Implementation Guide

Part 4-50-1 Issuer Initiated Closed Loop Payment API

18 October 2024

Version 1.1

Document Summary

This document provides guidance for API based payments for closed loop cards. The scope of version 1 of this standard is for issuer-initiated payments as might be used to support customer payment using an issuer provided mobile payment application. Although the primary use of this API will be mobile payment at this time, it is intended that this API will be useable in a broader context wherever API based closed loop payment is required. Future extensions to this API will provide broader support for non-mobile based payments.

While the Payment APIs might be also applicable for forms of Site to Host payment authorizations, the initial scope is central integration that can apply to payments originated at site, or over the internet as eCommerce.

The initial focus is about the payment authorization, so this is not yet a complete implementation.

Security requires additional analysis; initial assumptions are to leverage encryption in transit TLS1.2, Oauth2 for API authentication.

Contributors

Paolo Franco Magnoni, Shell
Ian S. Brown, IFSF
Gonzalo Fernandez Gomez, OrionTech
Lucia Marta Valle, OrionTech

Revision History

Revision Date	Revision Number	Revision Editor(s)	Revision Changes
23 May 2022	V1.0	Gonzalo Fernandez Gomez, Lucia M. Valle OrionTech Paolo Magnoni, Shell Ian S Brown, IFSF	First release
5 Aug 2023	V1.0	Ian Brown, IFSF	Title change to Part 4-50-1 Issuer Initiated Closed Loop Payment API. Version number not changed
18 October 2024	V1.1	Lucia M. Valle	6 – Implementation Details: The name of the redoc was changed to replace bundle suffix by redoc 3.4 Use Cases Diagrams: reserve response changed from 200 to 201 in all diagrams A new 6.3 Error handling section was added Updated schema definitions: • Updated format of additionalProductCode field to be an object with posCodeType as enumerated field • Corrected typos Updated API definitions: • Added support for operationId field and support for additional HTTP return codes

Copyright Statement

Table of Contents

Project

Electronic Business to Business

Subtitle

Merchant Host to Host

1 Introduction

Payment APIs enable extending business opportunities and new channels of sales and payment acceptance. As the payment industry has diversified Method of Payments, channels of acceptance and technologies, IFSF has the opportunity to define modern interoperability standards for Fuel Retailers and B2B payment offers.

1.1 Overview

The proposed draft of Host-to-Host APIs uses message calls that are equivalent to the Authorization Request, Financial Request and Financial Advice used in the IFSF ISO8583 based H2H standard. For clarity, however, it is not intended that this API is a direct conversion of ISO8583 messages into JSON message structures nor is it intended that this API be completely interoperable with ISO8583. It may be convertible in some limited use cases, but this is not guaranteed.

This proposal enables reusing backend logic and co-existence of traditional and APIs based integration. Limiting the number of APIs to accomplish a payment transaction, also reduces the potential exceptions that might occur executing the payment process; it eliminates the necessity of a stateful handling of multiple APIs, before completing mandatory and optionally information required for authorization and capture of the payment transaction.

This approach assumes that the application server accepting the payment requests and sending the Payment APIs has the capability to manage the status of the requests till their completion (responsibility that might reside in an eCommerce platform, or ultimately in a payment terminal connecting into this server - depending on the use case).

1.2 Business Propositions

APIs enable multiple Business propositions:

- **Card not present solutions, for payment acceptance on the spot (at the Retail site). Customer and/or vehicle present payment execution.**
  Different technologies might apply to identify the entity and authenticate for payment authorization: QRcodes, Vehicle recognition, SmartDevice digital payment. Over the air or proximity payment might apply, depending on the technology.

- **Internet payment, by traditional eCommerce or ubiquitous mCommerce.**
  Enablement of payment by Card on File, or by other token form, used on ecommerce WEB sites, or mCommerce Apps.
  The enablement of payment of goods or services, of delivery to customer in various forms, would extend the traditional brick and Mortar shop (Fuel Retail Site).

1.3 Benefits

Benefits of Payment APIs are:

- Business development: enable new channels of purchase and payment – e.g., eCommerce
- Card-less payment: enable various forms of token-based payment acceptance.
- Simpler integration: easier to implement integration, more open interoperability.
- Cheaper development: leverage common industry skills for development of payment.
- Cheaper platforms: leverage platforms not fully dedicated to payment, that can operate outside of PCI Data Security Standard scope (though security remains a primary concern).
- Faster and agile flexibility: develop faster, leverage DevOps methodologies.
- Use of preauthorized (at Pump or at Pos transactions is transparent for the merchant; messages will be the same for both scenarios.
- Allows non petrol specialized participants to interact with sites and be dispensed in an open, modern, and standard way.

2 Architecture

2.1 Premises

- All uses cases are Issuer Initiated transactions only. It will be extended to merchant initiated in the next release.
- The card issuer is trusted to authorise the card payment
- The “Merchant Host” is connected to the “Issuer Host” through a trusted connection (H2H)
- The “Issuer Host” informs the “Merchant Host” that there is a customer at a pump or other location-based point of interaction, that the customer has been authorised to use their fuel card for a certain amount of money and the issuer guarantees payment. The “Merchant Host” therefore knows they can let the customer fill up. The “Merchant Host” does not need to know anything about the card owner, just non sensitive information to uniquely identify the transaction in case of dispute.
- The “Merchant Host” completes the transaction and send the details to the “Issuer Host”.
- Minimized shared information between merchant and issuer.
- Share bulk configuration information.
    - Minimize information shared with other organizations.
    - Notify information changes (if applicable) through SSE.
- Use simplified transaction schema for both fuels and non-fuels
- Use Open Retailing data dictionary as much as possible

2.2 Host to Host Integration

The explanation and diagram below show a high-level scope of the Host-to-Host APIs.

Conceptually, it is not a Fueling Point Approval, but it is a communication of an approved payment by the Issuer.

The Issuer takes responsibility for the financial payment and integrates to the merchant organization.

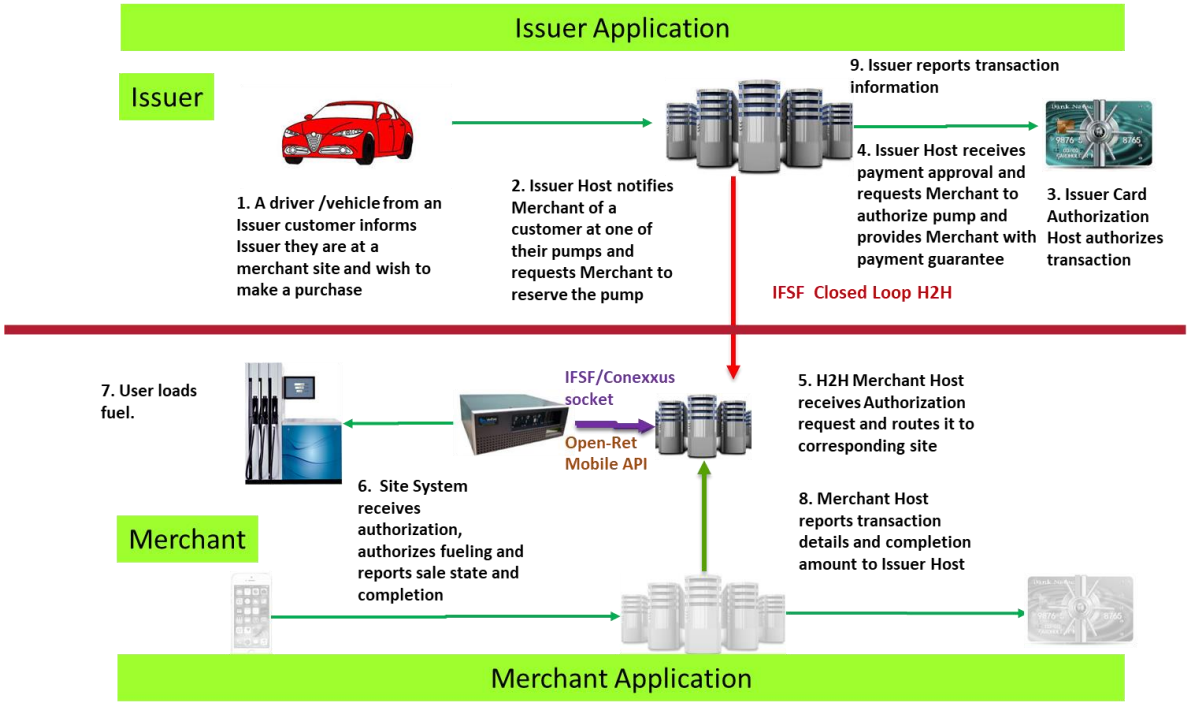
The Merchant is responsible of returning the proper transaction to the Issuer (Management of POS - Merchant Host flow and exceptions is out of the scope of the API). The Merchant deals with individual merchant sites if the merchant operates multiple sites.

The main responsibilities of an “Issuer Host” are:

- Interact with the Issuer applications to acknowledge that there is customer at a site to make a purchase
- Initiate a transaction with the “Merchant Host” providing the POI (Point Of Interaction)
- Provide the authorization to the “Merchant Host” for a certain amount and detailed list of approved products
- Complete the transaction sending the receipt to the “Merchant Host” and receiving the combined transaction receipt which completes the process on both sides

The main responsibilities of a “Merchant Host” are:

- Receive the transaction initiation request from the “Issuer Host” and interact with the Merchant applications to create the transaction
- Receive the authorization from the “Issuer Host” and manage the purchase notifying the “Issuer Host” of the evolution in the state of the transaction
- Inform the transaction details to the “Issuer Host” every time they are requested
- Provide the transaction receipt upon reception of the EFT receipt from the “Issuer Host” and set the transaction state as completed



