 Aggregated Record (R3)

Each day processed during the bi-weekly process is represented as a record.

Parameter	Required	Type	Description
RECORD_TYPE	Yes	String (C2)	Fixed value: R3
DAILY_SEQUENCE_NUMBER	Yes	Numeric (N6)	Daily Sequence Number (e.g. 000001). This corresponds to the sequence number of the daily files.
DAILY_FILE_DATE	Yes	String (C8)	Daily file generation date and time. Format: YYYYMMDD
DAILY_FILE_TIME	Yes	String (C4)	Daily file generation time (format: HHMM)
ACK_DAILY_FILE_DATE	Yes	String (C8)	Daily file Acknowledgement date and time. Format: YYYYMMDD
ACK_DAILY_FILE_TIME	Yes	String (C4)	Daily file Acknowledgement time (format: HHMM)
DAILY_FILE_TOTAL_TRANSACTIONS	Yes	Numeric (N9)	Number of records
DAILY_FILE_TOTAL_AMOUNT	Yes	Numeric (N17.2)	Total amount of all the transactions
DAILY_FILE_TOTAL_ACK_OK_TRANSACTIONS	Yes	Numeric (N9)	Number of ack ok records

Parameter	Required	Type	Description
DAILY_FILE_TOTAL_ACK_OK_AMOUNT	Yes	Numeric (N17.2)	Total amount of ack ok transactions

2.5.9.2.3 Trailer Record (R4)

The trailer record ensures file completeness and integrity.

Parameter	Required	Type	Description
RECORD_TYPE	Yes	String (C2)	Fixed value: R4
RECORD_COUNTER	Yes	Numeric (N9)	Number of records in the file
TOTAL_AMOUNT	Yes	Numeric (N17.2)	For Payment Demand Aggregated files: Total DAILY_FILE_TOTAL_ACK_OK_AMOUNT for all the records.
CHECKSUM	Yes	Numeric (N16.2)	Sum of all transaction amounts

Checksum rules:

- Calculated as the **sum of absolute transaction amounts**
- Debit and credit transactions are treated equally for the calculation

2.5.10 Transaction Classification

Since file-level separation is not applied, transaction classification is handled at the record level.

The following fields define transaction semantics:

- Debit/Credit Indicator** → identifies whether the transaction is a charge or a refund

This approach ensures:

- Clear interpretation of each transaction
- Consistent reconciliation by the FCP

2.5.11 Delivery Mechanism

Supported options include:

- SFTP**
- Secure file transfer mechanisms

Delivery requirements:

- Files must be delivered **daily per FCP**
- Retry mechanisms must be implemented in case of delivery failure

Internal

- Delivery status must be **logged and auditable**

2.5.12 Acknowledgement Mechanism for Daily files

We are exposing an endpoint (API) for the FCP to send CBO the acknowledgment for the **daily** clearing file received. All records should be informed (not only the ones with error).

There is no acknowledge mechanism for Payment Demand files.

POST <https://<hostname>:<port>/api/acknowledgement>

2.5.12.1 Request parameters

Parameter	Required	Type	Description
SequenceId	Mandatory	Integer	Identifies the sequential number of the clearing file.
FCPIId	Mandatory	Integer	Identifies the Fuel Card Provider.
AckTimestamp	Mandatory	Timestamp	Date of the acknowledgement.
Acknowledgements	Mandatory	Acknowledgement Array	Contains the array of acknowledgements.

2.5.12.1.1 Acknowledgement

This structure contains the Acknowledgement

Parameter	Required	Type	Description
AuthorizationCode	Mandatory	String	Reference code returned by the FCP for approved transactions.
CollectionOrderId	Mandatory	Integer	Reference number associated with the transaction.
AckCode	Mandatory	Byte	Acknowledgement Code: contains the reason for accepting or rejecting the identified message. Allowed values: • AckOk (1) • AckNoOk (2)
AckError	Optional	AcknowledgementError	If its ok, the value is null.

2.5.12.1.1.1 AcknowledgementError

This structure contains the AcknowledgementError

Parameter	Required	Type	Description
Code	Mandatory	Integer	Acknowledgement Error Code class: • 0 - OK. The record has been processed successfully. • >= 100 - Error. There was an error with the record.
Text	Optional	String	Acknowledgement Error Text: contains detailed textual information or other error codes.

