The only guide you need to integrate your point of sale (POS) with FUNDiMO. It goes from a sandbox key to registers that take payments for real merchants. It is for a POS vendor's developer.

The stages

Stage	What it is	What you need
1. Integrate	Section 1, in the sandbox. Build your POS until a conformance run is green	A sandbox key: one call, with no sign-up Help: an email to `support@fundimo.com.au`
2. Enrol	Section 2, on staging, then on production. Your server gets its certificate, and you list your registers	A first pairing code from FUNDiMO. You cannot enrol alone Help: your Slack Connect channel
3. Link	Section 3, at the counter. Link each register to its terminal	The cashier, with the terminal in their hand. A terminal fleet at a store that accepts your POS vendor Help: your Slack Connect channel

The environments

	Sandbox	Staging	Production
Address	`https://mw.sandbox.fundimo.com.au`	`https://mw.staging.fundimo.com.au`	`https://mw.fundimo.com.au`
For	Integrate. Simulated terminals. No card, and no money	Enrol and link, to test. Real terminals, on test cards only. Never use a real card	Enrol and link, for real merchants
You need	A sandbox key: one call, with no sign-up	A terminal fleet: real terminals at a store that accepts your POS vendor. A certificate that staging issued	The terminal fleet at your merchants' stores. A certificate that production issued

Contents

The system **3**

1 Integration **5**
1.1 Take a sandbox key 5
1.2 Send a first test sale 6
1.3 Read the sale until it settles 7
1.4 Try every test terminal 8
1.5 Recover from a lost response 9
1.6 Send a refund 9
1.7 Cancel a sale in progress, if your register has a cancel button 10
1.8 Enrol your server for its certificate 12
1.9 Send sales on your certificate 13
1.10 Run the conformance scenarios, and keep the report 14

2 Enrolment **16**
2.1 Get your first server listed 17
2.2 Enrol your server 17
2.3 List each register 18
2.4 Automate the renewal of every certificate 18

3 Link **20**
3.1 Link each register to its terminal, at the counter 20
3.2 Send a first sale through the linked terminal 21

What green conformance means for your POS software **22**

The sandbox's MCP server **23**
What it offers 23
An example 24

Get help **26**

The pages **27**

Addendum: the ISO 20022 terms in this guide **28**
Fields 28
Codes 29
FUNDiMO's own names 29

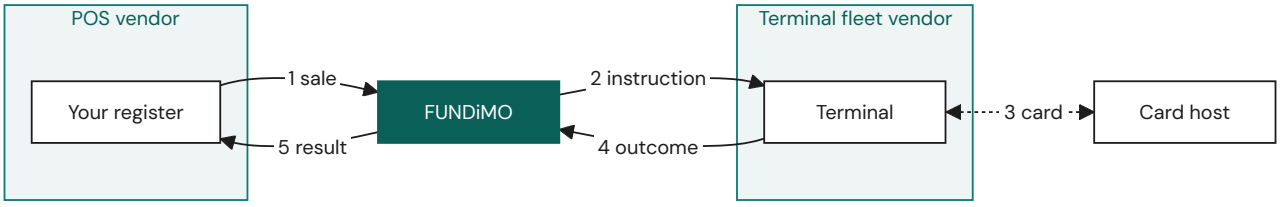
The system

FUNDiMO is the relay between a point of sale (POS) and a payment terminal. Three players take part in each sale:

Player	What it runs	What it holds
The POS vendor: you	The POS software at each register. It runs on your own server, or as an app on a device. Each register sends its sales to FUNDiMO, and reads their results.	The bill, its items and the store's own surcharge. Your own reference for each sale. The private key of each of your servers and devices.
The terminal fleet vendor	The payment terminals at each store, and the app on each terminal. A terminal takes the card with its own card host.	The card and the payment, between the terminal and its host. It lists its terminals at each store. It gives the merchant the store's `MerchantIdentification`.
FUNDiMO	The relay between the two. Your register calls FUNDiMO over HTTPS. Each terminal holds a WebSocket connection open to FUNDiMO.	Each sale, its status and the references that the terminal sent. Which register is linked to which terminal. A log of every message.

A register is the POS software at one counter of a store, where a cashier takes payments. It is an app on a device, or a name that your server gives. This guide calls it a register everywhere. The register's id is `RegisterIdentification`, and its header is `Mw-Register`.

How a sale moves. Your register sends a sale to FUNDiMO. FUNDiMO sends the sale to the terminal that your register is linked to. The terminal takes the payment with its own card host. The result then goes to FUNDiMO, and your register reads it. Card, bank and personal data are forbidden in every field. What a terminal sends passes an allow-list.



Your register calls FUNDiMO over HTTPS (1, 5). The terminal holds its own WebSocket connection to FUNDiMO (2, 4). The card goes between the terminal and its card host (3, dotted).

A certificate for each edge. On staging and production, each part that connects to FUNDiMO identifies itself with a client certificate. FUNDiMO issues each certificate. Each POS server, each register on a device and each terminal has its own certificate. A certificate is valid for 30 days. FUNDiMO issues a new one from day 15, for a new private key. After 30 days, a certificate cannot renew. Its holder then needs a new pairing code from a person.

What FUNDiMO does with your server's certificate, and why it is there. It is your server's credential on staging and production, in place of a sandbox key.

- FUNDiMO's own CA issues it at `POST /v1/enrol`, to the server that spends the pairing code. Your private key stays in your custody. FUNDiMO never sees it.
- The certificate names the kind of environment, `live` or `sandbox`, the role and your server's id. Staging and production are both `live`. Your server presents it at the TLS handshake of every call. FUNDiMO identifies the caller from the certificate only.

- Its scope sets what it reaches, and nothing more. A `pos` server's certificate reaches sales, their reads and the link calls, for the register that `Mw-Register` names. An `admin` server's certificate reaches the Operator API of your own POS vendor.
- A register that your server lists has no certificate: your `pos` server speaks for it. A register on a device has its own certificate, and names only itself.
- After a renewal, FUNDiMO retires the old certificate one hour after the first use of the new one, by default. It then refuses the old one with `403 certificate_retired`.

What FUNDiMO trusts. FUNDiMO loads nothing of yours into its trust store. A server has no attestation. The pairing code is the only tie between your first certificate and your POS vendor: the server that spends it first holds the identity. Thus keep each pairing code secret.

What your software trusts. FUNDiMO's server certificate is a public certificate. Your software checks it with its default trust store, as any HTTPS client does. Never disable that check.

Thus our guidance is: automate the renewal in each live instance of your POS software that holds a certificate. Use one renewer for each certificate. A register that runs its own instance, as an app on a device, renews its own certificate. If several registers share one server, one instance renews for all of them (step 2.4).

SECTION 1

Integration

The sandbox is FUNDiMO's environment for integration. It is self-help: one call gives you a sandbox key and a test world, and you do not sign up. Nothing in the sandbox is a record. No card is charged, and no money moves. Your test world is valid for 30 days. This section ends when a conformance run is green.

What you need. A shell with `curl` and `jq`. OpenSSL 3 or later, for step 1.8.

The address is a setting. Keep the address in one setting from the first line of code. Make every call from that setting. Never write an address into code. In this section, the commands read the sandbox's address from `$SANDBOX`:

```
SANDBOX=https://mw.sandbox.fundimo.com.au
```

The ids in every call. FUNDiMO's pages use ISO 20022's names. Your code uses them too. The addendum gives the definition of each term (Addendum).

Field	What it names
`MerchantIdentification`	The store. Every sale names it.
`POIIdentification`	The terminal that the sale goes to.
`RegisterIdentification`	The register's id.
`SaleReferenceIdentification`	Your own reference, for example an invoice number. FUNDiMO sends it to the terminal.
`IdempotencyKey`	Your register's idempotency key for one payment attempt. A retry with the same idempotency key never makes a second sale.
`CashierIdentification`	Optional. Your id for the user of your POS who made the sale or the refund, for example a staff number. Never the person's name, email address or phone number. FUNDiMO sends it to the terminal.
`TransactionReference`	FUNDiMO's id for the sale: `sal_` and 32 hex characters. You read the sale with it.

1.1 Take a sandbox key

Send one call. It needs no sign-up and no credential:

```
curl -s -X POST "$SANDBOX/sandbox/v1/keys"
```

Response: `201`, and a test world of your own:

- `SandboxKey`: your sandbox key, `sbx_` and 43 more characters. FUNDiMO shows it one time only. Send it as `Authorization: Bearer sbx_...`. Keep it out of logs and repositories.
- `TesterIdentification`, for example `sbx-3f9a1c2e7b10`. It is not a secret. The names of your store and terminals start with it.
- `MerchantIdentification`, your one store.
- `RegisterIdentification`, the id of your one register.
- `Terminals`: nine test terminals (step 1.4).
- `Enrolment`: a pairing code for your server. It is valid for 15 minutes (step 1.8).
- `ExpiryDateTime`: 30 days later. If you lose your sandbox key, get a new one. It comes with a new world.