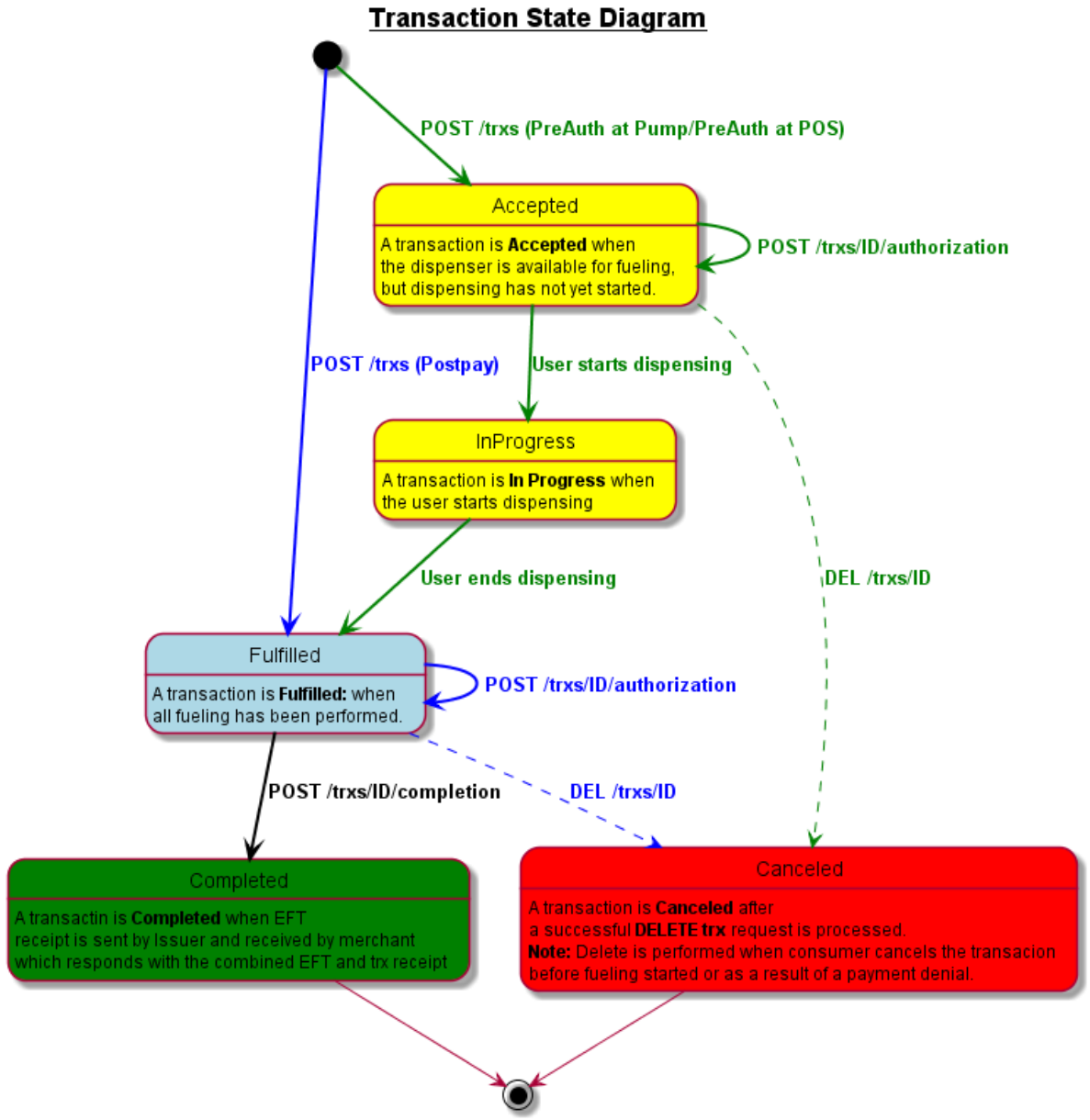
1. A driver/vehicle from an Issuer customer informs Issuer they are at a merchant site and wish to make a purchase
2. Issuer Host notifies Merchant of a customer at one of their pumps and requests Merchant to reserve the pump
3. Issuer Card Authorization Host authorizes transaction
4. Issuer Host receives payment approval and requests Merchant to authorize pump and provides Merchant with payment guarantee
5. Merchant Host receives Authorization request and routes it to corresponding site
6. Site System receives authorization, authorizes fueling and reports sale state and completion
7. User loads fuel
8. Merchant Host reports transaction details and completion amount to Issuer Host
9. Issuer reports transaction information

3 Process Flows and Use Cases

The different transaction and payment states, processing flows and use cases that are supported by Host-to-Host are the ones briefly outlined below:

3.1 Transaction States

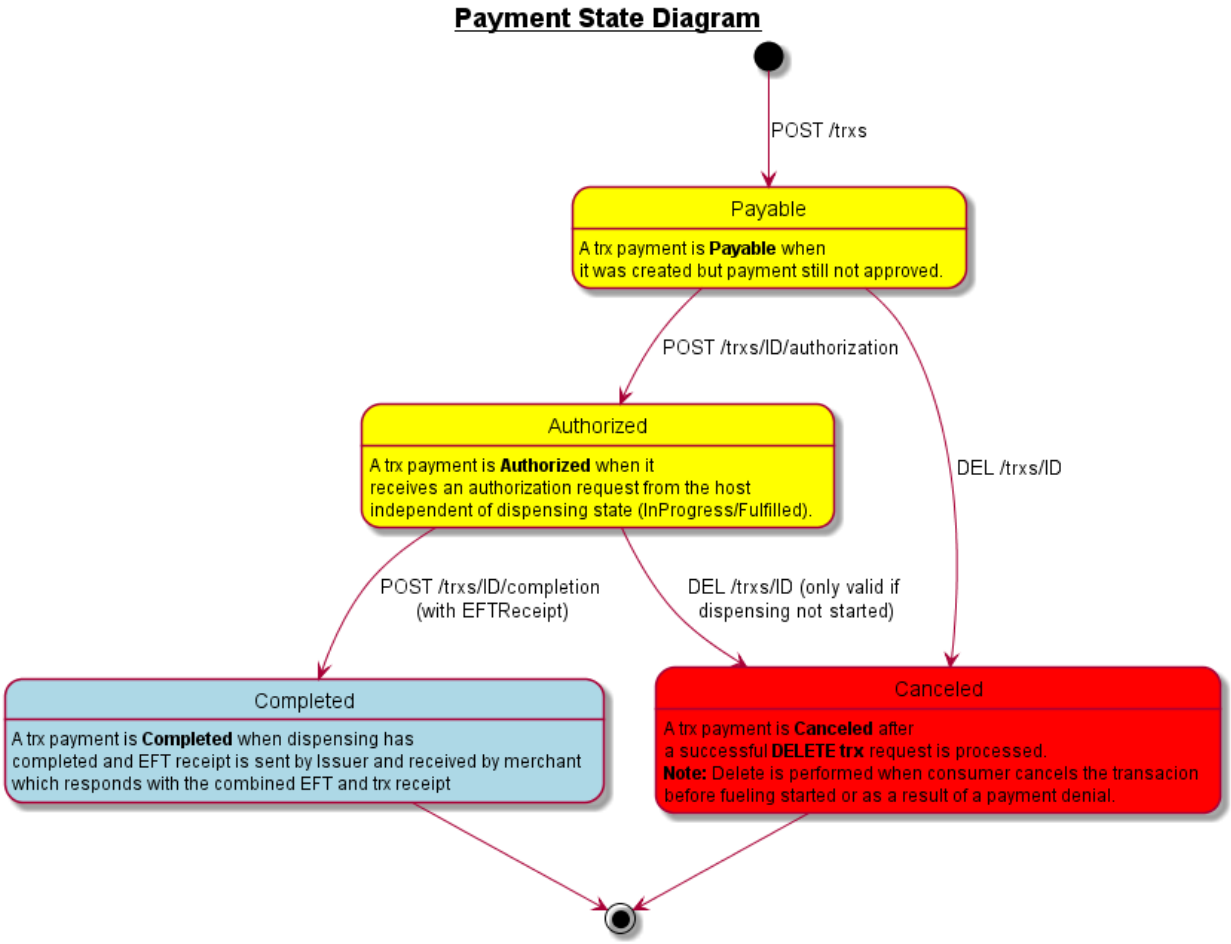
The diagram below shows the transaction states at the merchant and how they will evolve through the API calls from accepted to completed. Every time the transaction state changes, the issuer is notified via an event.



Ref: Post Paid Transaction States / Preauth Transaction States

3.2 Payment States

The diagram below shows the payment states at the issuer and how they will evolve through the API calls from payable to completed.

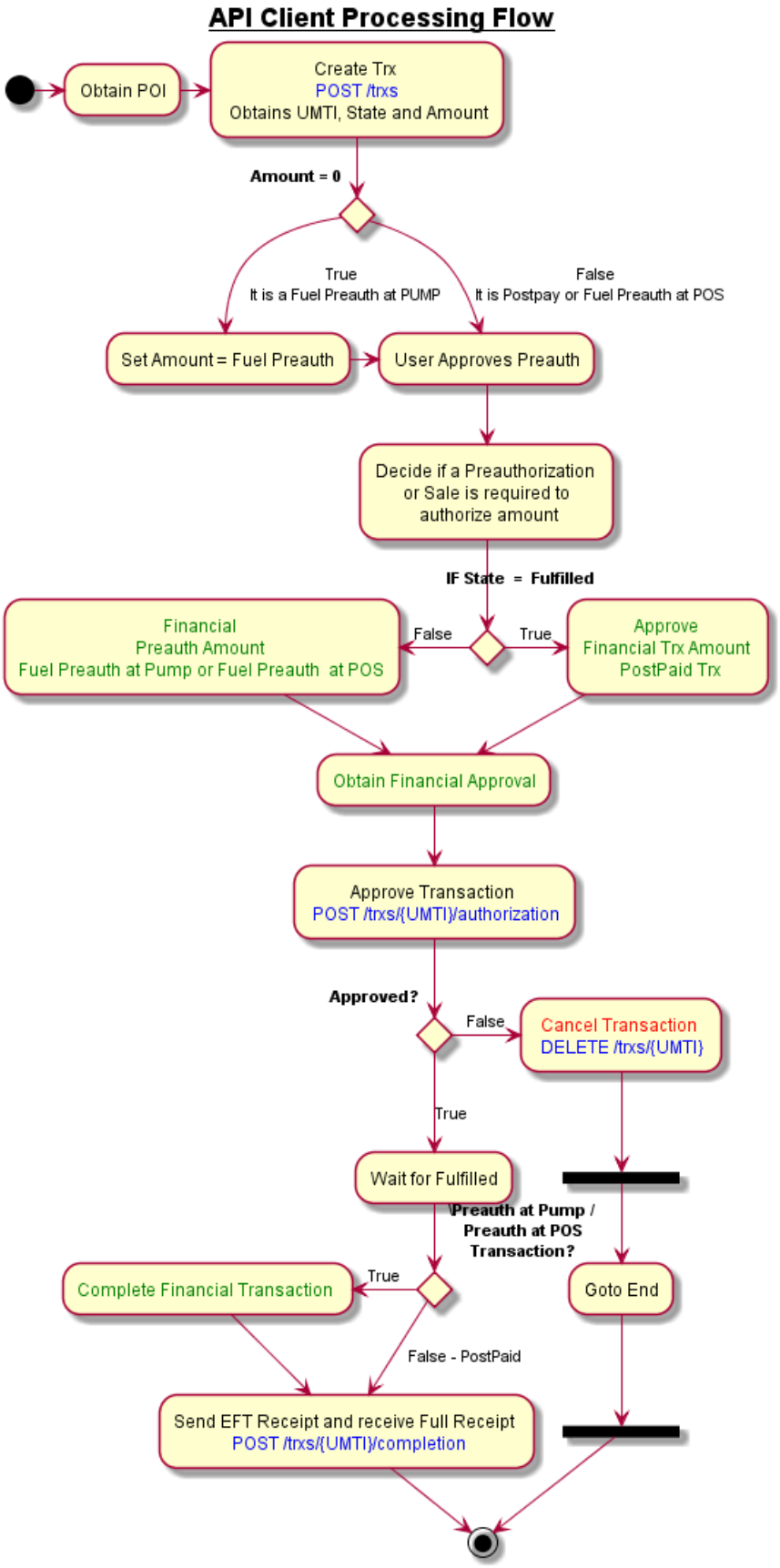


3.3 Processing Flow

The diagram below shows the general processing flow including the three uses cases described below:

- Pre-Authorization at PUMP
- Pre-Authorization at POS
- Post-Pay

This flow helps the API clients to identify the different scenarios and steps to follow depending on the authorized amount, transaction state and payment state. It also includes the exception of financial approval not obtained and the subsequent transaction cancelation.