Error Codes

Code	Text
201	Invalid FCP Id
202	Invalid Sequence Id
203	Invalid Authorization Code
204	Invalid Order Id
205	Invalid card identifier
206	Invalid card expiration date
207	Invalid record counter
208	Invalid checksum
401	Unauthorized
403	Unauthorized access
500	Internal Server Error

2.5.12.2 Response parameters

Parameter	Required	Type	Description
AckErrors	Optional	AcknowledgementError array	List of errors found during the processing of the request. Structure is specified in section AcknowledgementError above.

2.5.12.3 Example Request 1 (All records OK)

```
{
  "SequenceId": 12345,
  "FCPIId": 1001,
  "AckTimestamp": "20260603083000",
  "Acknowledgements": [
    {
      "AuthorizationCode": "AUTH123456",
      "CollectionOrderId": 987654321,
      "AckCode": 1,
      "AckError": {
        "Code": 0,
        "Text": "OK"
      }
    },
    {
      "AuthorizationCode": "AUTH123457",
      "CollectionOrderId": 987654322,
      "AckCode": 1,
      "AckError": {
        "Code": 0,
        "Text": "OK"
      }
    }
  ]
}
```

2.5.12.4 Example Request 2 (Mixed OK + Errors)

```
{
  "SequenceId": 12345,
  "FCPIId": 1001,
  "AckTimestamp": "20260603083000",
  "Acknowledgements": [
    {
      "AuthorizationCode": "AUTH123456",
      "CollectionOrderId": 987654321,
      "AckCode": 1,
      "AckError": {
        "Code": 0,
        "Text": "OK"
      }
    },
    {
      "AuthorizationCode": "AUTH999999",
      "CollectionOrderId": 987654999,
      "AckCode": 2,
      "AckError": {
        "Code": 203,
        "Text": "Invalid Authorization Code"
      }
    },
    {
      "AuthorizationCode": "AUTH123458",
      "CollectionOrderId": 0,
      "AckCode": 2,
      "AckError": {
        "Code": 204,
        "Text": "Invalid Order Id"
      }
    }
  ]
}
```

2.5.12.5 Example Response 1 (No errors)

```
HTTP 200 OK

{
  "AckErrors": []
}
```

2.5.12.6 Example Response 2 (Validation Errors in Request)

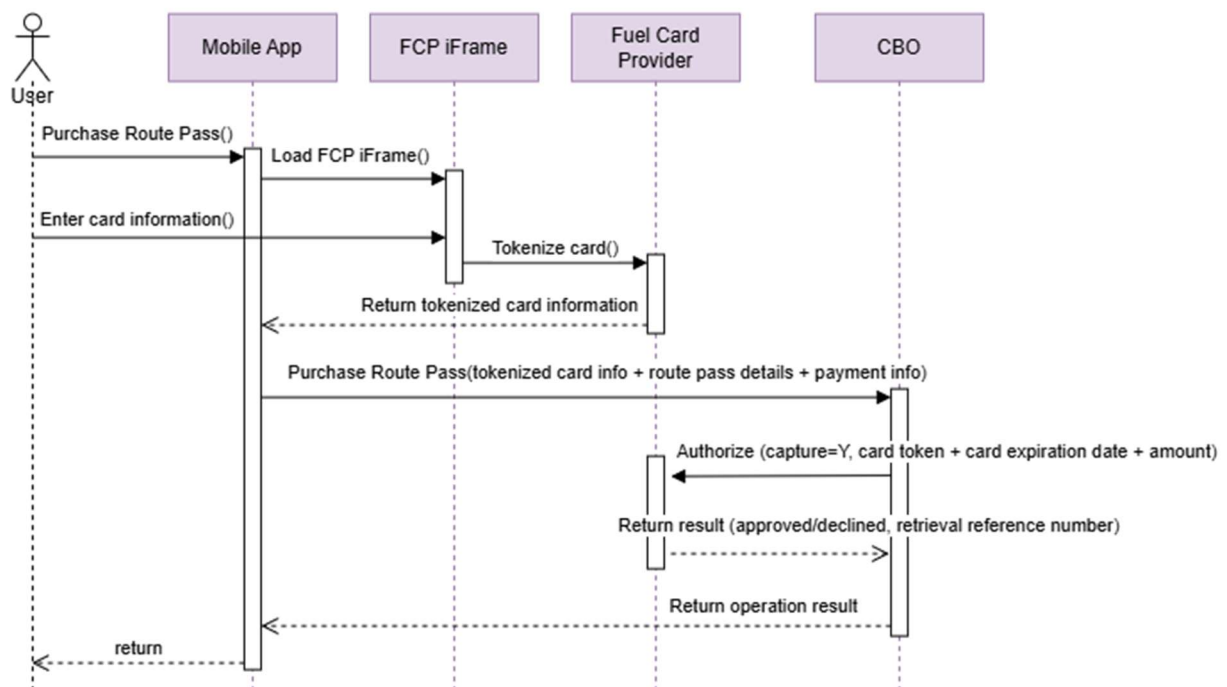
```
HTTP 200 OK
```

```
{
  "AckErrors": [
    {
      "Code": 201,
      "Text": "Invalid FCP Id"
    },
    {
      "Code": 202,
      "Text": "Invalid Sequence Id"
    }
  ]
}
```

