

UNIPAY

Developer Guide

Integrating payment processors and service providers with UniPay API v1

Build e1eb5c49 · generated 2026-10-01

The online docs at [/docs/](#) on each UniPay environment are always current for that environment, and include the endpoint-by-endpoint API reference and code in eight languages. This PDF shows cURL.

Contents

Start here

1. [What UniPay is](#)
2. [Environments and access](#)
3. [Test data](#)
4. [Quick start](#)
5. [Authentication](#)

Concepts

1. [Bills and accounts](#)
2. [Currencies](#)
3. [Payment lifecycle](#)
4. [Idempotency and retries](#)
5. [Reversals](#)

Payment processors

1. [Payment processor guide](#)
2. [Look up a bill](#)
3. [Pay a bill](#)
4. [Day settlements](#)
5. [Service provider directory](#)
6. [Go-live checklist](#)

Service providers

1. [Service provider guide](#)
2. [Publish bills](#)
3. [Accounts and exchange rates](#)
4. [Collect payments](#)
5. [Downloads](#)
6. [Go-live checklist](#)

Reference

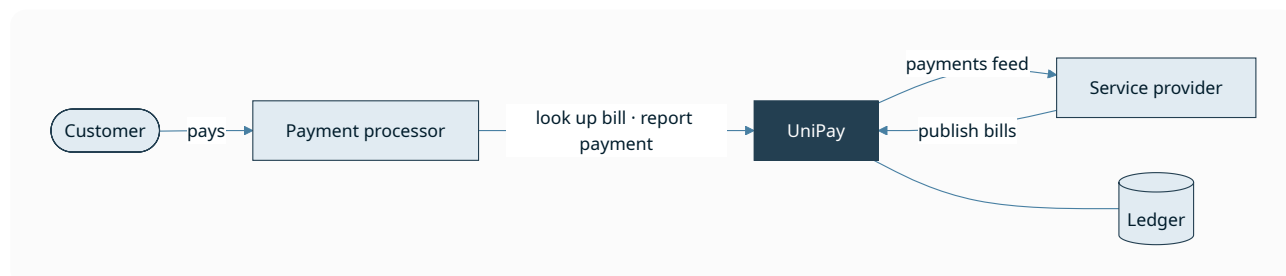
1. [Errors, limits and operations](#)
2. [Versioning and change policy](#)
3. [What's changed](#)
4. [Support](#)

What UniPay is

UniPay is a bill-payment gateway. It sits between the organisations that **raise bills** and the ones that **collect payment**, and gives each side one standardised connection instead of a bespoke integration per partner.

You are a...	Examples	You...
Service provider	Councils, utilities, statutory bodies	Publish bills and collect the payments made against them.
Payment processor	Banks, mobile-money operators, fintechs	Look up a customer's bill, take their money, and tell UniPay what was paid.

Both are **tenants** of UniPay: each has its own credentials and sees only its own data. A processor integrates once and can pay the bills of every service provider on the gateway.



How a payment flows

1. The service provider **publishes its bills** to UniPay — directly, or through the Axis Sync Engine beside its billing system.
2. A customer asks their bank or mobile-money operator to pay. The processor **looks the bill up** by bill number or by account, and shows the customer what they owe.
3. The processor takes the customer's money in its own system, then **reports the payment** to UniPay with its own unique reference.

4. UniPay records the payment in a **double-entry ledger** whose entries are hash-chained, so a recorded payment can't be silently altered.
5. The service provider **collects its payments** from UniPay's payments feed into its own system, and the money is settled to its bank account.

UniPay never converts money: it records exactly what the customer tendered, and what obligation that settles. See [Payment lifecycle](#) for the whole journey, and [Currencies](#) for payments in a different currency from the bill.

About these docs

These docs describe **the build of UniPay that serves them**: the test environment's docs describe the test environment, production's describe production. The build identifier is at the foot of every page — quote it when you contact [support](#).

Need them offline, or to attach to a tender? [Download the guides as one PDF](#) — generated from the same build, with code in cURL. The online docs add the endpoint-by-endpoint API reference and code in eight languages.

Next: [Environments and access](#).

Environments and access

Environments

Environment	Base URL	Use it for
Test	<code>https://test.unipay.co.zw</code>	Building and certifying your integration. Test data only — including a sandbox provider you can pay straight away.
Production	<code>https://online.unipay.co.zw</code>	Live payments, once you've completed the go-live checklist .

Each environment serves its own copy of these docs at `/docs/`. A change always reaches test before production; when the two differ, **the docs on the environment you are calling are the ones that apply**.

HTTPS only. Test and production are separate: credentials, organisations and data never cross between them.

Getting access

Axis Solutions onboards every partner. When your organisation is set up you receive:

- **API client credentials** — a `client_id` and `client_secret` your server uses to get access tokens. See [Authentication](#).
- **An administrator sign-in** for the UniPay portal, for managing your organisation, credentials and users. It isn't used by your integration code.
- A service provider also receives its permanent **provider number** and four-character **short code**, which processors use to find it.

Managing credentials

Your administrator can create further credentials in the portal under **API credentials**, each with its own `client_id` (cred-...) and only the scopes it needs. Common patterns:

- one credential per system or vendor that calls UniPay, so each can be revoked on its own;
- a new credential for each rotation, so you can roll over without downtime: create the new one, deploy it, then revoke the old one.

From the same page you can **regenerate a secret**, **disable** a credential or **revoke** it:

- A regenerated secret takes effect at once — the old secret stops working immediately. Tokens already issued stay valid until they expire (at most 30 minutes).
- A disabled or revoked credential can't get new tokens; the token endpoint answers `access_denied`.

Keep secrets in a secrets vault, never in source control or anything shipped to a device, and rotate them at least every 90 days.

Next: [Test data](#), then the [Quick start](#).

START HERE

Test data

The test environment (<https://test.unipay.co.zw>) has a **sandbox service provider** whose bills and accounts you can look up and pay straight away, with your own test credentials. It exists **only on the test environment** — production has no sandbox, and these numbers find nothing there.

Provider	UniPay Sandbox
Provider number	990000
Short code	SB0X
Accepts	USD, ZWG (default USD)
Exchange rate	1 USD = 25 ZWG, published every day

Use your own `client_id` and `client_secret`: the sandbox is a provider you pay, not an organisation you sign in as, and it has no credentials of its own. Nothing paid to it is real money, receipted or settled.

Bills and accounts

Look a bill up by its `billNo`, or by account with `"serviceProvider": "SB0X"` (or `"990000"`) and the `accountNumber`. `status` is explained in [Lookup status](#).