3.4 Use cases diagrams

This section includes detailed diagrams related to the three different use cases. Additionally, for each of the cases there is a diagram showing the exceptions to the normal flow.

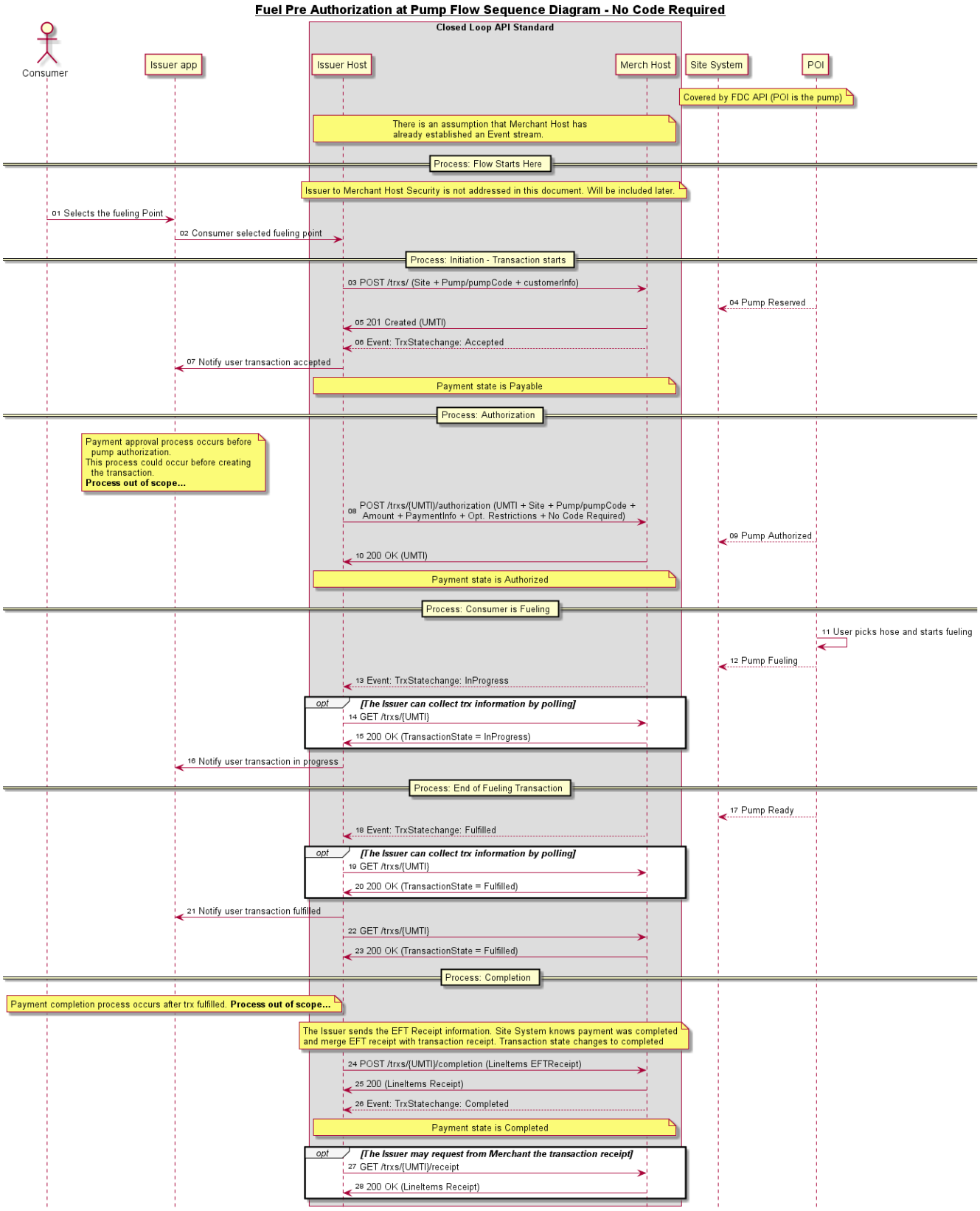
In the case of Pre-Authorization at Pump, an additional case shows the alternative of a validation code to be displayed at the customer device. The code is entered by the customer at the site and validated by the site or Merchant Host.

3.4.1 Pre-Authorization at Pump

The steps below take place in the case of Pre-Authorization at Pump

- Issuer consumer selects fueling position
- Process is initiated and transaction generated in “Accepted” state. Payment state is “Payable” at this point
- Financial approval is requested
- Payment authorization process takes place and payment state changes to “Authorized”
- Consumer starts fueling and transaction state changes to “InProgress”
- Fueling is completed and transaction state changes to “Fulfilled”
- Transaction details are obtained from the “Merchant Host”
- Payment completion process takes place after transaction “Fulfilled”
- “Issuer Host” sends the EFT receipt for the “Site System” to merge with transaction receipt and “Issuer Host” receives the combined transaction receipt
- Both transaction state and payment state changes to “Completed” indicating the finalization of the process on both sides (issuer and merchant)

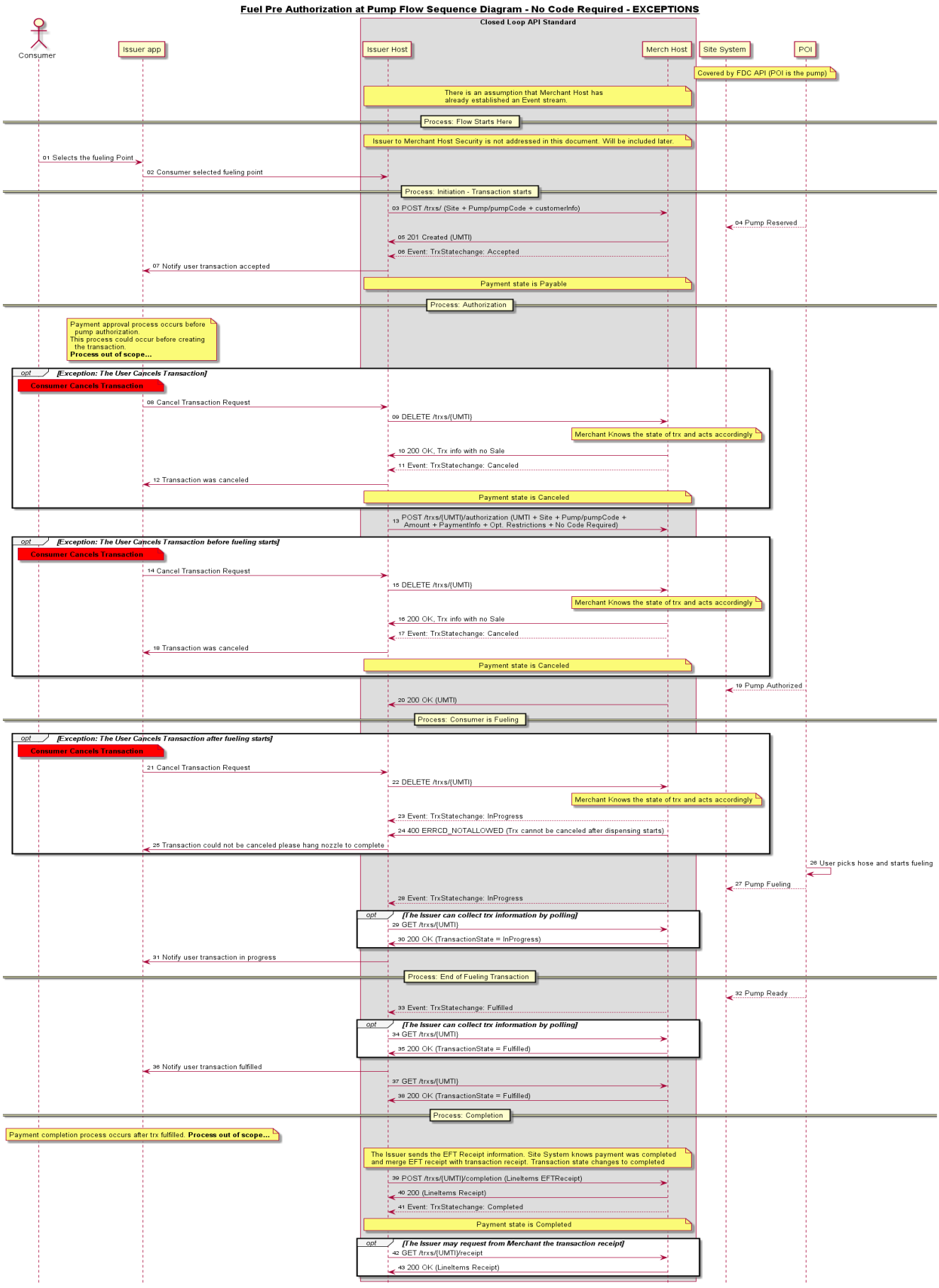
3.4.1.1 Pre-Authorization at Pump – Normal flow



3.4.1.2 Pre-Authorization at Pump– Exception flow

The exceptions considered in this case are:

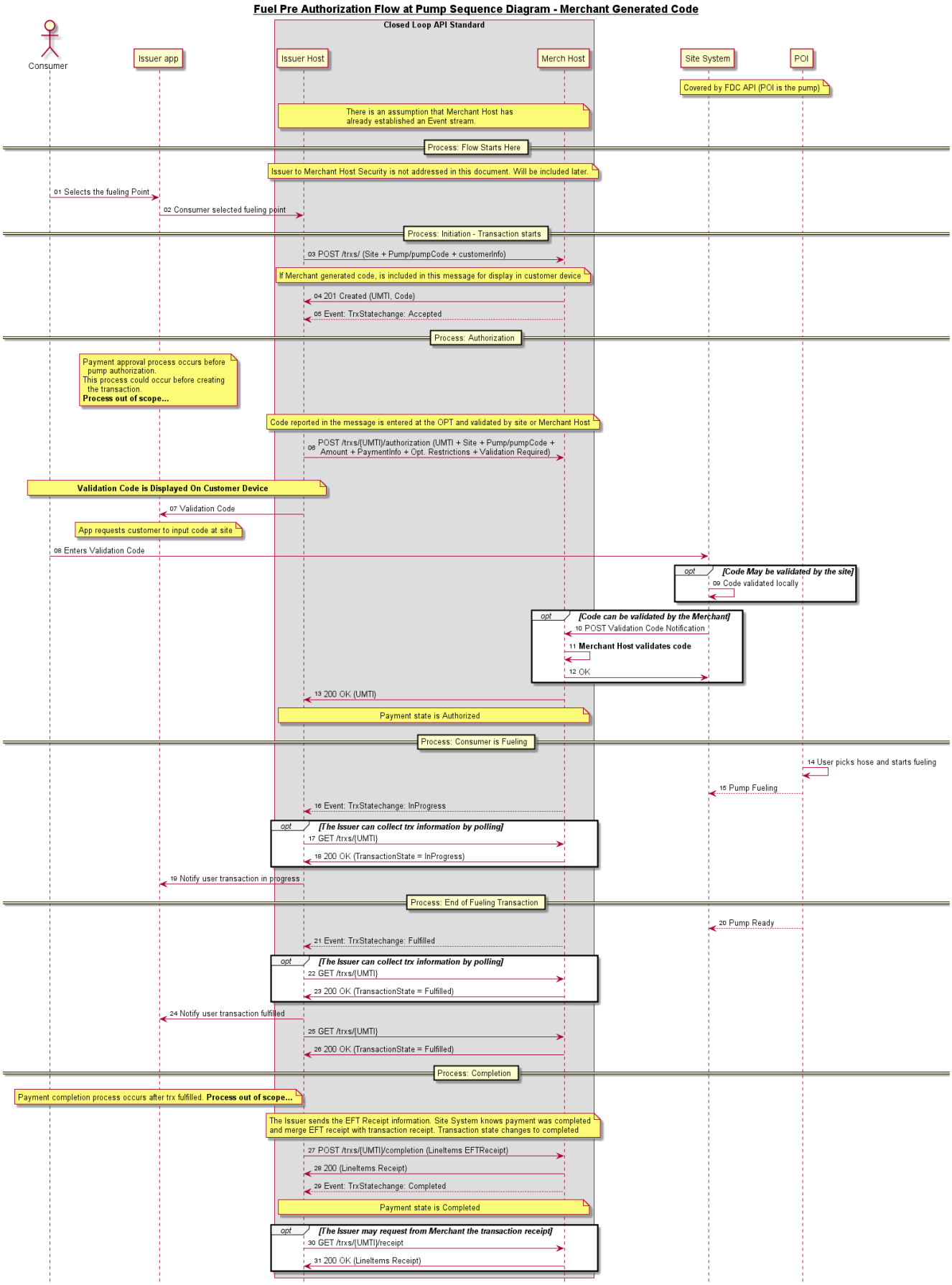
- The user cancels transaction before authorization when transaction state is “Accepted”, and payment state is “Payable”
- The user cancels transaction after Issuer authorization but before fueling starts. Transaction state is still “Accepted” and payment state is “Authorized”
- The user asks to cancel transaction after fueling starts. The Merchant responds with an error message saying that cancelation is not allowed because dispensing has already started. The transaction state remains in “InProgress”, and the payment state remains “Authorized” until completion



3.4.1.3 Pre-Authorization at Pump with code – Normal flow

This process is like the one below except for a code validation that has to take place during the authorization process:

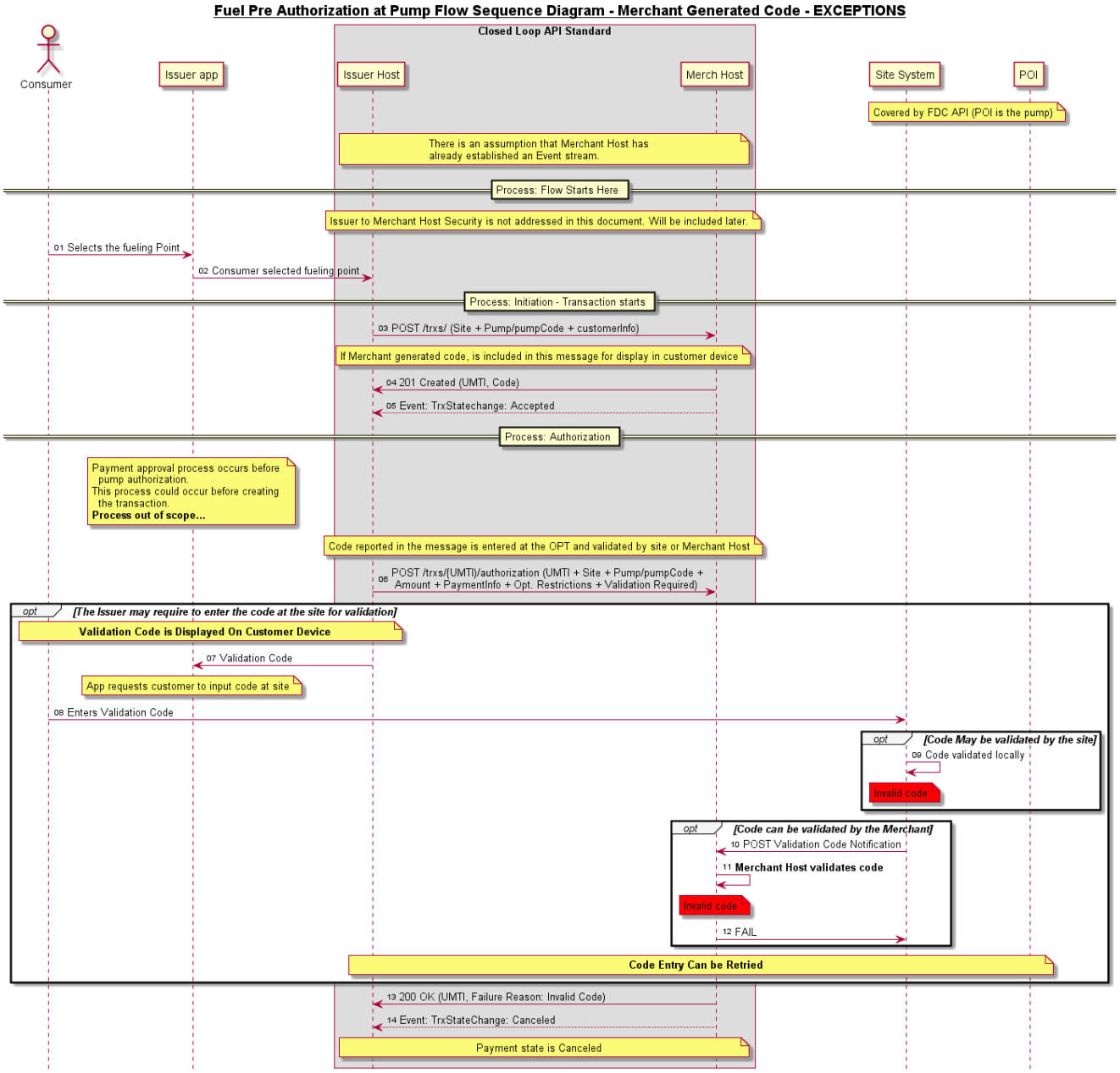
- The Merchant Host generates the validation code and sends it to the Issuer Host in the transaction initiation response with the action “display” at the customer device
- Code is displayed at the customer device
- Issuer Host asks the Merchant Host to “validate” the code during the authorization process
- Issuer application asks the customer to enter the code at the OPT
- Customer enters the code (retries are allowed)
- If code is OK, the Merchant Host responds with a “confirmed” in the authorization response



3.4.1.4 Pre-Authorization at Pump with code – Exception flow

The exception considered in this case is:

- The validation code entered at the OPT is wrong (retries are allowed) so the transaction is not authorized. Transaction state changes to “Canceled” and payment state changes to “Canceled”

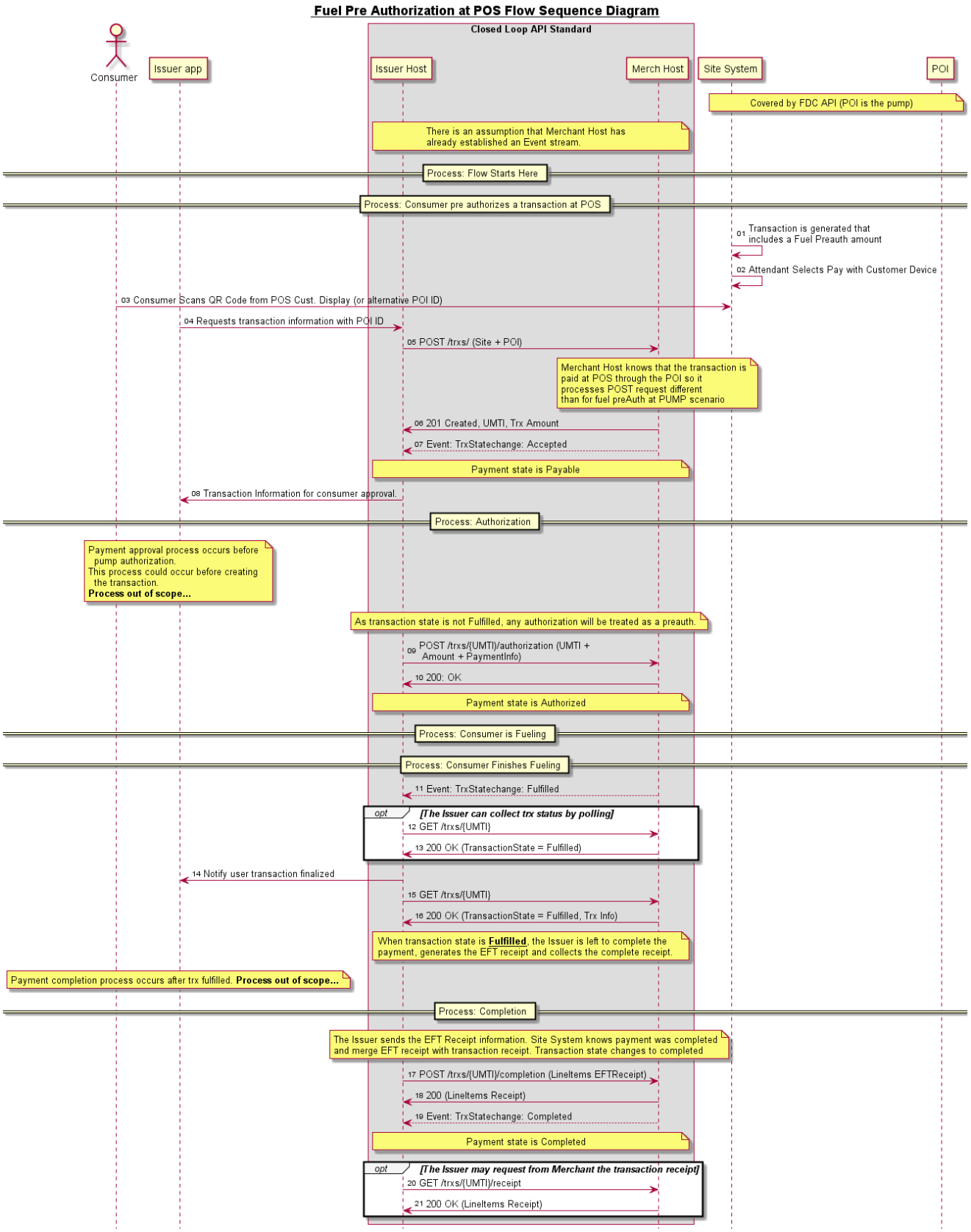


3.4.2 Pre-Authorization at POS

The steps below take place in the case of Pre-Authorization at POS

- POI / POS transaction identification can be done at the POS through a static or dynamic QR scan
- Process is initiated in a different way than Pre-Authorization at Pump because “Merchant Host” knows that the transaction is paid inside. Transaction is generated in “Accepted” state and payment state is “Payable” at this point
- Financial approval is requested
- Payment authorization process takes place and payment state changes to “Authorized”
- Consumer starts fueling and transaction state changes to “InProgress”
- Fueling is completed and transaction state changes to “Fulfilled”
- Transaction details are obtained from the “Merchant Host”
- Payment completion process takes place after transaction “Fulfilled”
- “Issuer Host” sends the EFT receipt for the “Site System” to merge with transaction receipt and “Issuer Host” receives the combined transaction receipt
- Both transaction state and payment state changes to “Completed” indicating the finalization of the process on both sides (issuer and merchant)