3 Examples of Payment Flows

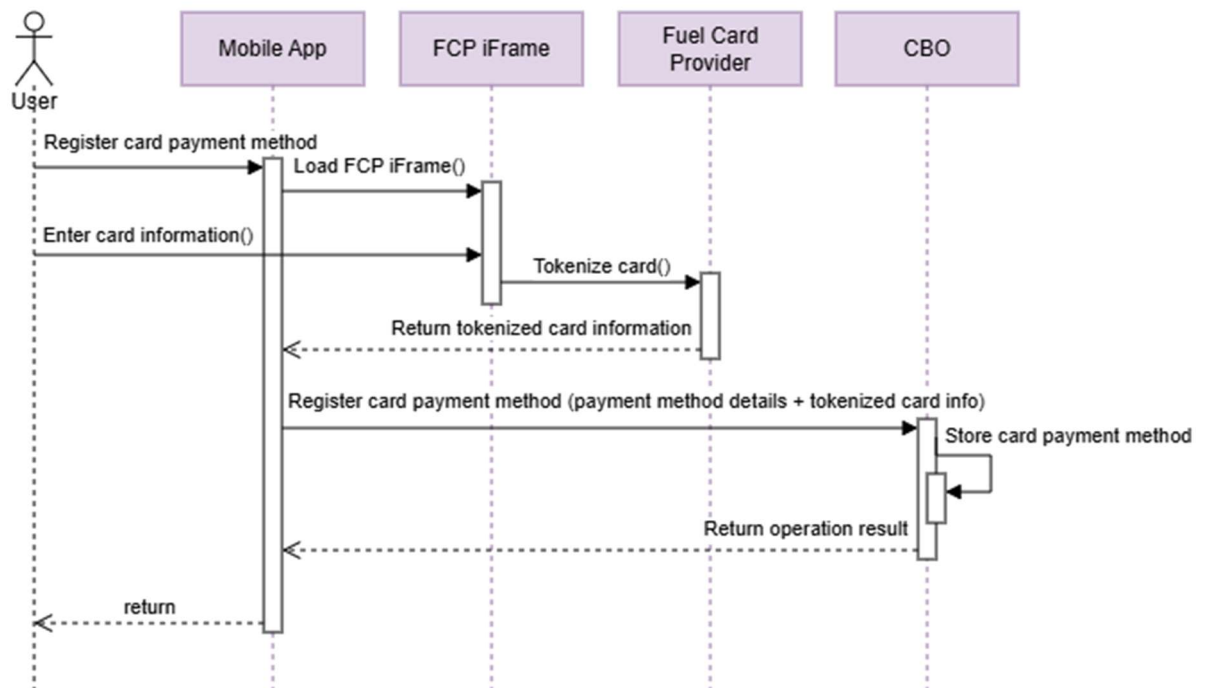
3.1 Fuel card payment without registered means of payment

Fuel card information must be entered for every purchase (no card is stored on the user account).