Refusals

Status	Code	What to do
429	sandbox_key_rate	The sandbox issued all the sandbox keys that it can this hour. Try again in one hour.
503	sandbox_full	Sandbox keys become free when they expire. Try again in one hour.
503	store_unavailable	FUNDiMO made no sandbox key. Retry with the normal backoff: wait 1 second, and double the wait after each failure. The maximum wait is 30 seconds. Add a small random time to each wait.

1.2 Send a first test sale

Send a sale to `-approve`. This test terminal always approves. Put your sandbox key in place of `sbx_...`. Put your `TesterIdentification` in place of `sbx-3f9a1c2e7b10`. Make a new `IdempotencyKey` for each sale:

```
curl -s -X POST "$SANDBOX/v1/sales" \
  -H 'Authorization: Bearer sbx_...' \
  -H 'Content-Type: application/json' \
  -d '{"MerchantIdentification": "sbx-3f9a1c2e7b10-M1",
       "POIIdentification": "sbx-3f9a1c2e7b10-approve",
       "SaleReferenceIdentification": "INV-1001",
       "IdempotencyKey": "7d0c6f4e-2b1a-4c3d-8e9f-0a1b2c3d4e5f",
       "TransactionDetails": {"Currency": "AUD", "TotalAmount": "12.50",
         "SaleItem": [{"ProductDescription": "Flat white", "ProductQuantity": 2,
                       "UnitPrice": "6.25", "ProductAmount": "12.50"}]}}'
```

Every amount is a string with exactly two decimals, for example `"12.50"`. The currency is `AUD`. The items are optional. The terminal shows them if it can. It always charges `TotalAmount`.

A store's own surcharge goes inside `TotalAmount`. An example is 15% on a public holiday. The store sets this surcharge in your POS. Your register adds it to `TotalAmount` before it sends the sale. FUNDiMO has no surcharge setting, and adds no amount. The terminal adds nothing. In the basket's header, `DetailedAmount`, put the surcharge in `Surcharge`, without its tax. Put its GST in `ValueAddedTax`. The sum of the items is then `TotalAmount` less `Surcharge`. A bill of `50.00` with 15% on a public holiday is a sale of `57.50`:

```
"TransactionDetails": {"Currency": "AUD", "TotalAmount": "57.50", "DetailedAmount":
  {"AmountGoodsAndServices": "45.45", "ValueAddedTax": "5.23", "Surcharge": "6.82"}}
```

Response: `202`. It holds the sale's `TransactionReference`, and `TransactionStatus` `PDNG` (pending). FUNDiMO has already sent the instruction to the terminal.

Refusals

Status	Code	What to do
400	invalid_field	`Error.Fields` names each field that is wrong. `missing_field` and `unknown_field` do the same. Correct the sale, and send it again.
401	sandbox_key_invalid	Check the sandbox key.
403	not_permitted	An id is not in your world. Check the store and terminal ids.
409	terminal_offline	FUNDiMO created nothing. Send the sale again with the same idempotency key.
409	terminal_busy	Another sale holds the terminal. FUNDiMO created nothing, and the idempotency key is not used. The cashier can wait, or select another terminal.
503	store_unavailable	FUNDiMO possibly made a sale and lost its response. Retry with the same idempotency key. Never use a new one.

1.3 Read the sale until it settles

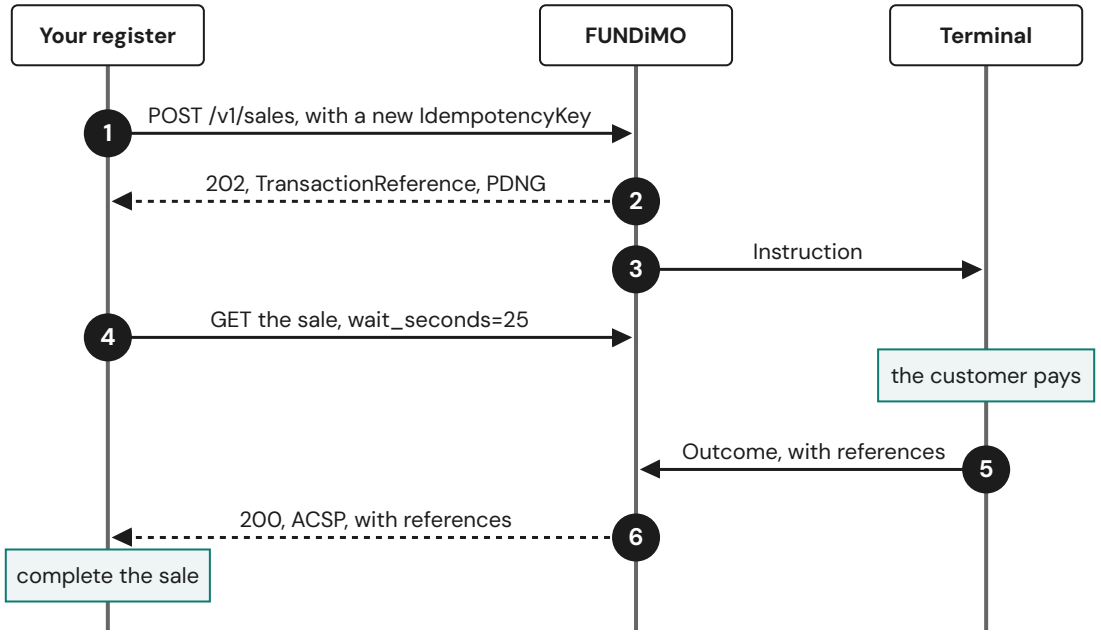
Read the sale with the `TransactionReference` from the `202`:

```
curl -s -H 'Authorization: Bearer sbx_...' \
  "$SANDBOX/v1/sales/sal_01970f6e7c3a7d3e8a4f2b1c9d0e1f2a?wait_seconds=25"
```

The read waits up to 25 seconds for the result to change. Then it responds. Read again until the sale settles.

Response: `200` and the sale. Its `TransactionStatus` tells your register what to show. Use your own words:

Waiting	Approved	Not paid	Unknown
For the customer, on the terminal.	Complete the sale. Print the references.	No payment was taken. Ask for another tender.	Check the terminal before another tender.
`PDNG`: read again	`ACSP`	`RJCT`, any reason	`UNKN`: continue to read



A sale, from start to end. FUNDiMO sends the instruction when it accepts the sale (1 to 3). Your register holds a read until the result changes (4 to 6).

With `RJCT`, the `Reason` in `StatusReasonInformation` gives the cause:

- `AG07`: the card or the host declined.
- `AB11`: no payment was taken, for a different cause.
- `DS02`: your register cancelled the sale (step 1.7).

Each of these means that no payment was taken.

An approved sale has the terminal's references in `PaymentReference`, for example the authorisation code. Print them on the receipt. Record `TransactionDetails.AuthorisedAmount` as the amount charged. On a refund, record it as the amount refunded. It is the amount that the host approved, and it is equal to `TotalAmount`. A lower amount means that the rest of the bill, or of the refund, is still owed. A higher amount is more than you asked for. In the two cases, the terminal did not obey the contract, and FUNDiMO logs it. `AuthorisedAmount` can be absent.

Read a status that you do not know as `UNKN`. Ignore a field that you do not know.

Refusals

Status	Code	What to do
404	sale_not_found	For an id that FUNDiMO gave you, this does not mean "no sale". Show the unknown screen.
503	store_unavailable	This tells you nothing about the sale. Read again.
400	invalid_wait	wait_seconds must be a whole number from 0 to 30.

1.4 Try every test terminal

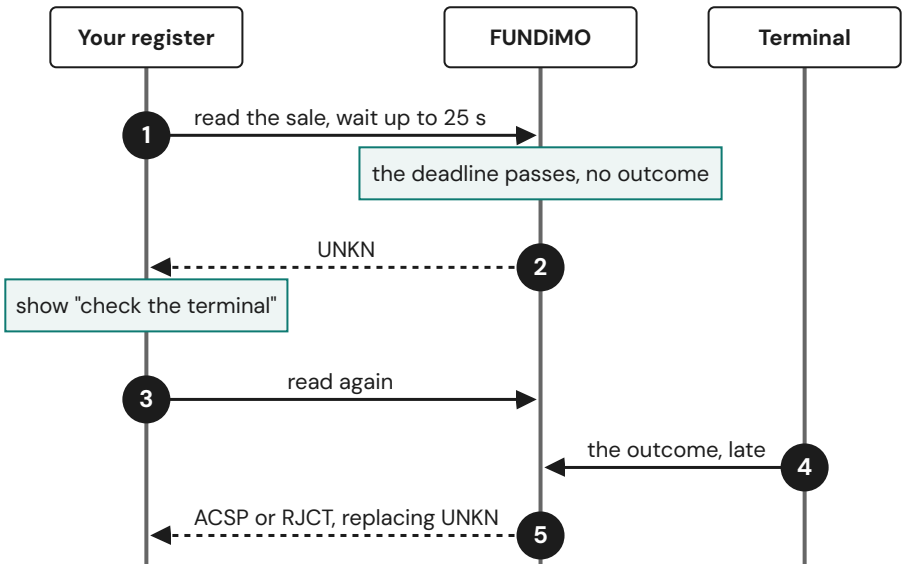
Send the same sale to each terminal of your world. Use a new IdempotencyKey each time. Check what your register shows.