3.4.2.1 Pre-Authorization at POS– Normal flow

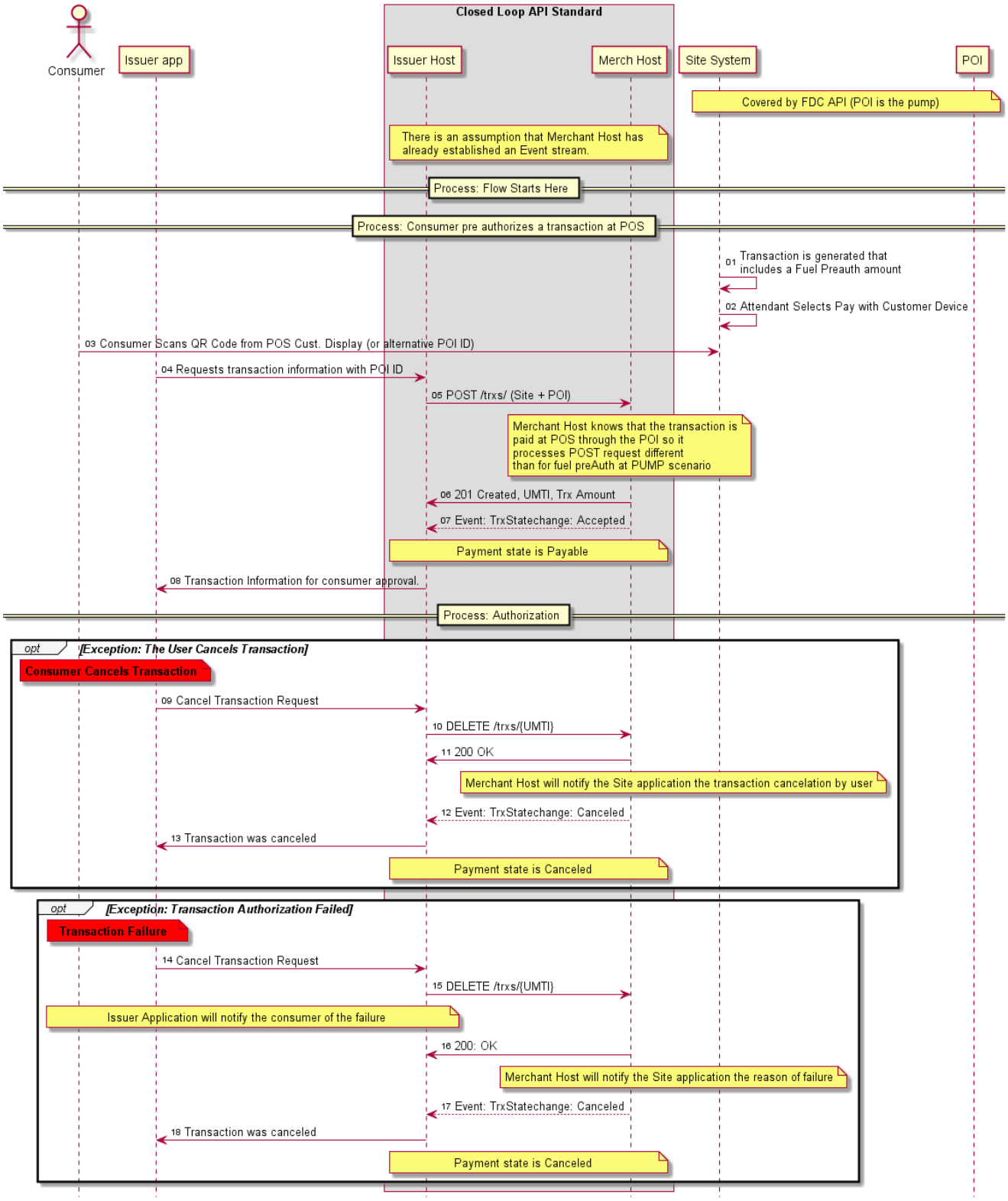


3.4.2.2 Pre-Authorization at POS– Exception flow

The exceptions considered in this case are:

- The user cancels transaction before authorization when transaction state is “Accepted”, and payment state is “Payable” (fueling has not started)
- The Issuer authorization fails so the Issuer Host post a delete transaction to the Merchant Host. Both transaction state and Payment states are canceled. Issuer application notifies the consumer about the failure and Merchant Host notifies the Site application

Fuel Pre Authorization at POS Flow Sequence Diagram - EXCEPTIONS

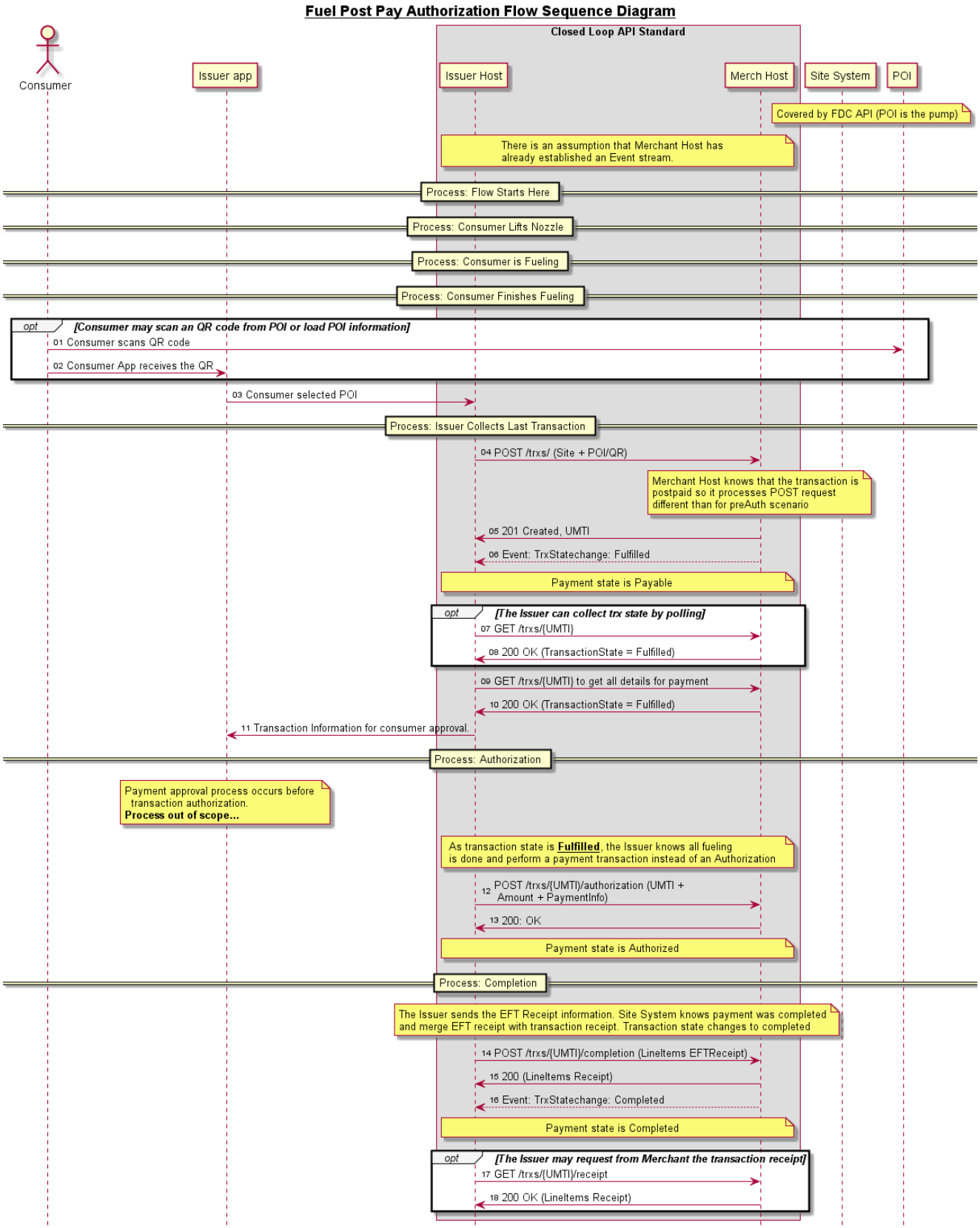


3.4.3 Post-Pay

The steps below take place in the case of Post-Pay

- Consumer may scan QR code from POI or load POI information
- Fueling takes place and after completion attendant may initiate transaction inside or outside
- Process is initiated in a different way than the pre-Auth scenario because “Merchant Host” knows that the transaction is postpaid. Transaction state is “Fulfilled”
- Transaction details are obtained from the “Merchant Host”
- “Issuer Host” knows transaction is done and perform a payment transaction instead of an authorization request
- “Issuer Host” sends the EFT receipt for the “Site System” to merge with transaction receipt and “Issuer Host” receives the combined transaction receipt
- Both transaction state and payment state changes to “Completed” indicating the finalization of the process on both sides (issuer and merchant)