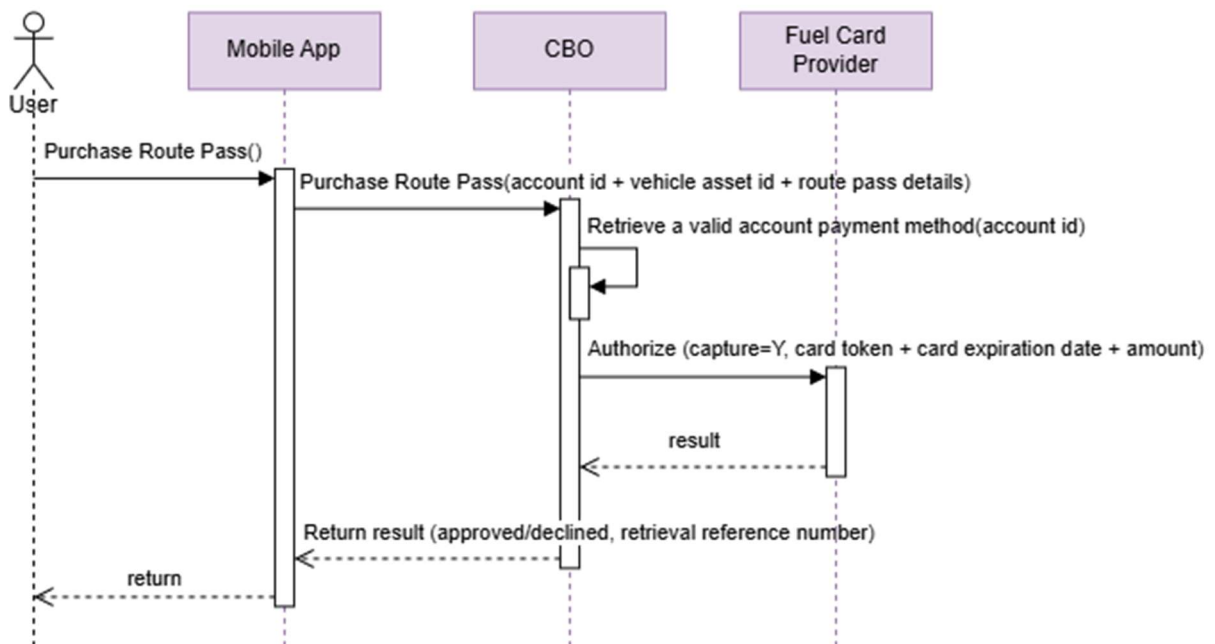
3.2 Registration of a fuel card as a payment method for registered users

Fuel card information is entered only once, and the tokenized card info is stored on the user account.



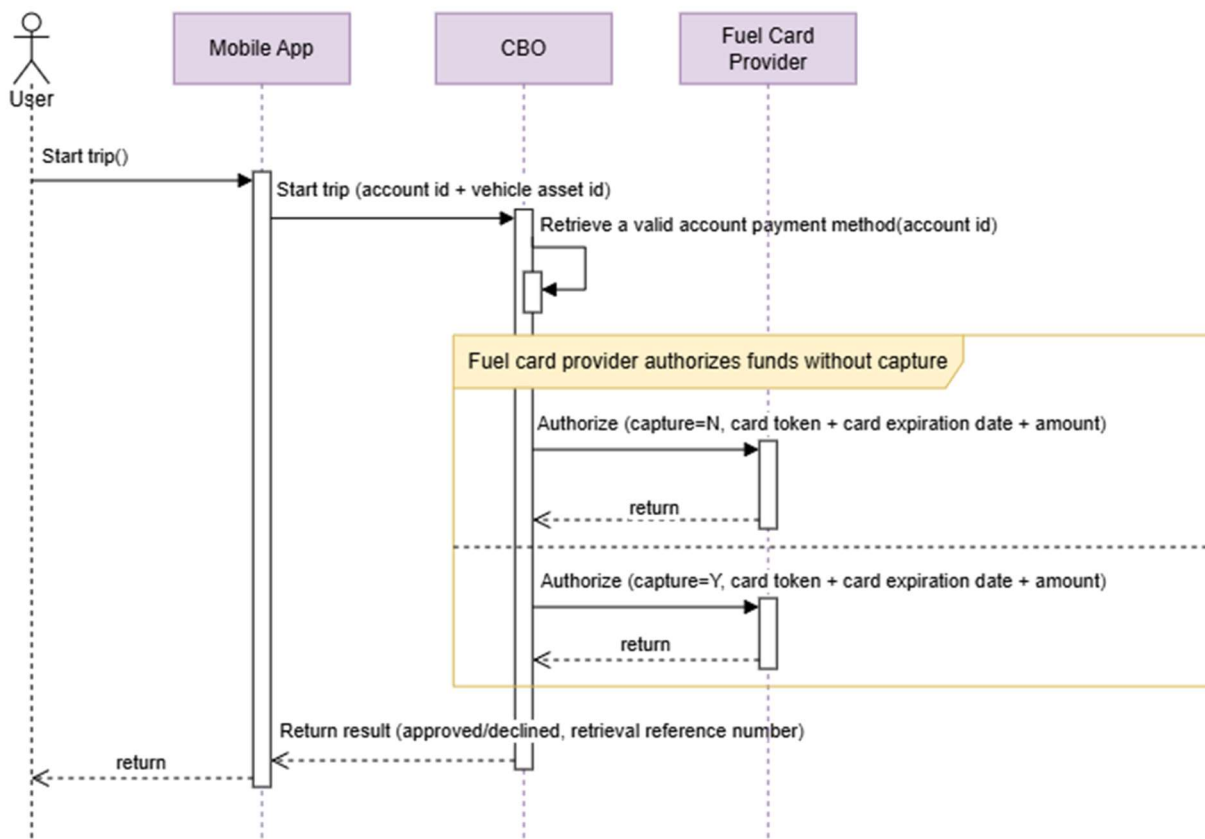
3.3 Payment of a Route Pass by a registered user with post-pay account