Terminal	What it does	Your register shows
-approve	ACSP after 500 ms, with AuthorisedAmount and card references	Approved
-decline	RJCT, Reason AG07: the card or the host declined	Not paid
-technical	RJCT, Reason AB11: the terminal stopped, and took no payment	Not paid
-slow	The same as -approve, after 8 seconds	Waiting, then approved
-late	After the deadline plus 15 seconds: UNKN, then ACSP	Unknown, then approved
-silent	Never responds. The sale reads UNKN	Unknown
-card	The same as -approve, with four card fields that a terminal must never send. FUNDiMO removes them	Approved
-manual	Waits for your response through the MCP server	What you sent
-yours	Your own terminal, if you also make one	

The sandbox's deadline is 30 seconds. The contract's default is 120 seconds. A sale with no outcome at the deadline reads UNKN.

A sale that reads UNKN. A terminal that responds to FUNDiMO's status requests gets the result from its host. It reports the result when it knows it. If its host responds, this is before the terminal's hold ends. The hold ends 300 seconds after the sale, at the default settings. After the hold, a sale that still reads UNKN is not resolved yet. The terminal's result can still come, however late. It then replaces UNKN. Thus the cashier must still check the terminal, and your register must continue to read.

FUNDiMO's people list the sales that read UNKN. They check each sale against the terminal and the acquirer's record. Your register cannot read that list. It reads its own sales, one at a time.



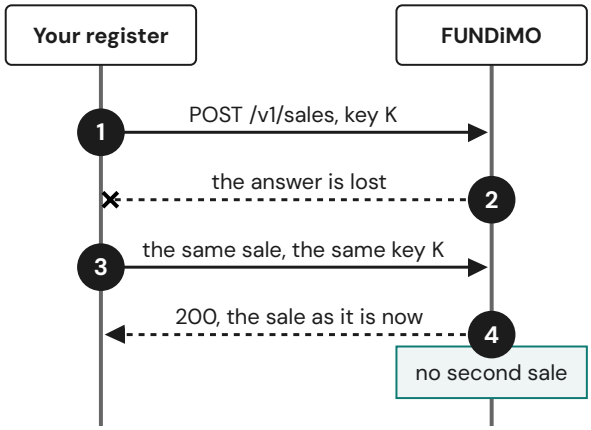
The unknown outcome and its recovery, as `-late` shows it. Show "check the terminal" when the sale reads `UNKN` (2). Continue to read (3). The late outcome replaces `UNKN` (4, 5). While the sale reads `UNKN`, a payment was possibly taken.

1.5 Recover from a lost response

A network can lose the response to your `POST`. Your register then cannot know if the sale exists. Send the same sale again. Use the same `IdempotencyKey` and the same register. Send the same items in the same order.

Response: `200`, and the sale as it is now. One idempotency key never makes a second sale.

Refusal
`409 idempotency_conflict`: the idempotency key names a sale with a different body. After a lost response, a sale can exist with that idempotency key. Thus read it as unknown, and show "check the terminal".



A lost response (2). Your register sends the sale again with the same idempotency key (3). FUNDiMO responds with the sale that it already has (4).

1.6 Send a refund

You can refund all or part of a sale that reads `ACSP`. Send a `POST /v1/sales` for the refund, with these items:

- `TransactionType` `RFND`.
- The sale's `TransactionReference`, in `OriginalTransaction`.
- The amount, in `TotalAmount`.
- Your own reference, for example a credit note number.
- A new `IdempotencyKey`.

A sale sends no `TransactionType`, or `CRDP`. The refund's terminal can be different from the sale's terminal.

```
curl -s -X POST "$SANDBOX/v1/sales" \
  -H 'Authorization: Bearer sbx_...' \
  -H 'Content-Type: application/json' \
  -d '{"TransactionType": "RFND",
       "OriginalTransaction": {"TransactionReference": "sal_01970f6e7c3a7d3e8a4f2b1c9d0e1f2a"},
       "MerchantIdentification": "sbx-3f9a1c2e7b10-M1",
       "POIIdentification": "sbx-3f9a1c2e7b10-approve",
       "SaleReferenceIdentification": "CN-1001",
       "IdempotencyKey": "0b6e2f4a-9c1d-4e7f-8a3b-5c6d7e8f9a0b",
       "TransactionDetails": {"Currency": "AUD", "TotalAmount": "6.25",
         "SaleItem": [{"ProductDescription": "Flat white", "ProductQuantity": 1,
                       "UnitPrice": "6.25", "ProductAmount": "6.25"}]}}'
```

Response: `202`, with the refund's own `TransactionReference`. Read the refund as you read a sale:

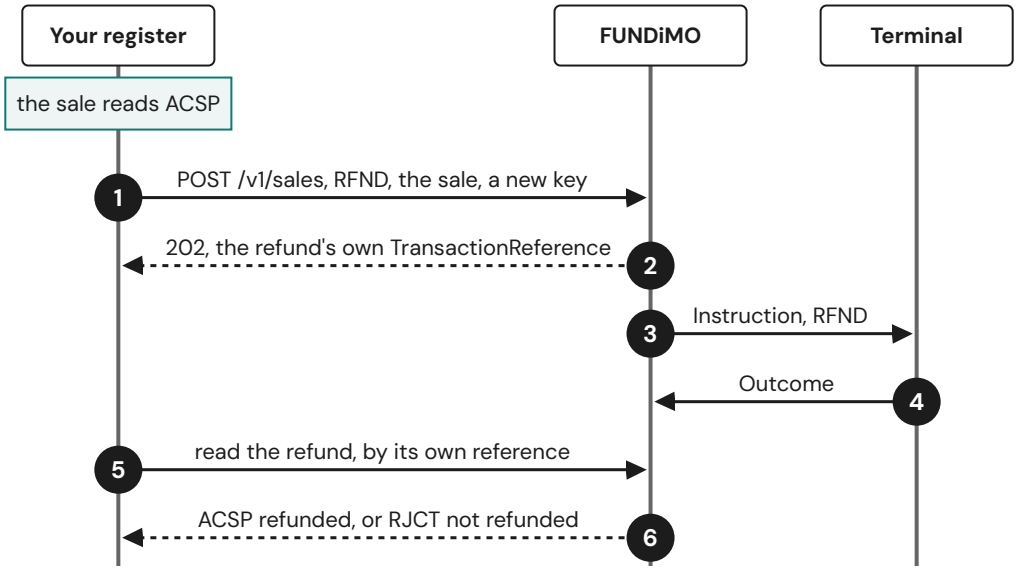
- `ACSP`: refunded. `AuthorisedAmount` is the amount refunded.
- `RJCT`: not refunded, for any `Reason`. This includes `DS02` after a cancel from your register.
- `UNKN`: nobody knows yet. Tell the cashier to check the terminal before a refund by a different method. Continue to read. Never send the refund again with a new idempotency key.

FUNDiMO keeps no total of the refunds of a sale. Your register shows the cashier the amount that it already refunded, from its own records.

Refusals

Status	Code	What to do
`409`	`sale_not_refundable`	FUNDiMO cannot refund that sale here. Refund at the terminal, or by a different method.
`409`	`terminal_cannot_refund`	That terminal did not declare that it does refunds. Select a terminal that did.

These refusals create nothing. The idempotency key stays unused.



A refund. It is a sale in the opposite direction, with its own reference (2). Your register reads it as a sale (5, 6).

1.7 Cancel a sale in progress, if your register has a cancel button

A cancel button is optional. A POS does not have to have one. A register with a cancel button can cancel a sale or a refund in these conditions:

- The sale reads `PDNG`.

- The sale reads `UNKN`, and its terminal's hold has not ended. The hold ends 300 seconds after the sale, at the default settings.

Send no body, with `Content-Length: 0`. Over HTTP/1.1, the network edge in front of FUNDiMO refuses a cancel without that header with `411`, and the cancel does not get to FUNDiMO. Use the same credential as for the sale: the register that made the sale cancels it.

```
curl -s -X POST -H 'Authorization: Bearer sbx_...' -H 'Content-Length: 0' \
  "$SANDBOX/v1/sales/sal_01970f6e7c3a7d3e8a4f2b1c9d0e1f2a/abort"
```

Response: `202`, and the sale as it is now, `PDNG` or `UNKN`. This is not the result of the cancel. FUNDiMO has asked the terminal to stop the payment. Read the sale until it settles, as in step 1.3. The final status tells you what the terminal did:

- `RJCT` with `Reason` `DS02`: the terminal stopped the payment before the payment got to its host. The sale is cancelled, and no payment was taken. On a refund, no refund was made.
- `ACSP`: the cancel was too late. The payment got to its host, and the host took it. Complete the sale, and record its `AuthorisedAmount`. A refund that the host approved stays approved.
- `RJCT` with a different `Reason`: no payment was taken, as on any read.
- `UNKN`: as on any read, the cashier checks the terminal before another tender.