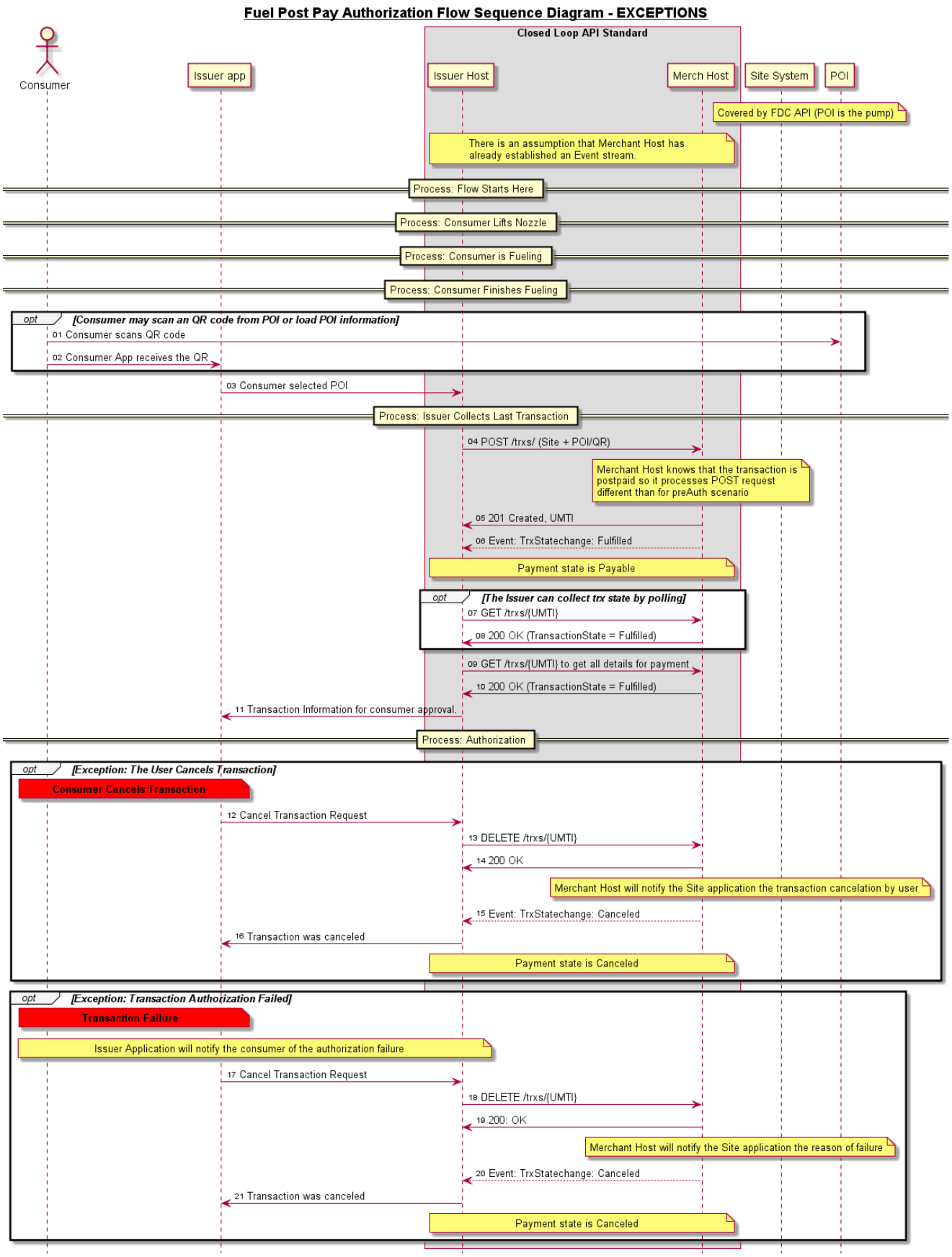
3.4.3.1 Post-Pay – Normal flow



3.4.3.2 Post-Pay – Exception flow

The exceptions considered in this case are:

- The user cancels transaction before authorization when transaction state is “Fulfilled”, and payment state is “Payable” (fueling has not started)
- The Issuer authorization fails so the Issuer Host post a delete transaction to the Merchant Host. Both transaction state and Payment states are canceled. Issuer application notifies the consumer about the failure and Merchant Host notifies the Site application



4 Security Considerations

Open Retailing provides an “Open Retailing API Implementation Guide: Security” document that addresses the security aspects of API transport technologies.

Payment technologies, including mobile payments, need to be properly assessed to ensure the solution provides the level of security needed to protect sensitive data. This implementation guide covers possible architectures, communication flows, message format and contents between the “Issuer Host” and “Merchant Host”; it does not address the security or compliance of specific implementations. It is recommended that solutions be developed in accordance with industry standards and security best practices (e.g., ISO 12812 – Part 2, NIST, PCI Standards) and that specific implementations are assessed to determine security and/or compliance considerations.

These APIs have been specifically designed so that no sensitive payment information needs to be shared with merchant. It is up to the issuer internal implementation how to protect this information.

5 Internationalization

The Host-to-Host API collection is mostly a system-to-system protocol. The "Open Retailing Design Rules for APIs OAS3.0" defines the format and use of dates, monetary amounts, and units of measurement when transmitting data. Internationalization is still applicable when sending receipts and prompts as text. However, for those cases, formatting dates, monetary amounts, and translation of textual data are implementation-specific and out of scope for this document

6 Implementation Details

The following messages are part of the Host-To-Host API collections:

- Common and Master Data
  - Connection (Heartbeat)
  - Sites Information
  - POIs (points of interaction)
  - Facilities
  - Site Facilities
  - Products

- Transaction process
  - Transaction Initiation
  - Authorize a sale transaction
  - Retrieve transaction details
  - Cancel a transaction
  - Complete transaction sending the EFT receipt and receiving combined transaction receipt from the Merchant
  - Send EFT receipt information to the Merchant (optional)
  - Retrieve receipt lines from the Merchant (optional)
  - Reconciliation between Issuer and Merchant
  - Server Sent Events

- Events data structure
  - Master changes
  - Transaction state change

It is not the intention of this manual to provide details of each message (just a brief description). The details can be found in the following documents:

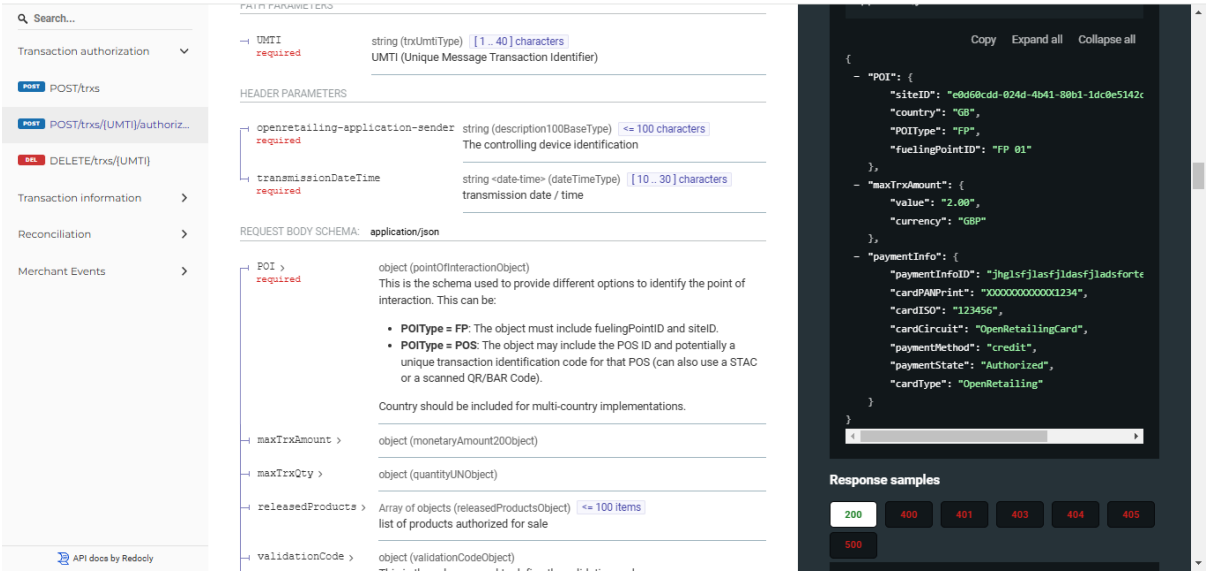
- **issuerInitiatedEventv1.1-redoc.html:** includes the Apis to get the event structure and examples

- **issuerInitiatedH2Hv1.1-redoc.html:** includes the APIs to manage the main processes

- **issuerInitiatedDCAv1.1-redoc.html:** includes the APIs to connect the "Merchant Host" and share information between the "Merchant Host" and the "Issuer Host

These documents were generated automatically from the OAS 3 API documentation files to ensure data integrity between the API and its documentation, using an open-source documentation tool called REDOC. The REDOC generated html information when presented in a PC can be visualized in the different sections of the screen:

- The list of APIs can be found on the left-hand side of the screen
- The request and response schemas can be found in the center
- The examples can be found on the right-hand side



Note: REDOC is the current tool used to document OAS 3 APIs files. Other tools may be used in the future, changing the layout of this document.

6.1 Common and Master Data

6.1.1 Connection

Relative URL	*/connection*
Method	POST
Input	application/json
Output	application/json

The “Issuer Host” must verify that the “Merchant Host” is available at regular intervals. The minimum recommended time lapse is forty-five seconds. Any shorter time will cause an unnecessary load on the network and undue burden on the “Issuer Host” resources. The H2H API protocol relies heavily on Server-Sent event streams. If the event stream is dropped (due to network failure, device spoofing, or other problem), there is no mechanism for the “Merchant Host” to re-establish connectivity. For that reason, the “Issuer Host” must call this heartbeat API at a regular interval so that it will know when to re-establish the event stream.

Note that if the response to the /connection request is a failure, the “Issuer Host” must re-establish the event stream after a success is received.

6.1.2 Sites

Relative URL	*/sites*
Method	GET
Input	application/json
Output	application/json

Allows to retrieve all sites information or filter by country. The update date is a required parameter for this request.

Provides sites information details (country, name, ID, GeoLocation, address lines, city, postal code, etc.). Allows to receive updated information only and/or filter by country.

6.1.3 POIs

Relative URL	*/POIs*
Method	GET
Input	application/json
Output	application/json

Allows to retrieve all POIs information or filter by country. The update date is a required parameter for this request.

Provides the list of available point of interaction identifiers for the customers at each site. This information includes list of pumps, QR codes, OPTs, POS Payment Terminals and every device with a fixed code identification (i.e., not identified through STAC).

6.1.4 Facilities

Relative URL	*/facilities*
Method	GET
Input	application/json
Output	application/json

Provides the list of available facilities at the network (001: Mogas, 002: Diesel, 003: Shop, 004: Coffee, etc.). Used to define the list of valid facilities codes.

6.1.5 Site Facilities

Relative URL	*/siteFacilities*
Method	GET

Input	application/json
Output	application/json

Allows to retrieve all sites facilities information or filter by country. The update date is a required parameter for this request.

Provides the list of available facilities at each site (Mogas, Diesel, Lubes, Shop, Coffee, Showers, etc.).

6.1.6 Products

Relative URL	*/products*
Method	GET
Input	application/json
Output	application/json

Provides the list of available product codes at the merchant network.

6.2 Transaction process

6.2.1 Transaction Initiation

Relative URL	*/trxs*
Method	POST
Input	application/json
Output	application/json

Depending on the use case the transaction initiation will trigger different actions from the merchant side. For example, in the case of Pre-Authorization at Pump, the “Merchant Host” will send a transaction initiation request to the “Site System” and upon confirmation from the “Site System” the “Merchant Host” responses with the UMTI number, transaction state and POI. Additionally, it sends a “trxStateChange” event to the “Issuer Host” with a “Accepted” transaction state.

Examples of Initiation Request and Response:

Request

```
{
  "POI": {
    "siteID": "e0d60cdd-024d-4b41-80b1-1dc0
e5142c26",
    "country": "GB",
    "POIType": "FP",
    "fuelingPointID": "FP 01"
  },
```

Response

```
{
  "statusReturn": {
    "timestamp": "2009-11-20T17:30:50",
    "result": "success",
    "error": "ERRCD_OK",
    "message": "Operation completed success
fully"
  },
```

<table>
<tr>
<td>

```
  "customerPreferences": {
    "receiptChoice": "yes"
  }
}
```

</td>
<td>

```
  "transaction": {
    "trxUmti": "968b12ea-caa5-1921-ecec-4cb
5503d6266",
    "trxState": "Accepted",
    "POI": {
      "siteID": "e0d60cdd-024d-4b41-80b1-1d
c0e5142c26",
      "country": "GB",
      "POIType": "FP",
      "fuelingPointID": "FP 01"
    }
  }
}
```

</td>
</tr>
</table>

6.2.2 Transaction Authorization

Relative URL */trxs/{UMTI}/authorization*

Method	POST
Input	application/json
Output	application/json

Depending on the use case the authorization will trigger different actions from the merchant side. For example, in the case of Pre-Authorization at Pump, the “Merchant Host” will send an authorize request to the “Site System” and upon confirmation of the authorization from the “Site System” the “Merchant Host” responses with the UMTI number, transaction state and POI. The payment state changes to “Authorized”.