The user purchases the route pass using a stored fuel card payment method



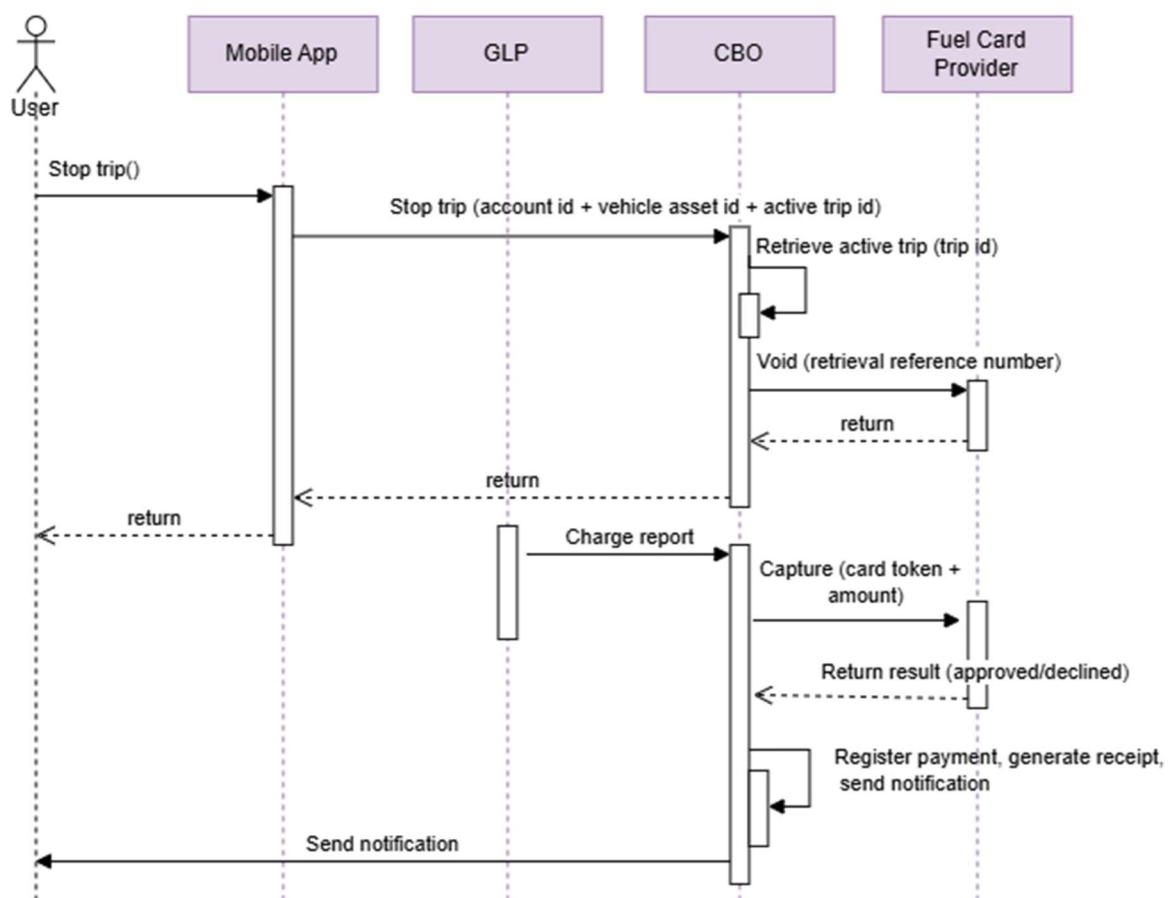
3.4 Start a trip by a registered user with post-pay account- Fee Reservation

The user starts a trip and reserves funds (for the longest possible route) using a stored fuel card payment method.