Bill	Account	Currency	Balance	Lookup returns
SBX0WEUSD	SBXA01	USD	1,000,000.00	status 1 Owing
SBX0WEZWG	SBXA02	ZWG	25,000,000.00	status 1 Owing

Bill	Account	Currency	Balance	Lookup returns
SBXDUALUSD	SBXA03	USD	500,000.00	status 1 Owing. By account otherCurrenciesAvailable is ["USD", "ZWG"]; with currency you get the USD
SBXDUALZWG	SBXA03	ZWG	12,500,000.00	status 1 Owing — the same account's ZWG bill; ask bill account with "currency": "ZWG".
SBXSETTLED	SBXA04	USD	0.00	status 2 NotOwing — Account is settled; nothing currently due account.
SBXCREDIT	SBXA05	USD	−75.50	status 4 InCredit — Account is in credit by account
SBXEXPIRED	SBXA06	USD	150.00	By bill: status 1 with an expiryDate in the past - check it before debiting. account: status 0, No outstanding bill found and no account on record for this number.
SBXCONVERT	SBXA07	USD	1,000,000.00	status 1 Owing. Ask with "currency": "ZWG" and response adds conversionApplied: true

Bill	Account	Currency	Balance	Lookup returns
				rate: 25 and convertedRemainingBa
—	SBXA08	USD	—	By account: successful false, status 3 Unknow Account standing is stale; no current information for this number.

A few more outcomes you can produce with the same data:

- **Replay:** pay again with the same rrn, bill, amount and currency and you get `successful: true, isNewRecord: false, Duplicate notification; already processed`. The same rrn with a different amount is refused: `RRN '...' was already used for a different payment`. See [Idempotency and retries](#).
- **A currency the provider doesn't take:** paying in EUR is refused with `Currency 'EUR' is not accepted by this service provider`. Accepted currencies: USD, ZWG.
- **A bill that doesn't exist:** any other SBX... number returns `Bill 'SBX...' not found`.

Balances move, then reset

The sandbox is shared by every partner on the test environment, so what you see includes everyone's payments:

- A payment reduces the bill's balance for the rest of the **business day** (Harare time) — look the bill up again and `totalPaid` and `remainingBalance` reflect it.
- At the start of the next business day each bill is reset to the balance in the table. Its `totalAmount` rises by what was paid before that day so that

remainingBalance is back where it started; the payments themselves stay on record.

- The owing bills are large so that a day's testing does not pay them off.

So write your tests against status, message and the field shapes, not against an exact balance.

Quick start

This round-trips one payment on the test environment: get a token, look up a bill, pay it. It's the whole payment processor integration in one program. Service providers: see [Publish bills](#) and [Collect payments](#) instead.

1 Get test credentials from your Axis Solutions contact: a `client_id` and `client_secret` for `https://test.unipay.co.zw`. For a bill to pay, use one from the sandbox in [Test data](#) — SBX0WEUSD always owes.

2 Set them in your environment:

```
export UNIPAY_BASE_URL=https://test.unipay.co.zw
export UNIPAY_CLIENT_ID=your-client-id
export UNIPAY_CLIENT_SECRET=your-client-secret
```

3 Run the example for your language with the bill number and an amount. It prints what the bill's holder owes, pays, and prints the result:

```
Sandbox Owing USD owes 999875.0000 USD (Sandbox: owing in USD)
EX1790704100495: Payment Saved (new record: True)
```

The program

Each tab is a complete program; its first lines say how to run it. The same code is split step by step through the rest of the docs.

```
#!/usr/bin/env bash
# UniPay payment processor example: get a token, look up a bill, pay it.
# bash + curl + jq.
#
#   export UNIPAY_BASE_URL=https://test.unipay.co.zw
#   export UNIPAY_CLIENT_ID=... UNIPAY_CLIENT_SECRET=...
#   ./payment_processor.sh <billNo> <amount>
set -euo pipefail
BASE_URL="${UNIPAY_BASE_URL:-https://test.unipay.co.zw}"
BILL_NO="$1"
AMOUNT="$2"

# Tokens last 30 minutes: reuse one for every call in that window.
TOKEN=$(curl -sS --fail-with-body -X POST "$BASE_URL/connect/token" \
  --data-urlencode "grant_type=client_credentials" \
  --data-urlencode "client_id=$UNIPAY_CLIENT_ID" \
  --data-urlencode "client_secret=$UNIPAY_CLIENT_SECRET" \
  --data-urlencode "scope=unipay:bills" | jq -r .access_token)

LOOKUP=$(curl -sS --fail-with-body -X POST "$BASE_URL/api/app/payment-processor/lookup" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"billNo": "$BILL_NO"}')

echo "$LOOKUP" | jq '{successful, message, name: .bill.name, currency:
.bill.currency, payableNow, acceptedCurrencies}'

CURRENCY=$(echo "$LOOKUP" | jq -r .bill.currency)
RRN="EX$(date +%s%3N)" # your unique reference: reuse it on every
retry of this payment

curl -sS --fail-with-body -X POST "$BASE_URL/api/app/payment-processor/payment-details" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "billNo": "$BILL_NO",
    "rrn": "$RRN",
  }'
```

```
\ "paymentDate\ ": \"$(date +%Y-%m-%dT%H:%M:%S%:z) \",
\ "currency\ ": \" $CURRENCY \",
\ "amount\ ": $AMOUNT
} "
# → { "successful": true, "message": "Payment Saved", "isNewRecord": true, ... }
```

What it does

- **Caches the access token** and renews it a minute before it expires — see [Authentication](#).
- **Looks the bill up** with [POST /lookup](#) and reads what's owed — see [Look up a bill](#).
- **Reports the payment** with [POST /payment-details](#) under a reference (rrn) it generates, and **retries with the same reference** if the outcome is unknown — see [Pay a bill](#) and [Idempotency and retries](#).

These examples are run against a UniPay instance before each docs release.

Authentication

UniPay uses OAuth 2.0 **client credentials** — machine to machine, with no user involved. Your server exchanges its `client_id` and `client_secret` for an access token, then sends the token on every API call.

Get a token

```
POST /connect/token
```

```
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=client_credentials&client_id=YOUR_CLIENT_ID&client_secret=YOUR_CLIENT_SECRET&scope=unipay:bills
```

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "token_type": "Bearer",
  "expires_in": 1799
}
```

- Send the credentials in the form body, as above, **or** as an Authorization: Basic `base64(client_id:client_secret)` header. Either works; pick one.
- URL-encode the values — a secret containing `+`, `&` or `#` breaks otherwise.
- Tokens last **30 minutes**. No refresh token is issued or needed: ask for a new token instead.
- Every environment publishes its OAuth settings at `/.well-known/openid-configuration`.

Then send the token on every call:

```
POST /api/app/payment-processor/lookup
```

```
Authorization: Bearer eyJhbGciOiJSUzI1NiIs...
```

Your organisation is identified from the token. There is no tenant header to send.

Cache the token

Request a token once and reuse it until shortly before it expires. Requesting one per call is slow, and the token endpoint is rate-limited to 10 requests a minute.

- Renew when about **60 seconds** remain, so requests already in flight don't carry an expired token.
- Hold **one token per process** and guard the renewal with a lock, so concurrent requests don't all renew at once.
- If a call gets **401**, drop the cached token, get a new one and **retry once**. A second 401 means the credentials are wrong or revoked — don't loop.
- Keep the token in memory, never on disk, and never log it.