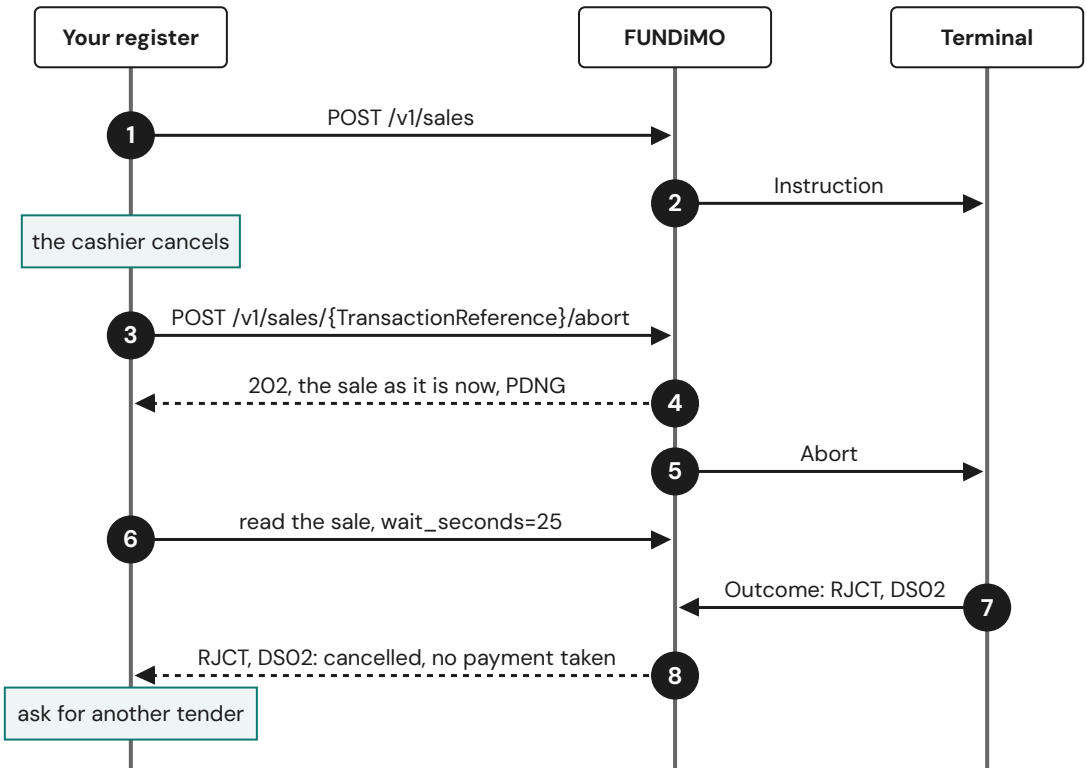
A cancel does not change a timer. If the terminal does not respond by its deadline, the sale reads `UNKN`. A later `DS02` replaces `UNKN`. The terminal stays claimed until it responds or its hold ends. Until then, a new sale to it gets `409 terminal_busy`.

To try a cancel, send a sale to `-slow`. Its 8 seconds give you the time. Cancel the sale immediately. The sale reads `RJCT` with `DS02`.

Refusals

Status	Code	What to do
`409`	`terminal_cannot_abort`	The terminal did not declare that it can cancel a sale. Tell the cashier to cancel at the terminal. Then read the sale. A retry does not help.
`409`	`sale_not_abortable`	The sale has ended, or its terminal's hold has ended. Read the sale. `ACSP` means that the cancel was too late.
`409`	`terminal_offline`	The terminal has no connection, or its queue is full. Cancel again, or tell the cashier to cancel at the terminal.
`404`	`sale_not_found`	For an id that FUNDiMO gave you, read the sale as unknown. Show "check the terminal".
`503`	`store_unavailable`	This tells you nothing about the sale. Cancel again, or read the sale.

After each refusal, FUNDiMO sends nothing to the terminal, and the sale continues as before. In the sandbox, `-silent` declares no cancel. A cancel of its sale gets `409 terminal_cannot_abort`, while the sale waits and during the hold.



A cancel from your register. The register asks (3). FUNDiMO responds immediately with the sale as it is now (4). FUNDiMO sends the cancel to the terminal (5). The terminal stops the payment before the payment gets to the host, and sends `RJCT` with `DS02` (7). The register's read returns it (8).

1.8 Enrol your server for its certificate

On staging and production, your server identifies itself with a client certificate, not a sandbox key. Practise this in the sandbox. The commands of the server walk read the address from `$MW`:

```
MW=$SANDBOX
```

The pairing code. A pairing code is a one-use secret. A server spends it to enrol for its certificate. It has no relation to linking in section 3.

The response to your sandbox key (step 1.1) held a pairing code. It is valid for 15 minutes. If it has expired, get a new one. You can get 6 each hour:

```
curl -s -X POST "$SANDBOX/sandbox/v1/codes" \
  -H 'Authorization: Bearer sbx_...' \
  -H 'Content-Type: application/json' \
  -d '{"Role":"server"}'
```

A server's pairing code has 16 characters, in four groups of 4. It is valid for 15 minutes and for one use. It is a secret.

Make a P-256 private key, and keep it in your own custody. Make a certificate signing request (CSR) for it:

```
umask 077
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -out server.key
openssl req -new -key server.key -subj '/CN=server' -sha256 -out server.csr
```

Then spend the pairing code. The call has no credential, because the pairing code is the credential:

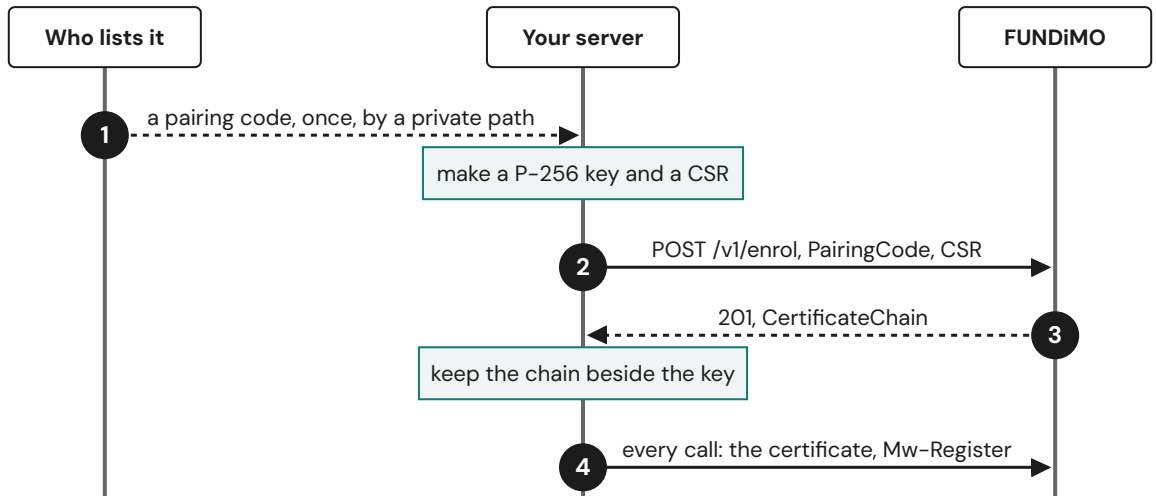
```
read -r -s -p 'Pairing code: ' CODE; echo
status=$(jq -n --arg code "$CODE" --rawfile csr server.csr \
    '{PairingCode: $code, CertificateSigningRequest: $csr}' \
  | curl -s -X POST "$MW/v1/enrol" \
      -H 'Content-Type: application/json' \
      --data-binary @- \
      -o enrolled.json -w '%{http_code}')
case "$status" in
  200|201) jq -r .CertificateChain enrolled.json > chain.pem ;;
  *) echo "not enrolled: HTTP $status, Error.Code in enrolled.json" >&2 ;;
esac
```

Response: `201`, with `CertificateChain`. This is your server's certificate, then the certificate that issued it. Keep it as `chain.pem`, next to `server.key`. The certificate is valid for 30 days, to `NotAfter`. You can renew it from day 15, `RenewFrom`. If you lose the response, send the same body again within the 15 minutes. FUNDiMO responds `200` with the same certificate. Never make a new private key to repeat a call that got no response.

Refusals

Status	Code	What to do
403	`code_not_valid`	The pairing code is wrong, expired or spent. Get a new one.
400	`invalid_field`	The CSR's signature is not valid, or its key is not P-256. Make a new private key and CSR. Enrol again within the 15 minutes.
503	`ca_unavailable`	The pairing code is not spent. Wait, and send the same body again. Do the same for `store_unavailable`.
429		If the body is not an `Error`, the network edge in front of FUNDiMO set this limit. Wait one minute, and send the same body.

