

FISPAY API

Payment terminal API specification

Specification version 0.1.0

commit c963fc91db9159a4cb93d3d06d6dac1513dff209

Placeholder content: the message catalog is not final.

Contents

1	FISPAY API	2
1.1	Where to go	2
2	Getting started	3
2.1	What the API is	3
2.2	One endpoint	3
2.3	The message envelope	3
3	Guides	4
3.1	Send a Diagnosis request	4
4	Reference	6
4.1	Message header	6
4.2	Diagnosis	6
4.3	Test vectors	7
5	Explanation	8
5.1	Versioning: additive only	8
5.2	Message signing	8

1 FISPAY API

Placeholder: This site shows the documentation pipeline. The message catalog is a placeholder until the real one lands.

The FISPAY API is how a sale system, such as a cash register, talks to a FISPAY payment terminal. Both sides exchange JSON messages; each message names its operation in a common header.

The specification lives in Git. This website, the PDF and the tests are all built from the same commit, so they always agree. The footer shows the specification version and that commit.

1.1 Where to go

- Getting started: the endpoint and the message envelope.
- Guides: step-by-step tasks.
- Reference: field tables, test vectors and the HTTP API.
- Explanation: how the API evolves and how messages will be signed.
- Downloads: PDF, OpenAPI document and JSON Schemas.

2 Getting started

Placeholder: The endpoint and message shape below are stable; the message catalog is a placeholder.

2.1 What the API is

The sale system sends a request message to the terminal and gets the matching response message back. The terminal does the payment work; the sale system only asks and reads the answer.

2.2 One endpoint

There is a single HTTP endpoint: POST / with a JSON body. The body itself says which operation it is, so new operations never need new URLs. The HTTP API page shows the contract.

2.3 The message envelope

Every request is wrapped in `SaleToPOIRequest`, every response in `SaleToPOIResponse`. Inside the wrapper there are two parts:

- `MessageHeader`: who is talking and which operation this is. See `Message header`.
- The body for that operation, for example `DiagnosisRequest`. See `Diagnosis`.

```
{
  "SaleToPOIRequest": {
    "MessageHeader": { "...": "..."},
    "DiagnosisRequest": {}
  }
}
```

Receivers ignore fields they do not know, so a newer sender keeps working with an older receiver.

3 Guides

Placeholder: This walk-through uses the placeholder Diagnosis messages. Real guides follow with the message catalog.

3.1 Send a Diagnosis request

A Diagnosis asks the terminal to check itself and, optionally, its connection to the payment host.

1. Build the request

Fill in the header with your sale system ID, the terminal ID and a new service ID. Leave the body empty for a terminal-only check:

```
{
  "SaleToPOIRequest": {
    "MessageHeader": {
      "ProtocolVersion": "3.1",
      "MessageClass": "Service",
      "MessageCategory": "Diagnosis",
      "MessageType": "Request",
      "ServiceID": "2041",
      "SaleID": "ECR-01",
      "POIID": "FISPAY-T-0001"
    },
    "DiagnosisRequest": {}
  }
}
```

To include the payment host, set `HostDiagnosisFlag` to `true`:

```
{
  "SaleToPOIRequest": {
    "MessageHeader": {
      "ProtocolVersion": "3.1",
      "MessageClass": "Service",
      "MessageCategory": "Diagnosis",
      "MessageType": "Request",
      "ServiceID": "2041",
      "SaleID": "ECR-01",
      "POIID": "FISPAY-T-0001"
    },
    "DiagnosisRequest": {
      "HostDiagnosisFlag": true
    }
  }
}
```

2. Send it

Post the JSON to the terminal endpoint (POST /). See the HTTP API.

3. Read the response

The response repeats the header with `MessageType` set to `Response`. The `Result` field tells you how the check went:

```
{
  "SaleToPOIResponse": {
```

```

    "MessageHeader": {
      "ProtocolVersion": "3.1",
      "MessageClass": "Service",
      "MessageCategory": "Diagnosis",
      "MessageType": "Response",
      "ServiceID": "2041",
      "SaleID": "ECR-01",
      "POIID": "FISPAY-T-0001"
    },
    "DiagnosisResponse": {
      "Response": {
        "Result": "Success"
      }
    }
  }
}

```

On a fault, Result is Failure:

```

{
  "SaleToPOIResponse": {
    "MessageHeader": {
      "ProtocolVersion": "3.1",
      "MessageClass": "Service",
      "MessageCategory": "Diagnosis",
      "MessageType": "Response",
      "ServiceID": "2041",
      "SaleID": "ECR-01",
      "POIID": "FISPAY-T-0001"
    },
    "DiagnosisResponse": {
      "Response": {
        "Result": "Failure"
      }
    }
  }
}

```

All fields are described in Diagnosis.

4 Reference

Placeholder: Tables are generated from the placeholder schemas on every build.

Every table on these pages is generated from the JSON Schema files at build time, so it always matches the schemas.

- Message header: the header shared by all messages.
- Diagnosis: the Diagnosis request and response.
- Test vectors: AES-CMAC vectors to check an implementation.
- HTTP API: the OpenAPI contract, rendered with Redoc.

Multiplicity reads as min..max: 1..1 is required, 0..1 is optional, * means many.

4.1 Message header

Placeholder: Generated from the placeholder schema `schemas/common/message-header.json`.