```
# Tokens last 30 minutes: reuse one for every call in that window.
TOKEN=$(curl -sS --fail-with-body -X POST "$BASE_URL/connect/token" \
  --data-urlencode "grant_type=client_credentials" \
  --data-urlencode "client_id=$UNIPAY_CLIENT_ID" \
  --data-urlencode "client_secret=$UNIPAY_CLIENT_SECRET" \
  --data-urlencode "scope=unipay:bills" | jq -r .access_token)
```

Scopes

Ask only for the scopes your integration needs, as a space-separated list in scope.

Scope	Grants	Who needs it
unipay:bills	Processors: look up and pay bills. Providers: publish bills.	Everyone

Scope	Grants	Who needs it
<code>unipay:payments</code>	Read the payments and reversals made against your bills. Needs <code>unipay:bills</code> too.	Service providers
<code>unipay:directory</code>	List the service providers you can pay.	Payment processors, optional
<code>unipay:directory.settlement</code>	Adds each provider's settlement accounts to the directory, account numbers masked.	Processors that settle directly
<code>unipay:directory.settlement.reveal</code>	Reveals a settlement account's full number. Every reveal is audited.	Only processors that initiate transfers

A call without the scope it needs gets **403 Forbidden** with an empty body; the WWW-Authenticate response header says `error="insufficient_access"`.

Credentials only work on their own side of UniPay: payment-processor credentials can't call service-provider endpoints, and the reverse. Such a call gets **403** with an error object whose code is `Authorization:PartnerTypeMismatch`.

When the token request fails

The token endpoint answers 401 (bad credentials) or 400 (anything else) with a standard OAuth error body — `error`, `error_description` and `error_uri`:

error	Meaning
invalid_client	Unknown client_id, or wrong client_secret. Check for encoding problems in the secret.
invalid_scope	A scope name UniPay doesn't recognise.
invalid_request	A real scope your credential isn't allowed. Ask for it to be added, or drop it.
access_denied	The credential has been disabled or revoked.
unsupported_grant_type	grant_type isn't client_credentials.

Security checklist

1. Keep client_secret in a secrets vault. Never commit it or ship it to a phone, browser or POS device.
2. Use separate credentials for test and production, and for each system that calls UniPay.
3. Rotate secrets at least every 90 days — see [Managing credentials](#).
4. Treat access tokens like passwords: keep them in memory, never log them or echo them in errors.
5. Call UniPay only over HTTPS, and don't disable certificate validation.

Bills and accounts

A **bill** is an amount a service provider says a customer owes. Customers also have an **account** at their provider, and one account has a bill per billing period — and sometimes one per currency.

Two identifiers on every bill

Field	What it is	Unique	What to do with it
billNo	The UniPay bill number, usually the provider number plus its own invoice number — e.g. 4444INV0009. Letters and digits, 4–50 characters.	Within the service provider (one per currency)	Payment processors: send it back when you pay. Treat it as opaque: don't parse, split or reformat it.
uniqueNo	The service provider's own internal reference: invoice ID, statement number, case reference.	No	Display it if it helps the customer. Never send it on a payment.

Customers type bill numbers with spaces, dashes and dots; lookups ignore those, so 4444 - INV - 0009 finds 4444INV0009.

Finding what's owed

A payment processor can find a bill two ways:

Look up by	When	Needs
Bill number	The customer has a bill in hand.	billNo
Account	The customer knows their account but not a bill — “pay my rates account”.	serviceProvider (its number or short code) and accountNumber

Account numbers are only unique within a provider, so an account lookup always names the provider. An account lookup returns the account’s **current** bill: the newest bill still owing — so settling this month’s statement while last month’s is unpaid doesn’t report “nothing due” — or, if nothing is owing, its newest bill.

Lookup status

Lookups return a numeric status:

Value	Status	Meaning	Take payment?
1	Owing	The account owes money. bill holds the bill to pay.	Yes
2	NotOwing	Known account, nothing due. bill holds its latest bill.	Only if you allow paying in advance
4	InCredit	Known account, in credit.	Only if you allow paying in advance
3	Unknown	The account exists but its balance is out of date, or it’s closed.	No
0	NotFound	No bill and no account for this number.	No

New statuses may be added. Treat any value you don't recognise as "don't take payment".

What is owed now

A bill's total can differ from what the account owes today — part may already be paid, through UniPay or at the provider's own counter. The lookup response carries:

Field	Meaning
<code>bill.totalAmount</code>	The bill as issued.
<code>bill.totalPaid</code>	Paid against it through UniPay so far.
<code>bill.remainingBalance</code>	<code>totalAmount - totalPaid</code> .
<code>amountDueNow</code>	What the account owes right now, including payments made outside UniPay when the provider <u>shares its balances</u> . Negative when in credit.
<code>payableNow</code>	<code>amountDueNow</code> floored at zero: the most to ask the customer for .
<code>balanceIsStale</code>	<code>true</code> when that balance may be out of date.

Show the customer payableNow, not `totalAmount`. Look the bill up again immediately before taking payment — don't reuse a lookup from an earlier session, because other channels may have collected in the meantime.

Privacy

When you look a bill up **by bill number**, the account number comes back masked — every character but the last four replaced with * (123456789 → *****6789). You don't need the account number to pay; `billNo` identifies the bill.

Currencies

Every bill has a **native currency** — the currency the service provider raised it in. An account can have bills in more than one currency.

What the customer can pay in

Each service provider chooses the currencies it accepts; a lookup lists them in `acceptedCurrencies`. A payment in any other currency is **refused**:

```
{ "successful": false, "message": "Currency 'EUR' is not accepted by  
this service provider. Accepted currencies: USD, ZWG" }
```

If a lookup response has no `acceptedCurrencies`, pay in the bill's own currency.

Asking for a currency

Every lookup takes an optional currency (a three-letter ISO 4217 code such as USD or ZWG):

- If the account has a bill in that currency, you get that bill.
- Otherwise you get the provider's default bill, and — when the provider publishes an exchange rate — its amounts converted into the currency you asked for: `conversionApplied: true`, with `convertedTotalAmount`, `convertedRemainingBalance`, `rate` and `rateDate`. `conversionIsIndicative: true` means the figure is a guide, not a quote.
- `otherCurrenciesAvailable` lists the other currencies the account has bills in.

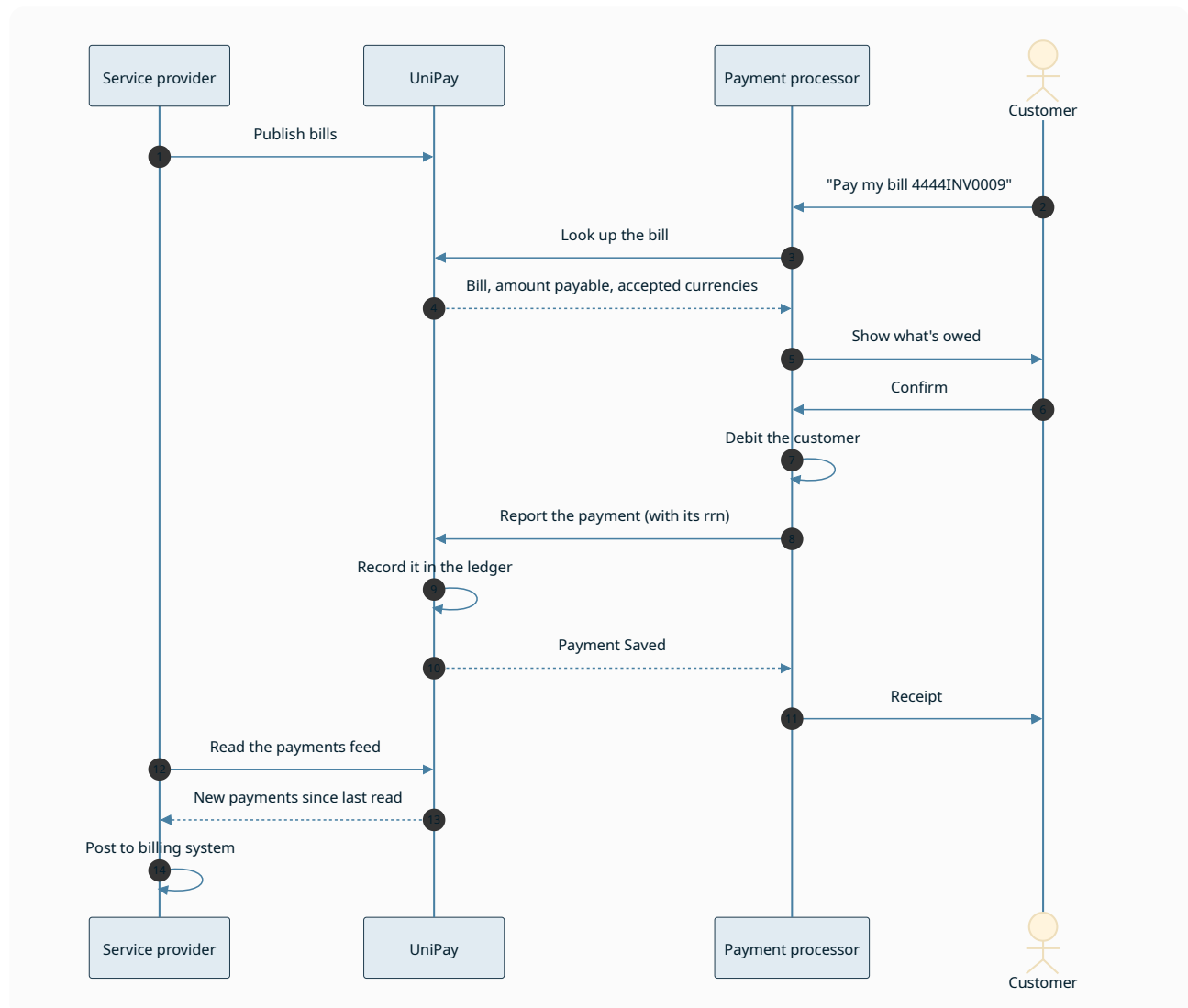
Paying in a different currency

UniPay **never converts money**. It records exactly what the customer tendered, and separately what obligation that payment settles:

1. **The account has a bill in the payment currency:** the payment settles that bill directly.
2. **It doesn't, the provider bills in a single currency, and it has an exchange rate for the payment date:** the payment settles the bill, converted at the provider's own rate. The rate is recorded with the payment.
3. **Otherwise:** the payment is still accepted and recorded, but not against a bill — money that has already left the customer is never turned away. The service provider resolves it through its own exceptions process.

The service provider sees both figures on every payment: what was **tendered** (amount, currency) and what it **settled** (allocatedAmount, allocatedCurrency, allocationRate) — see [Collect payments](#).

Payment lifecycle



Steps	Who	What happens	Guide
1	Service provider	Publishes bills, and republishes them as they change.	Publish bills
2-5	Payment processor	Looks the bill up and shows the customer payableNow.	Look up a bill
6-7	Payment processor	Takes the customer's money in its own systems. UniPay isn't involved.	

Steps	Who	What happens	Guide
8–10	Payment processor	Reports the payment. UniPay checks it, records it, and answers.	Pay a bill
11	Payment processor	Gives the customer their receipt — only after UniPay has confirmed.	
12–14	Service provider	Pulls new payments and posts them to its billing system.	Collect payments

Things that are easy to get wrong

- **Report a payment only after the customer has been debited**, and give the customer a receipt only after UniPay confirms it.
- **The order is fixed: debit, then report.** If you can't reach UniPay after debiting, keep retrying with the same rrrn — see [Idempotency and retries](#). Don't reverse the customer's debit because a report timed out.
- **UniPay doesn't push.** Service providers read the payments feed on their own schedule; nothing calls them back.
- **Money moves separately.** Recording a payment in UniPay is not a bank transfer; settlement to the service provider's bank account follows. A processor that pays providers directly reports each transfer as a [day settlement](#).
- **A recorded payment is never edited.** If one shouldn't stand, it is [reversed](#).

Idempotency and retries

Networks fail. A request can time out after UniPay has already recorded the payment, and your side can't tell. UniPay is built so that **retrying is always safe** — provided you retry the right way.

The payment reference (rrn)

Every payment you report carries an rrn (retrieval reference number): **your** unique reference for that payment, up to 50 characters. UniPay uses it to recognise a repeat:

You send	UniPay answers
An rrn it hasn't recorded	Records the payment. "successful": true, "isNewRecord": true
An rrn it has already recorded, for the same payment	Records nothing new. "successful": true, "isNewRecord": false
An rrn it has already recorded, for a different bill, account, amount or currency	Refuses it: "successful": false, RRN 'X' was already used for a different payment.
An rrn it refused before (e.g. the date was wrong)	Processes it afresh — a corrected retry isn't blocked by the earlier refusal.

References are **per processor**: another processor using the same rrn for its own payment doesn't affect yours.

So:

- **Generate the rrn once per payment**, before the first attempt, and store it with the transaction.
- **Reuse it on every retry** of that payment — after a timeout, a dropped connection, a 429 or a 5xx.

- **Treat `isNewRecord: false` as success.** It means UniPay already had the payment.
- **Never reuse an `rrn` for a different payment.** UniPay refuses it. Make references unique across your whole organisation — prefixing your own transaction ID works well.

What to retry

Outcome	Retry?
Timeout, connection reset, no response	Yes , same request, same <code>rrn</code> . You don't know whether it arrived.
429 Too Many Requests	Yes , after a pause.
500, 502, 503, 504	Yes , with backoff.
401 Unauthorized	Get a new token, retry once .
200 with <code>"successful": false</code>	No . UniPay understood and refused; the same request gets the same answer. Fix the cause.
400, 403	No . Fix the request or your credentials.

Back off exponentially with some randomness — for example 2 s, 4 s, 8 s, 16 s, 30 s — and stop after about five attempts. If a payment still isn't confirmed, park it for investigation rather than dropping it: the customer has paid.