After five refusals, a pairing code is unusable. Correct the cause of a refusal before you try again.



Enrolment of your server. In the sandbox, the pairing code comes with your sandbox key. On staging and production, FUNDiMO lists your first server. FUNDiMO gives you its pairing code one time only (step 2.1).

1.9 Send sales on your certificate

From now on, your server presents its certificate at the TLS handshake of every call. It names the register that it acts for in the register's header, `Mw-Register`. The header holds the register's id, `RegisterIdentification`. Do not send an `Authorization` header with a certificate. Make sure that the certificate works:

```
curl -s "$MW/v1/certificates/self" --cert chain.pem --key server.key
```

Response: `200`, with `NotAfter`, `RenewFrom` and `RenewNow`. Then send the sale of step 1.2 again. Replace the `Authorization` header with `--cert chain.pem --key server.key -H 'Mw-Register: reg_...'`. Read the sale in the same way.

Refusals

Status	Code	What to do
`401`	`certificate_invalid`	You sent a certificate and an `Authorization` header together. Send the certificate only.
`403`	`not_permitted`	`Mw-Register` is absent or sent two times, or it names a register that is not yours.

1.10 Run the conformance scenarios, and keep the report

A conformance run does the ten POS scenarios, one step at a time. It judges each step from your world's log. It needs your certificate, and it counts only the calls that have your certificate. Thus a sale that the MCP server sends does not count. A run of a POS that works correctly takes less than 3 minutes. You operate the run with the tools of the MCP server (The sandbox's MCP server):

1. Send `conformance_start` with `Side: "pos"`, `ProductName` and `ProductVersion`. It responds with the `RunIdentification` and the first `Step`. The `Step` gives its `Instruction`, its `Terminal`, and what it is `Waiting` for.
2. Do what the `Instruction` says. Use your own code, on your certificate.
3. Send `conformance_status` with the `RunIdentification`, and `WaitSeconds` up to 25. It judges the step and goes to the next step. Each step has 300 seconds.
4. When `Result` is no longer `running`, send `conformance_report`. It gives the signed report. Save the report.

Scenario	Terminal	What your code does
`approved`	`-approve`	Send one sale. Read it until it settles
`declined`	`-decline`	The same, to a decline
`timed_out`	`-technical`	The same, to a terminal that stops
`unknown`	`-late`	Read `UNKN`. Then continue to read until `ACSP`
`card_field`	`-card`	Make sure that FUNDiMO removed the card fields
`duplicate_key`	`-approve`	Send a sale again with the same idempotency key. Then send it with a different body
`resend`	`-approve`	FUNDiMO loses the response to your first sale on purpose. Send the same body with the same idempotency key
`restart`	`-slow`	An outage of 5 seconds starts at your first read. Retry with the backoff
`lost_instruction`	None	Not run. Nothing changes for a POS. It reads the terminal's result as any late outcome
`cancel`	None	Not run. A POS does not have to have a cancel button. A POS with one reads the results of a cancel as in step 1.7

A green report shows the two scenarios that are not run as `not_applicable`.

Response: `Result` `green`, `red`, `undetermined` or `partial`. `Done` gives the result of each scenario. The report is `Report`, `Signature` and `KeyIdentification`, signed with Ed25519. `sandbox.md` § 6 shows how to check it with OpenSSL. A green report is a test result. It is not a certification from a card scheme.

Refusals

Status	Code	What to do
	`certificate_required`	Enrol first (step 1.8).
	`run_not_finished`	The run has not ended. Read `conformance_status`.

Sometimes the log lines of a step leave your world's log before FUNDiMO judges the step. The step is then `undetermined`. Run again with less traffic. A step that fails names its assertion, the `seq` of the log line, and what to change. `mw://conformance` holds every assertion.

> **A green run ends this section.** Keep the signed report. Its `SandboxVersion` names the release of the sandbox that ran it. Compare it with staging's release before you start section 2.

SECTION 2

Enrolment

In this section, your server gets its certificate in an environment, and your registers get their ids. Do this section on staging first. Then do it on production. The steps are the same in the two environments. Only the address changes, and with it the pairing code, the private key and the certificate. A credential belongs to the environment that issued it. Thus enrol in each environment with its own pairing code. Never use a copied private key.

> **You cannot enrol alone.** Your first server needs a pairing code that only FUNDiMO gives, one time only (step 2.1). Ask FUNDiMO for it in each environment, and ask for an `admin` server (Get help). After that, your `admin` server can list more servers and each register on a device, and get their pairing codes. FUNDiMO's onboarding team can also list a register for you, and give its first pairing code. The terminals at a store are the store's fleet vendor's. The fleet vendor enrols them, not you.

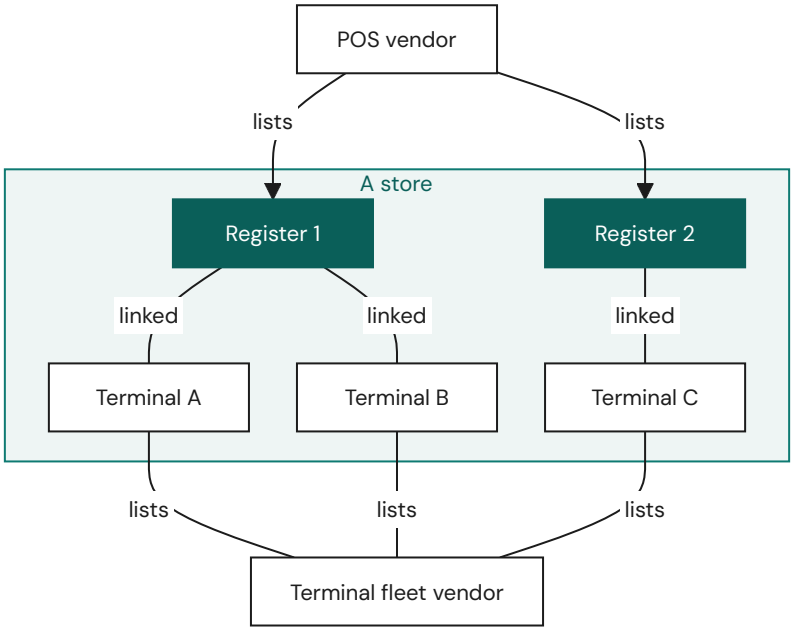
	Staging	Production
Address	`MW=https://mw.staging.fundimo.com.au`	`MW=https://mw.fundimo.com.au`
For	Tests with real terminals, on test cards only. Never use a real card	Real merchants
You need	A terminal fleet: real terminals at a store that accepts your POS vendor	The terminal fleet at your merchants' stores

Before staging. Compare staging's release with your report's `SandboxVersion`. Staging's release is `Version` in the response to `GET $MW/v1/health`. The two releases can be different for a time.

Then production. When your registers take test payments on staging's real terminals (section 3), you are ready for real merchants. Do this section and section 3 again, with `MW` set to production's address.

How enrolment fits together:

- Your `admin` server, or FUNDiMO, lists each register. A register belongs to one POS vendor only.
- A register can link to several terminals of one store.
- A terminal links to one register only.
- A register is at no store until it links to a terminal there. When its last link ends, it can link at a different store.
- A store can have registers of more than one POS vendor.
- A terminal fleet vendor lists each terminal at one store.



One store, with two registers of one POS vendor. Register 1 is linked to two terminals, and register 2 to one. Each terminal is linked to one register. The terminal fleet vendor lists each terminal at the store.

2.1 Get your first server listed

FUNDiMO lists your first server. FUNDiMO gives you its pairing code one time only, by a path that nobody else reads. You cannot get this pairing code in a different way: ask FUNDiMO for it (Get help). This pairing code has 16 characters. It is valid for 15 minutes and for one use.

Each server is a separate identity, with its own pairing code, private key and certificate. If you list your own registers, you have two servers:

- A `pos` server, for sales and linking.
- An `admin` server, for your back office.

Ask FUNDiMO for an `admin` server if your back office lists your registers and other servers. Your `admin` server can then list more servers:

```
curl -s -X POST "$MW/v1/vendors/ven_.../servers" \
  --cert admin-chain.pem --key admin.key \
  -H 'Content-Type: application/json' \
  -d '{"Scope":"pos"}'
```

Response: `201`, with `ServerIdentification` and `PairingCode`. FUNDiMO shows them one time only.

Refusals

Status	Code	What to do
`409`	`too_many_keys`	You can have a maximum of two servers of one scope. Remove one first.

A pairing code that expires before you spend it does not renew. Remove that server, and list a new one.

2.2 Enrol your server

Set `MW` to the address of the environment. Enrol as in step 1.8. Use a new private key and that environment's pairing code. Never use the sandbox's private key on staging. Never use staging's private key on production.