Every message starts with this header. It names the operation and identifies the sale system and the terminal.

Field	Type	Multiplicity	Description	Example
ProtocolVersion	string	1..1	Revision of the message protocol the sender speaks, as major.minor.	3.1
MessageClass	string	1..1	Broad family of the exchange, for example a service started by the sale system.	Service
MessageCategory	string	1..1	Which operation the message belongs to, for example a diagnosis.	Diagnosis
MessageType	string	1..1	Whether this message asks for something or answers an earlier message.	Request
ServiceID	string	1..1	Identifier chosen by the sale system for this request; the terminal uses it to detect retries (idempotency key). A response repeats the value of its request.	2041
SaleID	string	1..1	Identifier of the sale system (cash register) that takes part in the exchange.	ECR-01
POIID	string	1..1	Identifier of the payment terminal that takes part in the exchange.	FISPAY-T-0001

4.2 Diagnosis

Placeholder: Generated from the placeholder schemas `schemas/diagnosis-request.json` and `schemas/diagnosis-response.json`.

A Diagnosis asks the terminal to check its own state. The header fields fixed below tell the terminal which message it received.

Request

Field	Type	Multiplicity	Description	Example
SaleToPOIRequest	object	1..1	Wrapper for every message the sale system sends to the terminal.	
SaleToPOIRequest.MessageHeader	object (MessageHeader)	1..1	Common envelope, narrowed to a diagnosis request.	
SaleToPOIRequest.MessageHeader.MessageClass	string, fixed: Service	1..1	Fixed to Service for this message.	Service

Field	Type	Multiplicity	Description	Example
SaleToPOIRequest.MessageHeader.MessageCategory	string, fixed: Diagnosis	1..1	Fixed to Diagnosis for this message.	Diagnosis
SaleToPOIRequest.MessageHeader.MessageType	string, fixed: Request	1..1	Fixed to Request for this message.	Request
SaleToPOIRequest.DiagnosisRequest	object	1..1	Body of the diagnosis request.	
SaleToPOIRequest.DiagnosisRequest.HostDiagnosisFlag	boolean	0..1	When true, the terminal also checks that it can reach the payment host. When absent or false, only the terminal itself is checked.	true

Response

Field	Type	Multiplicity	Description	Example
SaleToPOIResponse	object	1..1	Wrapper for every message the terminal sends back to the sale system.	
SaleToPOIResponse.MessageHeader	object (MessageHeader)	1..1	Common envelope, narrowed to a diagnosis response.	
SaleToPOIResponse.MessageHeader.MessageClass	string, fixed: Service	1..1	Fixed to Service for this message.	Service
SaleToPOIResponse.MessageHeader.MessageCategory	string, fixed: Diagnosis	1..1	Fixed to Diagnosis for this message.	Diagnosis
SaleToPOIResponse.MessageHeader.MessageType	string, fixed: Response	1..1	Fixed to Response for this message.	Response
SaleToPOIResponse.DiagnosisResponse	object	1..1	Body of the diagnosis response.	
SaleToPOIResponse.DiagnosisResponse.Response	object	1..1	Outcome of the diagnosis.	
SaleToPOIResponse.DiagnosisResponse.Response.Result	string, one of: Success, Failure, Partial	1..1	Overall verdict: Success when every check passed, Failure when the diagnosis could not be done or found a fault, Partial when only some checks passed.	Success

4.3 Test vectors

These are the public AES-128-CMAC vectors from RFC 4493. Use them to check your CMAC implementation before message signing arrives. Generated from `test-vectors/cmac/`.

ID	Key	Message (hex)	Tag
rfc4493-example-1	2b7e151628aed2a6 abf7158809cf4f3c	(empty)	bb1d6929e9593728 7fa37d129b756746
rfc4493-example-2	2b7e151628aed2a6 abf7158809cf4f3c	6bc1bee22e409f96e93d7e117393172a	070a16b46b4d4144 f79bdd9dd04a287c
rfc4493-example-3	2b7e151628aed2a6 abf7158809cf4f3c	6bc1bee22e409f96e93d7e117393172a ae2d8a571e03ac9c9eb76fac45af8e51 30c81c46a35ce411	dfa66747de9ae630 30ca32611497c827
rfc4493-example-4	2b7e151628aed2a6 abf7158809cf4f3c	6bc1bee22e409f96e93d7e117393172a ae2d8a571e03ac9c9eb76fac45af8e51 30c81c46a35ce411e5fbc1191a0a52ef f69f2445df4f9b17ad2b417be66c3710	51f0bebf7e3b9d92 fc49741779363cfe

5 Explanation

Placeholder: The rules on this page apply now; the signing details come in a later unit.

5.1 Versioning: additive only

The specification version follows semantic versioning. Within one major version, changes are **additive only**:

- New optional fields and new messages may appear.
- An existing field is never removed, renamed or given a new meaning.
- An existing constraint is never tightened.

Receivers **must ignore fields they do not know**. That is what keeps a newer sender compatible with an older receiver, and the reason the schemas do not reject extra properties. A change that breaks these rules needs a new major version.

5.2 Message signing

Messages will be protected with the nexa `SecurityTrailer`, using AES-CMAC as the message authentication code. The exact recipe, meaning which bytes are signed and how keys are agreed, is defined in a later unit. Until then, the test vectors let you verify an AES-CMAC implementation.