Timeouts

Use a **10-second connect timeout** and a **30-second read timeout**. UniPay normally answers in well under a second; anything above five seconds is worth investigating.

Service providers

Publishing a bill is also safe to repeat: sending a bill UniPay already has, unchanged, returns `AlreadyExists` and changes nothing. Reading the payments feed is safe to repeat too, because you choose where to resume — see [Collect payments](#).

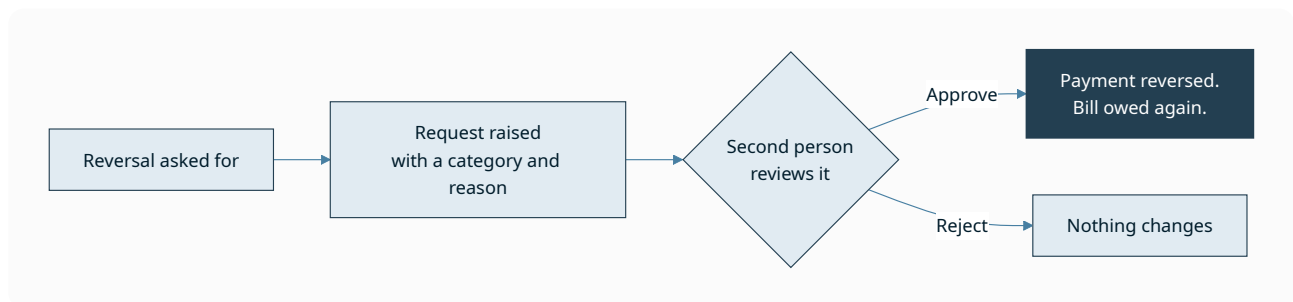
Reversals

A payment UniPay has recorded is never deleted or edited. If it shouldn't stand — the customer's bank recalled it, a chargeback was upheld, it was keyed against the wrong account — it is **reversed**: the payment stays on record, marked reversed, and the bill it paid is owed again.

A reversal is a correction to the record. UniPay doesn't move money, so any refund to the customer is between you and them.

Who can reverse a payment

Only Axis Solutions operations, and never one person alone:



1. Someone at Axis raises a **request** for the payment, with a category and a reason. A request is Pending; nothing has changed yet.
2. A **different** person approves or rejects it. Approval reverses the payment at once. Rejection leaves it as it was.

There's no API for partners to reverse a payment or raise a request. To ask for one, contact your Axis Solutions contact, as for support, with the payment's rrrn, its billNo, the amount, and why it should be reversed. A payment is reversed once; a payment already reversed, or with a request pending, can't be asked for again.

Each reversal carries one of these categories:

category	Used for
Dispute	A customer dispute or a chargeback.
Fraud	A fraudulent payment.
BankCorrection	A correction by a bank.
PayoutRejected	A transfer that was rejected.
OperatorError	A payment keyed wrongly.
SystemCorrection	A correction to a system error.

What the payment processor sees

- **The bill is owed again.** A lookup shows the reversed amount back in `remainingBalance`, and `totalPaid` no longer counts it.
- **The `rrn` stays used.** Reporting a payment again with the same `rrn` returns `isNewRecord: false` and records nothing — the reversed payment still holds it. If the customer pays again, that's a new payment with a new `rrn`.
- **Day settlements.** A reversed payment is owed nothing: leave it out of your transfer. If you include it, its line comes back `Reversed` — see [Day settlements](#).
- Your administrators see reversals of your payments in the UniPay portal report **Payments, reversals & fees**.

What the service provider sees

- The payment you collected **doesn't change** in the payments feed.
- The reversal arrives in the **reversals feed** once it is approved, with the `paymentId` it reverses, its category and reason. Pending and rejected requests never appear. Read it with its own checkpoint and reverse the payment in your own system — see [Collect payments](#).

Payment processor guide

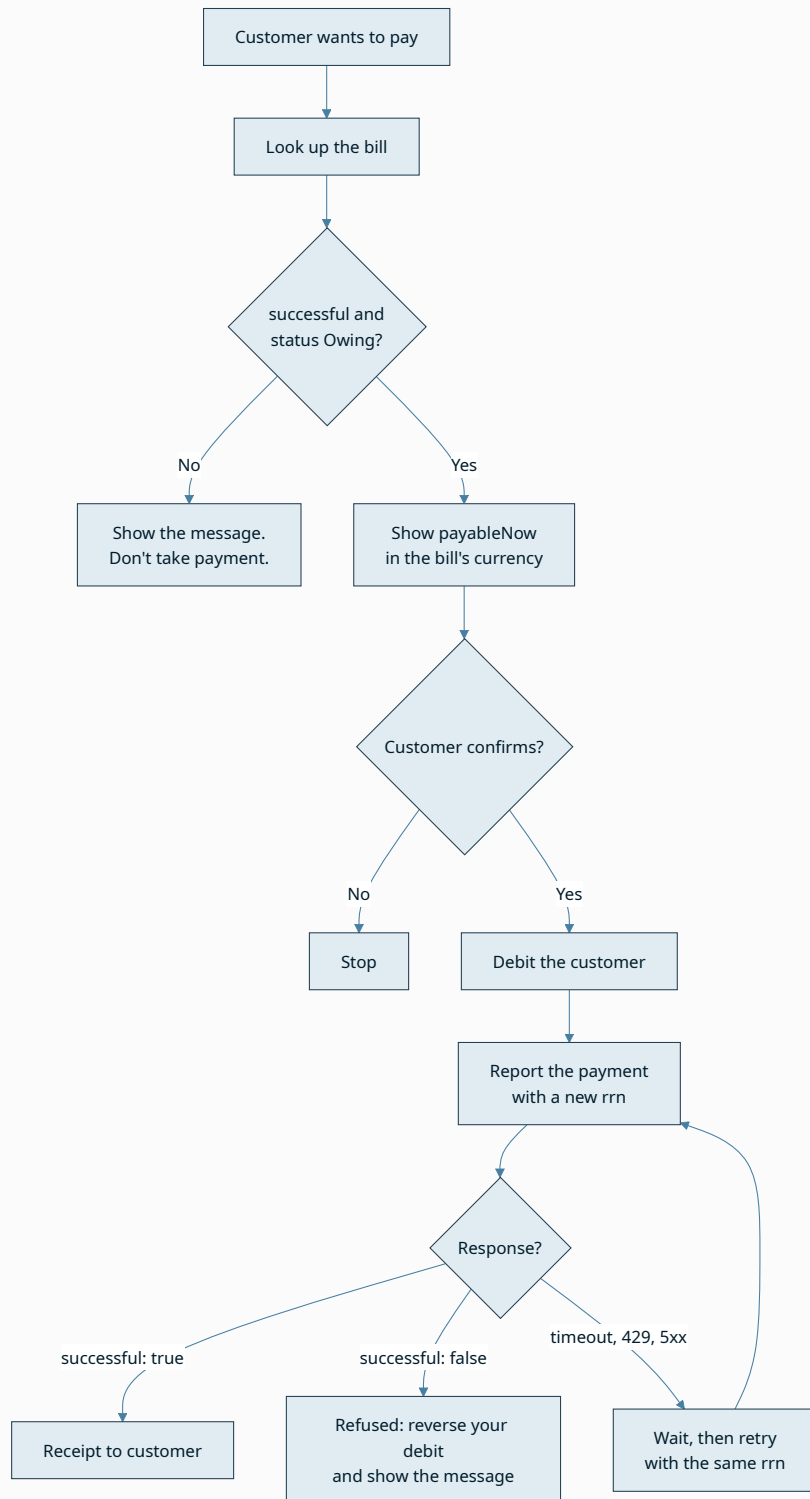
As a payment processor you take payment from a customer and tell UniPay which bill it pays. One integration covers every service provider connected to UniPay. Your part of the [payment lifecycle](#):

1. **Get a token** with your client credentials — [Authentication](#).
2. **Find the bill** by bill number or by account, and show the customer what they owe — [Look up a bill](#).
3. **Take the customer's payment** in your own system.
4. **Report the payment** to UniPay with your unique reference for it — [Pay a bill](#).

If you pay service providers directly, also **report each day's transfer** so it can be reconciled — [Day settlements](#). Optionally, list the service providers customers can pay — [Service provider directory](#). To see all of it working at once, start with the [Quick start](#).

A payment you've reported can't be withdrawn through the API; if one shouldn't stand, Axis can [reverse it](#).

Handling one payment



A refusal (`successful: false`) means UniPay recorded nothing — reverse the customer's debit on your side. A timeout or server error means the outcome is **unknown**: never reverse the debit then; retry with the same rrn until you get a definite answer. See [Idempotency and retries](#).

What you can call

Everything lives under `/api/app/payment-processor/` and `/api/app/directory/`:

Endpoint	Scope	Reference
POST <code>/api/app/payment-processor/lookup</code>	<code>unipay:bills</code>	lookup
POST <code>/api/app/payment-processor/payment-details</code>	<code>unipay:bills</code>	pay
POST <code>/api/app/payment-processor/settle-day</code>	<code>unipay:bills</code>	settleDay
GET <code>/api/app/payment-processor/{id}/day-settlement</code>	<code>unipay:bills</code>	getDaySettlement
GET <code>/api/app/payment-processor/day-settlements</code>	<code>unipay:bills</code>	listDaySettlements
GET <code>/api/app/directory/service-providers</code>	<code>unipay:directory</code>	listServiceProviders
GET <code>/api/app/tenant-config</code>	<code>any</code>	tenantConfig

GET `.../bill` and GET `.../latest-bill-by-account` still work for existing integrations; new integrations should use `lookup`, which does both. The [Payment Processor API reference](#) lists every endpoint and field.

Look up a bill

One endpoint does both kinds of lookup — `POST /api/app/payment-processor/lookup`, scope `unipay:bills`.