Sometimes FUNDiMO refuses a new pairing code with `403 code_not_valid`, although you sent it as listed and within its 15 minutes. Treat this as an alarm: possibly another person spent it. `onboarding-server.md`, step 4, shows how to read your servers. Compare the `SpkiDigest` of each server with the digest of your private key.

2.3 List each register

Your `admin` server lists each register at `/v1/registers`, the register's route:

```
curl -s -X POST "$MW/v1/registers" \
  --cert admin-chain.pem --key admin.key \
  -H 'Content-Type: application/json' \
  -d '{}'
```

Response: `201`, with `RegisterIdentification` (the register's id), `Kind server` and `State active`. Your `pos` server names the register in `Mw-Register` on that register's calls. The register can send sales to no terminal until you link it to one at the counter (step 3.1).

Give each register its own `RegisterIdentification`. Never send the sales of two registers with one `RegisterIdentification`. A retry must name the same register as the first send. If it does not, it makes a second sale.

> **If your register is an app on a device.** A register on a device holds its own certificate. Thus your `admin` server lists it with the serial number on the device: `POST /v1/registers` with its `SerialNumber`. The response holds the device's pairing code. FUNDiMO's onboarding team can also list the register and give this pairing code (Get help). This pairing code is five different words of FUNDiMO's word list, `/docs/api/words.txt`. The words are in lower case and joined by dots, for example `bronze.farm.loop.prism.tempo`. The pairing code is valid for 15 minutes and for one use. The installer opens your app's enrolment screen from the app's own menu or settings, with only the device. The installer types the words, or scans them from a QR code. The app then enrols as `onboarding-device.md` shows. Its section "The enrolment screen" gives the requirements of that screen. In the sandbox, a register on a device enrols with a server's pairing code (step 1.8). The sandbox does not have a register's own pairing code yet.

2.4 Automate the renewal of every certificate

Your POS software must renew each certificate automatically. No person does it. Once a day, your software asks FUNDiMO if it must renew. From day 15, the answer is yes, and your software renews. After its `NotAfter`, a certificate cannot renew. Its server then needs a new pairing code, from a new listing. Thus put the renewal in your software. Each certificate in use must have one live instance that renews it.

The daily question is `GET /v1/certificates/self`, on the certificate. To test it manually, send it with `curl`:

```
curl -s "$MW/v1/certificates/self" --cert chain.pem --key server.key | jq -r .RenewNow
```

Response: `true` from day 15. Your software then does these steps:

1. It makes a new private key and CSR.
2. It sends `POST /v1/certificates/renew` on the current certificate. The body holds `CertificateSigningRequest` only.
3. It installs the new private key and chain on each copy of your server.
4. It deletes the old private key only after FUNDiMO responds to a call on the new certificate.

The old certificate stays valid for one hour after the first use of the new certificate. `onboarding-server.md`, step 9, has the renewer's daily script. It also has the script that each copy runs to use the new certificate.

One renewer for each certificate:

- A register that runs as an app on a device holds its own certificate. The app renews it, as `onboarding-device.md`, steps 9 and 10, show. The app sets its own daily check.
- A register that your server lists holds no certificate. Your `pos` server speaks for it.
- All copies of one server use the same private key and certificate. Name one copy as the renewer, in its configuration. Do not let the copies race. The other copies get the new private key and chain from the renewer, well within the hour.
- Any copy can ask the daily question. Only the renewer acts on the answer.

A second renewer causes problems. It retires the first renewer's new certificate, and FUNDiMO reads this as a sign of a copied private key. Or FUNDiMO refuses it with `403 certificate_retired`. After 5 certificates in 24 hours, FUNDiMO refuses every renewal with `429 renewal_rate`.

Refusals

Status	Code	What to do
`409`	`renewal_not_due`	Wait until `RenewNow` is `true`.
`429`	`renewal_rate`	FUNDiMO issued 5 certificates in 24 hours. The most likely cause is a second renewer, or a copy of your private key. Find it.
`403`	`certificate_retired`	On your newest certificate: stop, and tell a person. Something else holds a newer certificate.
`401`	`certificate_expired`	The certificate cannot renew. The server needs a new pairing code, from a new listing.

Never delete a private key or a certificate because of a refusal.

SECTION 3

Link

Linking connects a register to the terminal next to it. It pairs the register and the terminal: the register's sales then go to that terminal. The calls are under `/v1/links`, and `linking.md` describes them. Link on staging first, with real terminals and test cards. Then link on production, at your merchants' stores. The steps are the same, at production's address. Only the cashier at the counter is necessary.

Linking is your responsibility. You are the POS vendor, and you link each register to its terminals. Nobody else does it for you. FUNDiMO refuses a sale that names a terminal that is not linked to the register: `403 not_permitted`. FUNDiMO creates nothing, and the idempotency key stays unused. This refusal also has other causes, for example a store that is not live. FUNDiMO gives one code for all of them.

Before you link, make sure of these conditions:

- The register is listed (step 2.3), and your `pos` server is enrolled (step 2.2).
- The terminal's fleet vendor listed the terminal at the store. The terminal is enrolled and connected.
- The terminal is not linked to a register yet.
- The store accepts your POS vendor. Every store does this until its fleet vendor changes it.
- Your register has the store's `MerchantIdentification`. The fleet vendor gives it to the merchant.

3.1 Link each register to its terminal, at the counter

Your register sends the request:

```
curl -s -X POST "$MW/v1/links" \
  --cert chain.pem --key server.key \
  -H 'Mw-Register: reg_...' \
  -H 'Content-Type: application/json' \
  -d '{"MerchantIdentification":"mid_..."}'
```

Response: `201`, with `LinkIdentification` and `ExpiresInSeconds`, 120 by default. Each terminal at the store that is not linked now shows two words, for example `rabbit.lamp`. Your register never sees the words.

Ask the cashier to type the two words from the terminal in their hand. A dot or a space goes between the words. While you wait, `GET /v1/links/{LinkIdentification}` gives `Asked` and `Shown`. While `Shown` is 0, no terminal shows its words yet. Then send what the cashier typed:

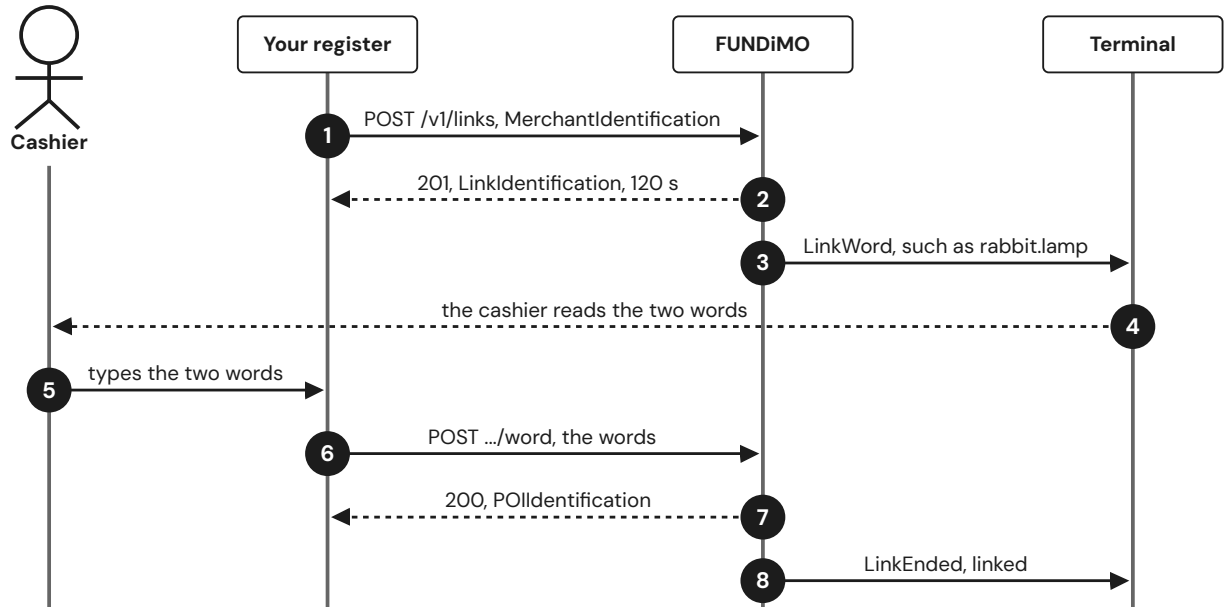
```
read -r -p 'Words on the terminal: ' WORD
jq -n --arg word "$WORD" '{Word: $word}' \
  | curl -s -X POST "$MW/v1/links/lnk_.../word" \
      --cert chain.pem --key server.key \
      -H 'Mw-Register: reg_...' \
      -H 'Content-Type: application/json' \
      --data-binary @-
```

Response: `200`, with the terminal's `POIIdentification`. Keep it with the register, because the register's sales name it. The terminal shows that it is linked.