Examples of Authorization Request and Response:

Request


```
{
  "POI": {
    "siteID": "e0d60cdd-024d-4b41-80b1-1dc0
e5142c26",
    "country": "GB",
    "POIType": "FP",
    "fuelingPointID": "FP 01"
  },
  "maxTrxAmount": {
    "value": "2.00",
    "currency": "GBP"
  },
  "paymentInfo": {
    "paymentInfoID": "jhglsfjlasfjldasfjlad
sforteowtowetljlsfjlsdfjls",
    "cardPANPrint": "XXXXXXXXXXXX1234",
    "cardISO": "123456",
    "cardCircuit": "OpenRetailingCard",
    "paymentMethod": "credit",
    “paymentState”: “Authorized”,
    "cardType": "OpenRetailing"
  }
}
```


Response


```
{
  "statusReturn": {
    "timestamp": "2009-11-20T17:30:50",
    "result": "success",
    "error": "ERRCD_OK",
    "message": "Operation completed success
fully"
  },
  "authorizeObject": {
    "trxUmti": "968b12ea-caa5-1921-ecec-4cb
5503d6266",
    "trxState": "Accepted",
    "POI": {
      "siteID": "e0d60cdd-024d-4b41-80b1-1d
c0e5142c26",
      "country": "GB",
      "POIType": "FP",
      "fuelingPointID": "FP 01"
    }
  }
}
```


6.2.3 Retrieve transaction details

Relative URL */trxs/{UMTI}*

Method	GET
Input	application/json
Output	application/json

The “Issuer Host” can retrieve transaction details from the “Merchant Host” at any time of the process. Depending on the transaction state some pieces of information will be available or not. The same type of transaction details lines is used for fuel and non-fuel products and if applies, the tax code and amount are within each transaction line (no total taxes line is provided).

Example of Transaction Details Response:

Request	Response

	{
	"statusReturn": {
	"timestamp": "2009-11-20T17:30:50",
	"result": "success",
	"error": "ERRCD_OK",
	"message": "Operation completed successfully"
	},
	"trxDetails": {
	"transactionInfo": {
	"trxUmti": "968b12ea-caa5-1921-ecec-4cb5503d6266",
	"trxDateTime": "2021-11-26T17:17:24.600Z",
	"POI": {
	"siteID": "e0d60cdd-024d-4b41-80b1-1dc0e5142c26",
	"country": "GB",
	"POIType": "FP",
	"fuelingPointID": "FP 01"
	},
	"trxState": "Fulfilled",
	"transactionLines": [
	{
	"trxLineSequenceNumber": 1,
	"productCode": 1,
	"unitPrice": {
	"value": "2.159"
	},
	"salesQuantity": {
	"value": "0.926"
	},
	"salesAmount": {
	"value": "2.00"
	},
	"refillingPointID": "FP 01"
	}
	]
	},
	"paymentInfo": {
	"paymentInfoID": "jhglsfjlasfjldasfjladsforteowtowetljlsfjlsdfjls",
	"cardPANPrint": "XXXXXXXXXXXX1234",
	"cardISO": "123456",
	"cardCircuit": "OpenRetailingCard",
	"paymentMethod": "credit",
	"paymentState": "Authorized",
	"preAuthAmount": {
	"value": "2.00"
	},
	"finalAmount": {
	"value": "2.00"
	},
	"cardType": "OpenRetailing"
	},
	"customerPreferences": {
	"receiptChoice": "yes"
	}
	}
	}

6.2.4 Cancel a transaction

Relative URL	*/trxs/{UMTI}/*
Method	DELETE
Input	application/json
Output	application/json

The cancelation of a transaction may respond to different reasons: the user cancels the transaction, there are no funds, the required product is not in the list of allowed products, etc. After a DELETE the transaction information can still be retrieved through a GET, although no further processing may be performed.

Depending on the state of the transaction, different results will occur:

- If transaction was not yet authorized, any pump reservation will be cleared and no ‘trxDetails’ will be returned.
- If transaction was authorized but dispensing has not yet started, the authorization will be cancelled and no ‘trxDetails’ will be returned.
- If fueling already started, the Merchant Host responds with an error message saying that transaction cannot be canceled after dispensing starts.
- If fueling was completed, the “Merchant Host” will clear the transaction at the site and return the actual dispensed volume in `trxDetails`.

Example of a Cancel Transaction Response:

Request	Response

	{
	"statusReturn": {
	"timestamp": "2009-11-20T17:30:50",
	"result": "success",
	"error": "ERRCD_OK",
	"message": "Operation completed successfully"
	},
	"trxDetails": {
	"transactionInfo": {
	"trxUmti": "968b12ea-caa5-1921-ecec-4cb5503d6266",
	"trxDateTime": "2021-11-26T17:17:24.600Z",
	"POI": {
	"siteID": "e0d60cdd-024d-4b41-80b1-1dc0e5142c26",
	"country": "GB",
	"POIType": "FP",
	"fuelingPointID": "FP 01"
	},
	"trxState": "Canceled"
	},
	"paymentInfo": {
	"paymentInfoID": "jhglsfjlasfjldasfjladsforteowtowetljlsfjlsdfjls",
	"cardPANPrint": "XXXXXXXXXXXX1234",
	"cardISO": "123456",
	"cardCircuit": "OpenRetailingCard",
	"paymentMethod": "credit",
	"paymentState": "Canceled",
	"preAuthAmount": {
	"value": "2.00"
	},
	"cardType": "OpenRetailing"
	}
	}
	}

6.2.5 Complete a transaction

Relative URL	*/trxs/{UMTI}/completion*
Method	POST
Input	application/json
Output	application/json

This API is used for the “Issuer Host” to send EFT receipt and retrieve transaction receipt from the “Merchant Host” which combines EFT and merchant information. Both transaction state and payment state changes to “Completed” indicating the finalization of the process on both sides (issuer and merchant).

Examples of a completion Request and Response:

Request


```
[
  {
    "alignment": "Left",
    "charStyle": "Bold",
    "text": "PayCard Mobile"
  },
  {
    "alignment": "Left",
    "charStyle": "Bold",
    "text": "PAN 38785768*****7645963"
  },
  {
    "alignment": "Left",
    "charStyle": "Bold",
    "text": "Auth # 675465"
  },
  {
    "alignment": "Left",
    "charStyle": "Normal",
    "text": "EFT # 546783"
  },
  {
    "alignment": "Left",
    "charStyle": "Normal",
    "text": "Payment Total 2.00"
  },
  {
    "alignment": "Center",
    "charStyle": "Bold",
    "text": "Customer Copy"
  },
  {
    "alignment": "Center",
    "charStyle": "Bold",
    "text": "Mobile verified"
  },
  {
    "alignment": "Center",
    "charStyle": "Bold",
    "text": "Please Retain for your records
"
```


Response


```
{
  "statusReturn": {
    "timestamp": "2009-11-20T17:30:50",
    "result": "success",
    "error": "ERRCD_OK",
    "message": "Operation completed success
fully"
  },
  "receipt": [
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "WELCOME"
    },
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "2141 NONAME ST"
    },
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "IFSF"
    },
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "VAT No GB9294758678"
    },
    {
      "alignment": "Left",
      "charStyle": "Bold",
      "text": "PayCard Mobile"
    },
    {
      "alignment": "Left",
      "charStyle": "Bold",
      "text": "PAN 38785768*****7645963"
    },
    {
      "alignment": "Left",
```

<table>
<tr>
<td>

```
  }
]
```

</td>
<td>

```
      "charStyle": "Bold",
      "text": "Auth # 675465"
    },
    {
      "alignment": "Left",
      "charStyle": "Normal",
      "text": "EFT # 546783"
    },
    {
      "text": "DATE 09/07/16 12:29"
    },
    {
      "text": "FP# 01"
    },
    {
      "text": "PRODUCT: PLUS"
    },
    {
      "text": "GALLONS: 0.926"
    },
    {
      "text": "PRICE/G: $ 2.159"
    },
    {
      "text": "FUEL SALE $ 2.00"
    },
    {
      "text": "THANK YOU"
    },
    {
      "text": "HAVE A NICE DAY"
    }
  ]
}
```

</td>
</tr>
</table>

6.2.6 Retrieve receipt lines from the Merchant

Relative URL	*/trxs/{UMTI}/receipt*
Method	GET
Input	application/json
Output	application/json

This API is used by the “Issuer Host” to retrieve receipt information from the “Merchant Host”.

Example of a receipt Response:

<table>
<tr><th>Request</th><th>Response</th></tr>
<tr>
<td></td>
<td>

```
{
  "statusReturn": {
    "timestamp": "2009-11-20T17:30:50",
    "result": "success",
    "error": "ERRCD_OK",
```

</td>
</tr>
</table>

```
    "message": "Operation completed success
fully"
  },
  "receipt": [
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "WELCOME"
    },
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "2141 NONAME ST"
    },
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "IFSF"
    },
    {
      "alignment": "Center",
      "charStyle": "Bold",
      "text": "VAT No GB9294758678"
    },
    {
      "alignment": "Left",
      "charStyle": "Bold",
      "text": "PayCard Mobile"
    },
    {
      "alignment": "Left",
      "charStyle": "Bold",
      "text": "PAN 38785768*****7645963"
    },
    {
      "alignment": "Left",
      "charStyle": "Bold",
      "text": "Auth # 675465"
    },
    {
      "alignment": "Left",
      "charStyle": "Normal",
      "text": "EFT # 546783"
    },
    {
      "text": "DATE 09/07/16  12:29"
    },
    {
      "text": "FP# 01"
    },
    {
      "text": "PRODUCT: PLUS"
    },
    {
      "text": "GALLONS: 0.926"
    },
    {
      "text": "PRICE/G: $ 2.159"
    },
    {
      "text": "FUEL SALE  $ 2.00"
    },
    {
      "text": "THANK YOU"
```

<table>
<tr><td></td><td><pre>
    },
    {
      "text": "HAVE A NICE DAY"
    }
  ]
}
</pre></td></tr>
</table>

6.2.7 Send EFT receipt information to the Merchant

Relative URL	*/trxs/{UMTI}/EFTReceipt*
Method	POST
Input	application/json
Output	application/json

This API is optional and is used for the “Issuer Host” to send EFT receipt information to the “Merchant Host”.