```
LOOKUP=$(curl -sS --fail-with-body -X POST
"$BASE_URL/api/app/payment-processor/lookup" \
-H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
-d "{\"billNo\": \"$BILL_NO\"}")

echo "$LOOKUP" | jq '{successful, message, name: .bill.name, currency:
.bill.currency, payableNow, acceptedCurrencies}'
```

Request

Send **exactly one** of: a `billNo`, or a `serviceProvider` with an `accountNumber`.

```
{ "billNo": "4444INV0009" }
```

```
{ "serviceProvider": "MUT1", "accountNumber": "100-2345",
"currency": "USD" }
```

Field	Notes
<code>billNo</code>	The bill number from the customer's bill. Spaces, dashes and dots are ignored.
<code>serviceProvider</code>	The provider's number (4444) or its short code (MUT1). Find them in the directory .

Field	Notes
accountNumber	The customer's account number at that provider. Separators don't need to match. Up to 64 characters.
currency	Optional: the currency the customer wants to pay in. See Currencies .

Response

```
{
  "successful": true,
  "message": "Bill found successfully",
  "status": 1,
  "bill": {
    "billNo": "4444INV0009",
    "uniqueNo": "INV0009",
    "serviceProviderNumber": "4444",
    "serviceProviderName": "City of Mutare",
    "accountNumber": "*****2345",
    "name": "J. Moyo",
    "description": "Rates - September 2026",
    "billDate": "2026-09-01T00:00:00Z",
    "dueDate": "2026-09-30T00:00:00Z",
    "expiryDate": null,
    "currency": "USD",
    "totalAmount": 220.00,
    "totalPaid": 50.00,
    "remainingBalance": 170.00
  },
  "amountDueNow": 170.00,
  "payableNow": 170.00,
  "balanceIsStale": false,
  "acceptedCurrencies": ["USD", "ZWG"],
  "otherCurrenciesAvailable": null,
  "conversionApplied": false
}
```

Check successful first, then status — see [Lookup status](#). Take payment only when status is 1 (Owing), and:

- ask the customer for at most **payableNow**, in the bill's **currency** or one of **acceptedCurrencies**;
- pay with the **billNo** from this response;
- refuse before debiting if **expiryDate** has passed — the payment would be refused.

The fields are explained in [Bills and accounts](#), and every field is listed in the [reference](#). The bill also carries `serviceProvider`, `accountNo` and `amount`: older names for `serviceProviderNumber`, `accountNumber` and `totalAmount`, kept for existing integrations.

When nothing is found

A lookup that can't find a bill still answers **HTTP 200**, with `successful: false` and a `message`:

message	Cause
Bill '4444INV0009' not found	No bill with that number.
No outstanding bill found and no account on record for this number	Account lookup: no bill and no account (status 0).
Account standing is stale; no current information for this number	Account lookup: account known, balance out of date (status 3).
Account is settled; nothing currently due	Account lookup: nothing owed (status 2) — <code>successful</code> is true.
Account is in credit	Account lookup: in credit (status 4) — <code>successful</code> is true.

message	Cause
Service provider 'X' not found	Unknown provider number or short code.
Service provider short code 'X' is ambiguous; use the provider number	Use the provider number instead.
Provide either a bill number or a service provider and account number, not both	Send one or the other.
Provide a bill number, or a service provider and account number	Neither was sent.
Bill number must be at least 4 characters / cannot exceed 50 characters / can only contain letters and numbers	Malformed bill number.
Currency must be a 3-letter ISO 4217 code	Malformed currency.
Show the message to your operator or log it; don't match on its exact wording, which may be improved over time. The full list is in Errors and messages .	

Pay a bill

Once you have taken the customer's money, report it with `POST /api/app/payment-processor/payment-details`, scope `unipay:bills`. The examples retry safely until the outcome is known.