Refusals

Status	Code	What to do
400	invalid_field	Not two words. Ask for the two words, in the order shown. The request stays open.
403	link_word_unknown	The words are not correct. Offer to ask again, for new words.
409	link_in_progress	Another register at the store is linking. Try again in one minute.
409	no_terminal_to_link	No terminal is waiting. Make sure that it is on, enrolled at this store, connected, and not linked.
410	link_expired	The words expired. Ask again.
429	link_rate	FUNDiMO stopped linking at this store for a time. Wait one hour. After three wrong pairs of words, wait until tomorrow.
403	not_permitted	This register cannot link at this store. Check the store's MerchantIdentification. Ask the store's fleet vendor.

Never log the words, and never keep them. linking.md has the full table of what your register shows for each response. In the sandbox, your register is linked to every test terminal from the start. Thus a request gets 409 no_terminal_to_link. To try linking, unlink -yours, and connect a terminal of your own.



Linking at the counter. The words go to the terminal (3). Only a person who holds the terminal can read them (4). Your register sends what the cashier typed (6).

3.2 Send a first sale through the linked terminal

Send a sale as in step 1.9, on your certificate. Name the POIIdentification from step 3.1. Read the sale as in step 1.3. On staging, the cashier uses a test card. Never use a real card on staging. On production, your register takes real payments for a real merchant.

What green conformance means for your POS software

A green conformance run means one thing: your POS software did each step one time, in the sandbox. It is not a certification. On every other day, your POS software must obey these rules.

- **The users of your POS.** The people who use your POS are your responsibility. When your POS sends a sale or a refund, FUNDiMO takes it that a user of the POS was authenticated to make it. FUNDiMO judges nothing about that user.
- **The idempotency key.** Your register's idempotency key for one payment attempt has 8 to 64 letters, digits and `. _ : / -`. Make a new one for each sale and each refund. For a retry, send the same idempotency key with the same sale, from the same register. An idempotency key lasts as long as its sale. By default, FUNDiMO forgets a sale 24 hours after it stopped being pending. After that, the idempotency key starts a new sale.
- **The unknown screen.** Show "check the terminal" when a sale reads `UNKN`. Do not wait until the reads stop. Show it also for `404 sale_not_found` on an id that FUNDiMO gave you. Show it also for `409 idempotency_conflict` after a lost response. A payment was possibly taken. Continue to read. Take no other tender, and no refund by a different method, until the cashier has checked the terminal.
- **No card, bank or personal data** in a field that you fill. This includes an item's description and your invoice number. Send no card number, expiry, track data, BSB, account number or name. FUNDiMO cannot find these in a string that you choose. FUNDiMO logs every message.
- **One register, one id.** A sale belongs to its register. Its idempotency key is unique in the register. Only that register can read the sale.
- **A cancel is a request, not a result.** The `202` of `POST /v1/sales/{TransactionReference}/abort` is the sale as it is now. Continue to read. `RJCT` with `DS02` means cancelled. `ACSP` means that the payment was taken. If the customer cancels at the terminal, the sale reads `RJCT` with `AB11`.
- **The amount charged.** Record the `TransactionDetails.AuthorisedAmount` of an approval as the amount charged, or refunded. You add the store's own surcharge to `TotalAmount`. Nobody else adds one.
- **Renewal is automatic.** Your software renews each certificate from day 15, with one renewer for each certificate (step 2.4).
- **Read what you do not know safely.** Read a status that you do not know as `UNKN`. Ignore a field that you do not know. `RJCT` means no payment, for any `Reason`.

The sandbox's MCP server

What it is. The sandbox runs an MCP (Model Context Protocol) server at `POST /mcp`. With it, an AI coding assistant can read FUNDiMO's pages and check a sale before you send it. The assistant can also send test sales, operate a test terminal and read your world's log.

Who it is for. Your developer's AI coding assistant, in the sandbox only. Your register's own code calls the HTTP routes of section 1. A conformance run counts only the calls that have your certificate.

Connect your assistant. You can use any MCP client on revision `2026-07-28` or `2025-11-25`. It connects over Streamable HTTP at `$SANDBOX/mcp`, with your sandbox key on every request. There is no session and no OAuth. The pages give this command for the command line of one assistant:

```
claude mcp add --transport http mw-sandbox "$SANDBOX/mcp" --header "Authorization: Bearer sbx_..."
```

Then start a new session, because a client reads its servers when a session starts. You can also call the server over plain HTTP, with no client. Keep the line with your sandbox key in a file that only you can read. Send one request at a time:

```
touch ~/.mw-sandbox-auth && chmod 600 ~/.mw-sandbox-auth
(read -rs key && printf 'Authorization: Bearer %s\n' "$key" > ~/.mw-sandbox-auth)
curl -s "$SANDBOX/mcp" -H @"$HOME/.mw-sandbox-auth" \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json, text/event-stream' \
  -H 'MCP-Protocol-Version: 2025-11-25' \
  -d '{"jsonrpc":"2.0","id":1,"method":"resources/read","params":{"uri":"mw://world"}}'
```

Without a sandbox key, you can read every resource except `mw://world`. Also, `describe_field`, `validate_message` and `explain_refusal` respond. Every other tool responds `key_required`. A sandbox key that is unknown or expired gets `key_not_recognised`. A staging or production credential gets `401 credential_not_for_sandbox`. If you sent one, rotate it.

What it offers

Resource	Key	What it is for
`mw://start`	No	The sandbox's start page, `sandbox.md`
`mw://environments`	No	The two environments that the pages name, the sandbox and staging, and the address of each
`mw://world`	Yes	Your world's ids, its expiry, and the timers and TLS mode in force. How each terminal does a refund and responds to a `StatusRequest` and an `Abort`
`mw://fields`	No	The field catalogue: one row for each field of each message
`mw://fields/{Message}/{Path}`	No	One row, for example `mw://fields/Sale/TransactionDetails.TotalAmount`
`mw://refusals`	No	Every code that FUNDiMO and the sandbox give, what it means, and what to do next
`mw://conformance`	No	The conformance scenarios, and the assertions that judge each

`mw://docs/openapi.yaml`
`mw://docs/terminal.md`
`mw://docs/llms.txt`
	No	The OpenAPI document, the terminal contract and the index

`mw://onboarding/server`
`mw://onboarding/device`
`mw://onboarding/linking`
	No	Enrolment of a server, enrolment of an app on a device, and linking at the counter

Resource	Key	What it is for
`mw://report-key`	No	The public key that verifies a conformance report's signature

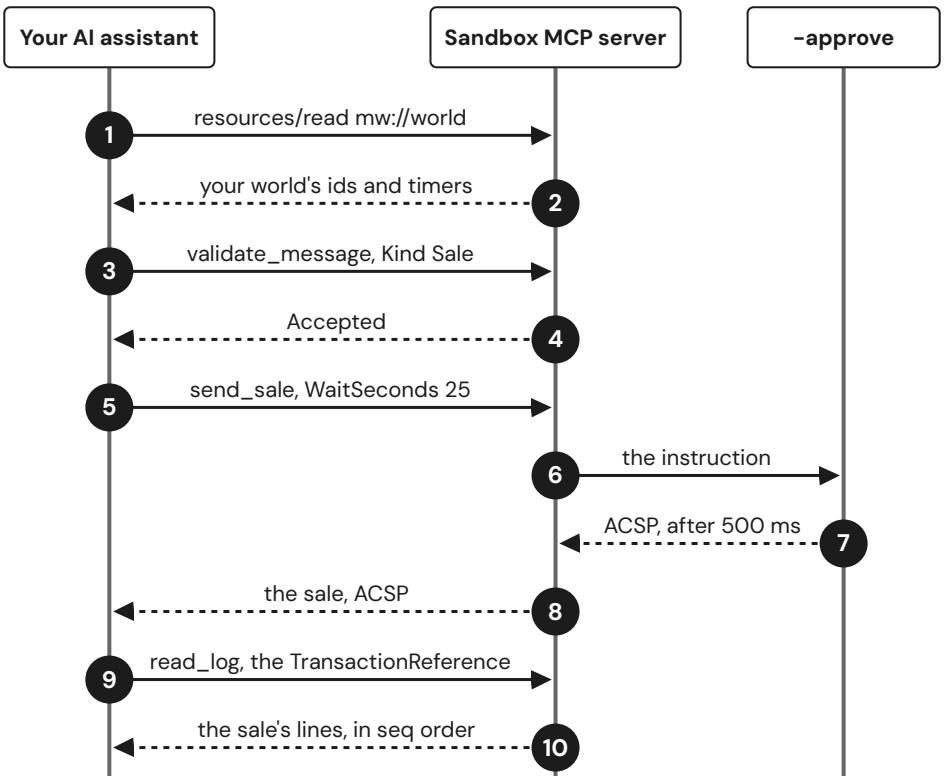
Tool	Key	What it is for
`describe_field`	No	The catalogue's rows for a field, by its `Path`
`validate_message`	No	Checks a `Sale` or an `Outcome` as FUNDiMO does. It gives `Accepted`, or the refusal with its fields, what FUNDiMO would remove (`Dropped`), and `Next`. It keeps nothing
`explain_refusal`	No	What a code means, what to do next, and which side gives it
`send_sale`	Yes	Sends a sale as your world's POS. It can wait up to 25 seconds for the result
`read_sale`	Yes	Reads one of your sales. It can wait for the sale to change
`terminal_next_instruction`	Yes	The next instruction, `StatusRequest` or `Abort` that FUNDiMO sent to your `-manual` terminal
`terminal_send_outcome`	Yes	Sends an outcome as your `-manual` terminal
`read_log`	Yes	The log lines of the sales that a `TransactionReference` or a `SaleReferenceIdentification` names: the newest five, each in `seq` order