Examples of a EFTReceipt Request and Response:

Request


```
[
  {
    "alignment": "Left",
    "charStyle": "Bold",
    "text": "PayCard Mobile"
  },
  {
    "alignment": "Left",
    "charStyle": "Bold",
    "text": "PAN 38785768*****7645963"
  },
  {
    "alignment": "Left",
    "charStyle": "Bold",
    "text": "Auth # 675465"
  },
  {
    "alignment": "Left",
    "charStyle": "Normal",
    "text": "EFT # 546783"
  },
  {
    "alignment": "Left",
    "charStyle": "Normal",
    "text": "Payment Total 2.00"
  },
  {
    "alignment": "Center",
    "charStyle": "Bold",
    "text": "Customer Copy"
  },
  {
    "alignment": "Center",
    "charStyle": "Bold",
    "text": "Mobile verified"
```


Response


```
{
  "timestamp": "2009-11-20T17:30:50",
  "result": "success",
  "error": "ERRCD_OK",
  "message": "Operation completed successfu
lly"
}
```

```
  },
  {
    "alignment": "Center",
    "charStyle": "Bold",
    "text": "Please Retain for your records
"
  }
]
```


6.2.8 Reconciliation between Issuer and Merchant

Relative URL	*/reconciliations*
Method	POST
Input	application/json
Output	application/json

Both issuer and merchant totals will include transaction count and transaction total, grouped by country, currency, and issuer.

The request includes the issuer reconciliation date / time, and the issuer reconciliation totals.

The response includes the merchant from date / time, the merchant to date / time (same as issuer reconciliation date / time) and the merchant reconciliation totals.

If for the same issuer ID, two reconciliation requests are received with the same Date/Time, the response will be ‘Duplicated Request’. No new reconciliation totals will be calculated, and the same results should be returned as per the previous request.

Examples of a Reconciliation Request and Response:

Request


```
{
  "issuerDateTime": "2021-11-29T16:03:25.766Z",
  "issuerReconciliationTotals": [
    {
      "issuerID": "H2HIFSF",
      "country": "GB",
      "trxCount": 34,
      "trxTotals": {
        "value": "288.22",
        "currency": "GBP"
      }
    },
    {
      "issuerID": "H2HIFSF",
      "country": "CH",
      "trxCount": 24,
      "trxTotals": {
        "value": "67.22",
        "currency": "CHF"
      }
    },
    {
      "issuerID": "H2HIFSF",
      "country": "IT",
      "trxCount": 17,
      "trxTotals": {
        "value": "55.46",
        "currency": "EUR"
      }
    }
  ]
}
```


Response


```
{
  "statusReturn": {
    "timestamp": "2009-11-20T17:30:50",
    "result": "success",
    "error": "ERRCD_OK",
    "message": "Operation completed successfully"
  },
  "merchantReconciliation": {
    "merchantFromDateTime": "2021-11-28T15:02:55.766Z",
    "merchantToDateTime": "2021-11-29T16:03:25.766Z",
    "merchantReconciliationTotals": [
      {
        "issuerID": "H2HIFSF",
        "country": "GB",
        "trxCount": 34,
        "trxTotals": {
          "value": "288.22",
          "currency": "GBP"
        }
      },
      {
        "issuerID": "H2HIFSF",
        "country": "CH",
        "trxCount": 24,
        "trxTotals": {
          "value": "67.22",
          "currency": "CHF"
        }
      },
      {
        "issuerID": "H2HIFSF",
        "country": "IT",
        "trxCount": 17,
        "trxTotals": {
          "value": "55.46",
          "currency": "EUR"
        }
      }
    ]
  }
}
```

6.2.9 Server Sent Events

Relative URL	*/merchantEvents*
Method	GET
Input	application/json
Output	application/json

For the H2H API standard to work correctly, the “Merchant Host” must send "unsolicited" requests to the “Issuer Host”. The OpenRetailing API rules and guidelines support the use of Server-Sent events for that purpose. The latest HTML5 specification defines Server-Sent Events (SSE). SSE works like a regular HTTP request where the server converts the response into an event stream by setting the response's "Content-Type" header to "text/event-stream." The message contains an “event:” field followed by a “data:” description. Two consecutive line feeds separate the events; for that reason, it is crucial to keep the events as short as possible.

When the "/merchantEvents" API is called, the “Merchant Host” will respond with an URL, that URL will be used to register for unsolicited server sent events. Through this connection the merchant sends events and the issuer listens and responds with the corresponding action.

The table below describes the event names and their purpose:

Event	Description
`masterChanges`	Changes in Master Data **Action:** The “Issuer Host” will reply by calling the /sites, /POIs, /facilities, /siteFacilities and /products APIs to get the updates
`trxStateChange`	Any transaction state event from the “Merchant Host” is responded by the “Issuer Host” with the corresponding action.

6.3 Error Handling

There will be several errors that will be returned in each API depending on the error reason. If the message is received by the server in good order, the server should

return a 2xx series message. Except for the load balancer 5xx errors, error codes and returned payload schemas, are included in the API OAS documentation.

6.3.1 Successful 2xx

The APIs will return 200 codes upon success and 201 when a new record is created.

Return Code	*200*
Description	OK
Reason	Normal successful return

Return Code	*201*
Description	Created
Reason	Resource created

6.3.2 Errors 4xx – Client Errors

400 errors are returned when no access is possible to the required resource, and the return code will depend on the reason, as depicted below.

Return Code	*400*
Description	Bad request
Reason	Requests with invalid payloads must use the response 400. Problem with either the representation or metadata.

Return Code	*401*
Description	Unauthorized
Reason	Requests with invalid credential must use the response 401.

Return Code	*403*
Description	Forbidden
Reason	Requests with valid credentials but missing valid scope must use de response 403.

Return Code	*404*
Description	Not found
Reason	Requests with invalid URLs (paths) must use the response 404.

Return Code	*405*
Description	Method not allowed
Reason	Requests with Http method not supported use the response 405.

6.3.3 Errors 500 – Internal Server Errors

500 error will be returned whenever there is an internal server processing error.

Return Code	*500*
Description	Internal server Error
Reason	Requests that cause an internal server error (outside of the service itself), such as out of memory or corrupted file, etc.

6.3.4 Errors 5xx – Load Balancer Errors

On several occasions, the API server will be operating behind a load balancer. The Load balancer may reply with specific errors in case that the server is not responding properly. These errors and their expected response schemas are NOT documented in the API documentation, as they are generated by the load balancer applications and will depend on the specific load balancer used, for example NGINEX, AWS or Azure. Please refer to the specific load balancer documentation for more information. The API client must support at least the following load balancer related errors:

Return Code	*502*
Description	Bad Gateway
Reason	The 502 Bad Gateway error is an HTTP status code that occurs when a server acting as a gateway or proxy receives an invalid or faulty response from another server in the communication chain.

Return Code	*503*
Description	Service Unavailable
Reason	The 503 Service Unavailable error is an HTTP status code that is returned when the services behind the load balancer are unhealthy. Please retry later.

Return Code	*504*
Description	Gateway Timeout
Reason	The 504 Gateway Timeout error is an HTTP status code that indicates that the load balancer closed a connection because a request did not complete within the idle timeout period. Possibly indicating very high load on the server.

6.4 Events data structure

6.4.1 Master changes

Example of a Master Change Event:


```
{
  "eventID": "340",
  "event": "masterChanges",
  "timestamp": "2021-11-29T17:11:37.167Z",
  "changes": {
    "country": "GB",
    "merchantID": "e0d60cdd-024d-4b41-80b1-1dc0e5142c26",
    "changeType": "New",
    "changedObject": "sites"
}
```


6.4.2 Transaction State Change

Example of a Transaction State Change Event:


```
{
  "eventID": "187",
  "event": "trxStateChange",
  "timestamp": "2021-11-29T17:11:37.167Z",
  "trxUmti": "968b12ea-caa5-1921-ecec-4cb5503d6266",
  "trxState": "InProgress"
}
```

A. References

A.1 Normative References

- Part 3-50 IFSF Host to Host V2 interface specification v2.16Mobile Payments – Implementation Guide - June 8, 2021 – API version 1.0
- Open Retailing API Design Rules for JSON
- Open Retailing API Implementation Guide – Security
- Open Retailing API Implementation Guide - Transport Alternatives
- Open Retailing Design Rules for APIs OAS3.0
- RESTFul Web Services - (https://en.wikipedia.org/wiki/Representational_state_transfer)
- Open API Specification Version 3.0.1 - (https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md)

A.2 Non-Normative References

None

B. Glossary

Term	Definition
Dispenser	Dispenser or Pump - The fuelling device that delivers product to a consumer (also known as a pump). This device may or may not include an OPT.
EPS	Electronic Payments Server – a hardware and software application integrated with the site system that processes payments (mobile or conventional) with an off-site payments application.
FC	Forecourt Controller - a device controlling the operation of the Dispensers and passing data to and from them. Note: this functionality may be part of the function of an FDC
FDC	Forecourt Device Controller - a central controlling device installed at the site which enables communication of data and control to all forecourt devices (e.g., Dispensers, price signs, etc.). In some applications the FDC and EPS are in the same device.
MD	Mobile Device - the mobile device (e.g., smart phone, tablet) used by the consumer to interface with the mobile payments application (MPA)
MPA	Mobile Payments Application - a software application downloaded by a consumer to a MD which enables mobile payments for "in-store" and forecourt transactions.
MPP	Mobile Payments Processor - a supplier of the application that provides communication between the MPA, the site and the PFEP. The supplier will provide an application (the MPPA) that enables the transactions to be processed and transactions to be enabled and settled. This is Mobile Financial Service Provider (MFSP) in the ISO 12812.
MPPA	Mobile Payments Processing Application - the application provided by the MPP that enables communication with the MPA, the site system and the PFEP to instruct the site to release dispensers, process transactions and obtains necessary authorisations and other data from the PFEP.
OPT	Outdoor Payment Terminal - a device installed at a retail petroleum site to enable payment outdoors without direct intervention from a site operator. For the purposes of this document, this may be a single device mounted in a central position that controls multiple dispensers or a device integrated into each dispenser. Note: a similar device may also be used to control an ACW
IPT	Indoor Payment Terminal – a device installed at the POS lane with consumer input capabilities (e.g., PIN entry)
POS	Point of Sale - the device (hardware and software) that is used to process transactions on the site.
PFEP	Payment Front End Processor- (sometimes referred to as the Front-End Processor or FEP) - the application or institution that

Term	Definition
	the Site uses for the processing of payments. This may be a third party provided application made available as a service or an in-house application provided by the MPP or a major fuel brand.
Site	Site - the retail fuel facility.
Site System	*Site System – site equipment and components (hardware and software) including, but not limited to, POS, EPS, FD, and FDC.*
POI	Point of Interaction – Unique identification of a point of sale
STAC	Single Transaction Authentication Code
UMTI	Unique Message Transaction Identifier – Single use unique transaction identifier assigned by the "Merchant Host".
OAS	The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.
Merchant Host	The Merchant is any party known to the Issuer and who has a contractual agreement with the Issuer to provide goods to the Issuer's customers. The Merchant may operate a single site or a network of sites. The Merchant host is a trusted host, known to the Issuer and which the Issuer trusts to manage the transaction at site and provide necessary details. Note: For financial cards, the agreement between Issuer and Merchant may be purely financial and not extend to the supply of specific goods.
Issuer	The Issuer can be any party known to the Merchant and who has a contractual agreement with the merchant to guarantee payment for any Issuer approved transaction (purchase). Depending on the specific implementation, the "issuer" may be a payment processor acting on behalf of the issuer or even an acquirer/acquirer payment processing providing payment guarantees for multiple issuers.
Issuer Host	The Issuer Host is a trusted host, known to the Merchant and which the Merchant trusts to provide Issuer (or Payment Processor) approval.