```
CURRENCY=$(echo "$LOOKUP" | jq -r .bill.currency)
RRN="EX$(date +%s%3N)" # your unique reference: reuse it on every
retry of this payment

curl -sS --fail-with-body -X POST "$BASE_URL/api/app/payment-
processor/payment-details" \
-H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
-d "{
  \"billNo\": \"$BILL_NO\",
  \"rrn\": \"$RRN\",
  \"paymentDate\": \"$(date +%Y-%m-%dT%H:%M:%S%:z)\",
  \"currency\": \"$CURRENCY\",
  \"amount\": $AMOUNT
}"
# → {"successful":true,"message":"Payment Saved","isNewRecord":true,...}
```

Request

```
{
  "billNo": "4444INV0009",
  "rrn": "PX20260929000123",
  "paymentDate": "2026-09-29T10:15:00+02:00",
  "currency": "USD",
  "amount": 170.00
}
```

To pay against an **account** instead, replace `billNo` with `serviceProvider` and `accountNumber`, as in the [lookup](#). The payment lands on the same bill the lookup returned.

Field	Required	Rules
<code>billNo</code>	Yes, or <code>accountNumber</code>	The bill being paid, from the lookup. If you send both, the bill number wins.
<code>serviceProvider</code>	With <code>accountNumber</code>	Provider number or short code.
<code>accountNumber</code>	Or <code>billNo</code>	Up to 50 characters.
<code>rrn</code>	Yes	Your unique reference for this payment, up to 50 characters. See Idempotency .
<code>paymentDate</code>	Yes	When the customer paid. Include a UTC offset (+02:00 or Z) — a time without one is read as UTC. Not in the future and not more than 90 days ago, judged by the calendar day in Zimbabwe.
<code>currency</code>	Yes	Three-letter ISO 4217 code: the bill's currency or one of its <code>acceptedCurrencies</code> .
<code>amount</code>	Yes	What the customer paid, greater than zero.

Paying more than is owed, and paying after the due date, are both accepted — unless the service provider has chosen to refuse overpayments.

Response

```
{ "successful": true, "message": "Payment Saved", "isNewRecord": true }
```

"isNewRecord": false with successful: true means UniPay already had this rrn — the payment is recorded; treat it as success. The message is then Duplicate notification; already processed.

Refusals

A payment UniPay can't accept answers **HTTP 200** with "successful": false.

Nothing is recorded: reverse the customer's debit on your side and show the reason.

message	Cause
Bill '4444INV0009' not found. Cannot process payment.	No bill with that number.
No bill found for that account. Cannot process payment.	Account payment: the account has no bill.
Currency 'EUR' is not accepted by this service provider. Accepted currencies: USD, ZWG	Pay in an accepted currency.
Invoice has expired	The bill's expiryDate has passed.
Payment would exceed the remaining balance on this invoice	Overpayment, where the provider refuses them.
Payment date cannot be in the future	Check your clock and the offset on paymentDate.
Payment date cannot be more than 90 days in the past	Too late to report through the API; contact support.
Service provider is required when paying by account number	Add serviceProvider.
RRN 'X' was already used for a different payment	You sent an rrn you'd already used, for a different bill, account, amount or

message	Cause
	currency. Give each payment its own r rn.
Retrieval Reference Number is required · RRN must not exceed 50 characters	Fix r rn.
Payment amount must be greater than zero · Currency must be 3 characters · Payment date is required	Fix the field.

A request the API can't parse at all gets **HTTP 400** — see [Errors and messages](#).

Paying exactly once

This is the part to get right. If a call times out, or you get a 429 or 5xx, **you don't know whether the payment was recorded**. Send the same request again with the **same r rn**: if it was recorded you get `isNewRecord: false`, and nothing is recorded twice.

An r rn is yours alone: another processor using the same value doesn't affect you. Never reuse one of your own for a different payment — UniPay refuses it — and never reverse a customer's debit because a report timed out. The whole policy is in [Idempotency and retries](#).

Day settlements

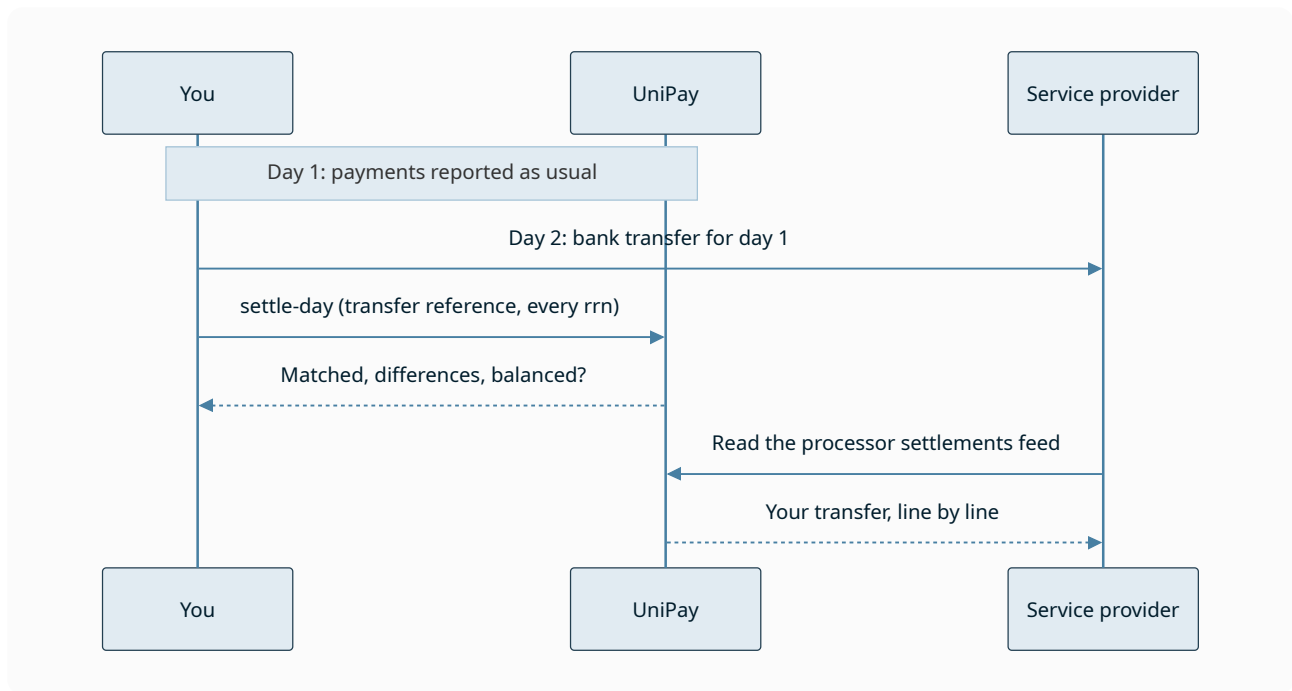
If you pay a service provider directly — one bank transfer covering a day's payments — tell UniPay about each transfer. UniPay matches every payment you list against the payments you reported, finds the ones you left out, and tells you at once whether the transfer balances. The service provider sees the same report when it reconciles its bank statement.

Send one day settlement per transfer with [POST /api/app/payment-processor/settle-day](#), scope `unipay:bills`.

When to send it

Send it when you make the transfer — usually the business day after the payments were taken.

- **settlementDate** is the day of the transfer. UniPay reconciles it against your payments dated **the day before**, in your business timezone (normally Africa/Harare).
- **One settlement is one transfer, to one service provider, in one currency.** If you pay a provider in two currencies, or pay two providers, send one settlement for each transfer.
- List **every payment the transfer covers**, by its `rrn`.



Request