`conformance_start`
`conformance_status`
`conformance_report`
	Yes	Start a conformance run, judge and read it, and get its signed report (step 1.10)

An example

Ask your assistant: "Send a test sale to my approve terminal, and show me its log." The assistant does these steps:

1. It reads `mw://world` to get your store, your terminals and the timers in force.
2. It checks the sale with `validate_message`, `Kind` `Sale`. The result is `Accepted`, or the refusal with the fields that are wrong and `Next`.
3. It sends the sale with `send_sale`. The sale goes in `Message`, and `WaitSeconds` is 25. `-approve` approves after 500 ms, and the tool returns the settled sale.
4. It reads `read_log` with the sale's `TransactionReference`. The log gives every message of the sale, in the two directions, in `seq` order. A `sale_state` line gives the sale's state in FUNDiMO's words.



The assistant reads your world (1, 2) and checks the sale (3, 4). It sends the sale and waits (5 to 8). Then it reads the sale's log (9, 10).

If a tool fails, its result has `isError` `true`, an `Error`, and a `Next` that says what to do. FUNDiMO never logs a tool's arguments. No tool changes a credential: sandbox keys and pairing codes come only from the routes under `/sandbox/v1/`.

Get help

In the sandbox, before you sign with FUNDiMO. The sandbox is self-help. Use this guide and the MCP server first (The sandbox's MCP server). If you need more help, send an email to `support@fundimo.com.au`.

When you sign with FUNDiMO. FUNDiMO invites your team to a Slack Connect channel. The channel is private: only your organisation and FUNDiMO are in it. Use it for enrolment and linking, and for your questions on staging and production.

If your organisation does not use Slack. Send an email to `support@fundimo.com.au` instead.

> **Never send a secret in the channel or in an email.** Send no pairing code, no sandbox key and no private key. FUNDiMO never asks for one there. A pairing code goes only by a path that nobody else reads (step 2.1). It never goes in the channel.

The pages

Every page is public at `mw.sandbox.fundimo.com.au`. Each page gives the details that this guide does not give.

Page	What it is for
`/llms.txt`	The index of every page below
`/docs/api/sandbox.md`	Start here. Your sandbox key and test world, the POS journey, the MCP server, conformance, and every code that the sandbox adds
`/docs/api/environments.md`	The sandbox and staging: the address of each, the differences, and how to move from the sandbox to staging
`/docs/api/openapi.yaml`	The calls, the sale and its statuses, and every refusal with what to do
`/docs/api/onboarding-server.md`	Your server's pairing code, private key and certificate, with code for curl, Rust and Java. Also `Mw-Register`, the listing of registers, and renewal
`/docs/api/linking.md`	Linking a register to a terminal at the counter. What your register shows for each response
`/docs/api/onboarding-device.md`	For a register that is an app on a device
`/docs/api/fields.json`	Every field: who sends it, its ISO 20022 element if ISO has one, its limits, and an example
`/docs/api/conformance.json`	The conformance scenarios, and how FUNDiMO judges each
`POST /mcp`	The sandbox's MCP server, for your AI coding assistant

Addendum: the ISO 20022 terms in this guide

Each ISO 20022 term in this guide, with ISO's definition as `fields.json` and `terminal.md` quote it, without ISO's usage notes. Then what the term is in this guide.

Fields

Field	ISO 20022's definition, and what it is here
`MerchantIdentification` `Merchant/Identification`, `Mrchnt/Id`	"Identification of the merchant." The store's id. Every sale names it.
`POIIdentification` `POIId`	"Identification of the POI (Point Of Interaction) performing the transaction." The terminal that a sale goes to.
`SaleReferenceIdentification` `SaleRefId`	"Global reference of the sale transaction for the sale system." Your own reference, for example an invoice number.
`CashierIdentification` `CshrId`	"Identification of the cashier who carried out the transaction." Your id for the user of your POS who made the sale or the refund.
`TransactionReference` `TxRef`	"Identification of the transaction that has to be unique for a time period." FUNDiMO's id for a sale or a refund.
`TransactionType` `TxTp`	"Type of transaction being undertaken for the main service." `RFND` on a refund. A sale sends none, or `CRDP`.
`OriginalTransaction` `OrgnlTx`	"Identification of the original transaction." On a refund, the sale that it refunds.
`TransactionDetails` `TxDtls`	"Details of the transaction." The amount, its currency and the basket.
`Currency` `Ccy`	"Currency associated with the transaction." `AUD`.
`TotalAmount` `TtlAmt`	"Total amount of the transaction." What the customer pays, and what the terminal charges.
`DetailedAmount` `DtldAmt`	"Detailed amounts associated with the total amount of transaction." The basket's header.
`AmountGoodsAndServices` `AmtGoodsAndSvcs`	"Amount of purchase goods and services without tax." Goods and services before tax.
`ValueAddedTax` `ValAddedTax`	"Value added tax amount." The sale's GST.
`Surcharge` `Srchrg`	"Additional charge paid by the cardholder. For example airline credit card surcharge." The store's own surcharge on the bill, without its tax.
`SaleItem` `SaleItm`	"Item purchased with the transaction." One line of the basket.
`ProductDescription` `PdctDesc`	"Description of the product or item." The line's description.
`ProductQuantity` `PdctQty`	"Product quantity." How many.
`UnitPrice` `UnitPric`	"Price per unit of product." The price of one.
`ProductAmount` `PdctAmt`	"Monetary value of purchased product." The line's amount, after its discount and with its tax.
`AuthorisedAmount` `AuthrsdAmt`	"Amount authorised for the payment transaction." The amount that the host approved: what the customer paid, or on a refund what was refunded.

Field	ISO 20022's definition, and what it is here
`TransactionStatus` `TxSts`	"Specifies the status of a transaction, as published in an external payment transaction status code list." Where the sale is now.
`StatusReasonInformation` `StsRsnInf`	"Provides detailed information on the status reason." It holds the `Reason`.
`Reason` `Rsn`	"Specifies the reason for the status report." With `RJCT`, why no payment was taken.

Codes

Code	ISO 20022's name and definition, and what it is here
`PDNG` Status	Pending: "Payment instruction is pending. Further checks and status update will be performed." The terminal has not responded yet. Read again.
`ACSP` Status	AcceptedSettlementInProcess: "All preceding checks such as technical validation and customer profile were successful and therefore the payment instruction has been accepted for execution." The host approved the full `TotalAmount`.
`RJCT` Status	Rejected: "Payment instruction has been rejected." No payment was taken, for any `Reason`.
`AG07` Reason	UnsuccesfulDirectDebit: "Debtor account cannot be debited for a generic reason." The card or the host declined.
`AB11` Reason	TimeoutDebtorAgent: "Transaction stopped due to timeout at the Debtor Agent." No payment was taken for a different cause. A cancel at the terminal is one cause.
`DS02` Reason	OrderCancelled: "An authorized user has cancelled the order" The terminal stopped the payment before it got to its host, after a cancel from your register.
`CRDP` Transaction type	"Card payment." A sale.
`RFND` Transaction type	"Refund transaction." A refund.

FUNDiMO's own names

These names are FUNDiMO's, not ISO 20022's. The other names of FUNDiMO's API in this guide are FUNDiMO's too. `openapi.yaml`, `sandbox.md` and `conformance.json` define them.

Name	What it is
`RegisterIdentification` FUNDiMO's: field	The register's id. FUNDiMO issues it.
`IdempotencyKey` FUNDiMO's: field	Your register's idempotency key for a sale. A retry after a lost response cannot make a second sale.
`PaymentReference` FUNDiMO's: field	The terminal's references for the payment. They are in the group of the kind of payment: `Card`, `Bank`, `Wallet` or `NPP`.
`UNKN` FUNDiMO's: status	FUNDiMO does not know the outcome. No result came by the deadline, or the terminal contradicted itself.
`Mw-Register` FUNDiMO's: header	The register's header. It holds the register's `RegisterIdentification`.

Name	What it is
`Sale`, `Instruction`, `Outcome`, `StatusRequest`, `Abort`, `Error` FUNDiMO's: messages	The messages between your register, FUNDiMO and a terminal. `Abort` is FUNDiMO's cancel to a terminal.