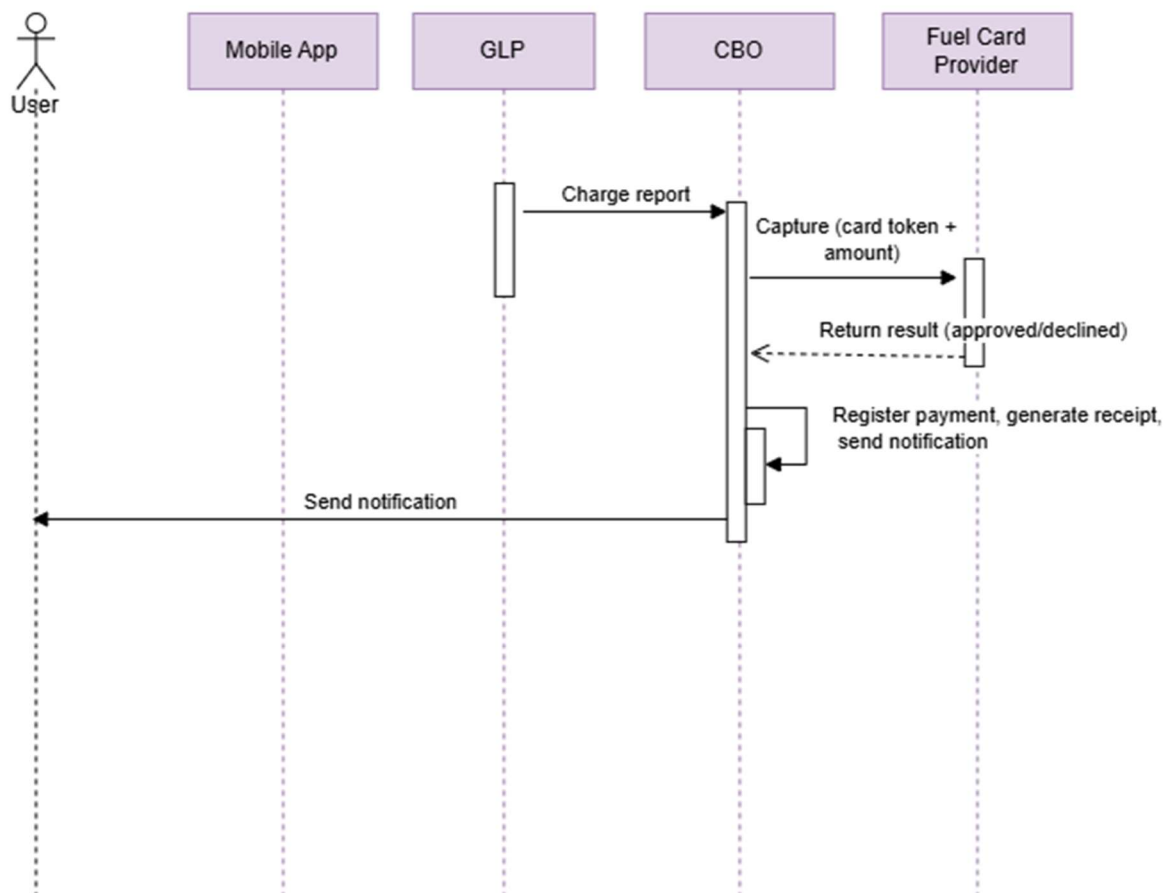
3.5 End trip by a registered user with post-pay account

The user ends the trip, and the reserved funds are canceled, regardless of whether they were captured. CBO performs the payment of the toll amount using a stored fuel card payment method through a background process responsible of processing charge reports received from GLP



3.6 Posting toll for an user with OBU (and Fuel Card as an automatic payment method)

The trip ends when GLP stops receiving positions associated to tolling sections. Then GLP sends Charge report to CBO. CBO performs the payment of the toll amount using a stored fuel card payment method through a background process responsible of processing charge reports received from GLP.



- END OF DOCUMENT -