```

{
  "settlementDate": "2026-09-30",
  "transferReference": "RTGS20260930-0042",
  "billerCode": "4444",
  "creditedAccountNumber": "1234567890",
  "currency": "USD",
  "totalAmount": 368.50,
  "feeAmount": 1.50,
  "lines": [
    { "rrn": "PX20260929000123", "billNo": "4444INV0009", "amount": 170.00 },
    { "rrn": "PX20260929000124", "amount": 200.00 }
  ]
}

```



```
curl -X POST https://test.unipay.co.zw/api/app/payment-processor/settle-day \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d @settlement.json
```

Field	Required	Rules
settlementDate	Yes	The day of the transfer, yyyy-MM-dd. Not in the future and not more than 90 days ago. A date-time is accepted, but only its date is used, exactly as written — the time and offset are ignored.
transferReference	Yes	Your bank or RTGS reference for the transfer, up to 100 characters. It makes the request <u>safe to retry</u> .
billerCode	Recommended	The service provider paid: its number or short code, up to 64 characters. Every rrn must then be one of its payments.
creditedAccountNumber	Recommended	The provider's bank account you credited, up to 64 characters. Providers bank with several banks and can't otherwise tell which account received the money.
currency	Yes	Three-letter ISO 4217 code of the transfer.
totalAmount	Yes	What you transferred: the sum of the lines less feeAmount. Greater than zero.

Field	Required	Rules
feeAmount		Commission you kept from the transfer, if any. Default 0. Must be less than the sum of the lines.
lines	Yes	1 to 5,000 lines, one per payment.
lines[].rrn	Yes	The payment's rrn, as you reported it. Up to 50 characters, each at most once.
lines[].amount	Yes	The amount of that payment. Greater than zero.
lines[].billNo		The bill paid, up to 100 characters. When you send it, it must be the bill UniPay recorded; spaces, dashes and case don't matter.

Without `billerCode`, UniPay takes the provider from the payments you list. Send it anyway: if none of the `rrns` is recognised, a settlement with no provider isn't shown to anyone but you. An account number UniPay doesn't recognise is recorded as you sent it and flagged to Axis for checking; it doesn't refuse the settlement, because the money has already moved.

Response

"successful": `true` means UniPay **recorded and reconciled** the settlement — not that it balanced. Check `isBalanced`.

```
{
  "successful": true,
  "message": "Settlement accepted with 2 difference(s).",
  "isNewRecord": true,
  "settlementId": "3a24048c-e7c8-c3bc-e816-72a95370cb3b",
  "settlementDate": "2026-09-30",
  "transferReference": "RTGS20260930-0042",
  "currency": "USD",
  "reconDate": "2026-09-29T00:00:00Z",
  "reconTimeZone": "Africa/Harare",
  "reconWindowStartUtc": "2026-09-28T22:00:00Z",
  "reconWindowEndUtc": "2026-09-29T22:00:00Z",
  "reportedTotal": 370.00,
  "feeAmount": 1.50,
  "transferredAmount": 368.50,
  "matchedTotal": 170.00,
  "reportedCount": 2,
  "matchedCount": 1,
  "unmatchedByUnipay": [
    { "rrn": "PX20260929000124", "billNo": null, "amount": 200.00,
"unipayAmount": 150.00, "reason": "AmountMismatch" }
  ],
  "unsettledByProcessor": [
    { "rrn": "PX20260929000131", "billNo": "4444INV0011", "amount":
40.00, "unipayAmount": 40.00, "reason": "NotSettledByProcessor" }
  ],
  "difference": 10.00,
  "isBalanced": false
}
```

Field	Meaning
settlementId	UniPay's ID for the settlement. Store it to read it back .
reconDate	The day whose payments were reconciled — the day before settlementDate. Only the date part means anything.

Field	Meaning
reconTimezone, reconWindowStartUtc, reconWindowEndUtc	That day as a UTC window: payments with a paymentDate inside it were expected.
reportedTotal, reportedCount	The sum and count of your lines, before commission.
feeAmount, transferredAmount	As you sent them. transferredAmount is your totalAmount.
matchedTotal, matchedCount	The lines that matched a payment exactly.
unmatchedByUnipay	Your lines that didn't match, each with a reason. amount is yours; unipayAmount is what UniPay recorded, when it found the payment.
unsettledByProcessor	Payments you reported for that day, to that provider and in that currency, that the settlement left out.
difference	reportedTotal less what UniPay holds as owed — its own amount for every payment you listed that it counts, plus the ones you left out. Negative: you sent less.
isBalanced	true only when every line matched and nothing was left out. Differences can cancel out, so a zero difference alone doesn't mean balanced.

The message is Settlement balanced – all payments matched. when it balances, otherwise Settlement accepted with N difference(s).

Why a line doesn't match

reason	Meaning	Counted as owed?
UnknownRrn	You reported no payment with this rrn.	No
AmountMismatch	The payment exists, with a different amount.	Yes, at UniPay's amount
BillMismatch	The payment exists, against a different bill from the billNo you sent.	Yes
CurrencyMismatch	The payment was made in a different currency from the transfer.	No
Reversed	The payment has been <u>reversed</u> : there's nothing to settle.	No
AlreadySettled	An earlier settlement already matched this payment.	No
NotSettledByProcessor	In unsettledByProcessor: a payment for that day you left out.	Yes

A payment counts as settled once a line **matches** it. So if you left a payment out, or reported the wrong amount, include it in a later transfer: that settlement matches it and clears the earlier difference. Include a matched payment again and it comes back `AlreadySettled`. New reasons may be added; treat one you don't recognise as a difference.

Every difference is also raised with Axis operations. Once you use day settlements, Axis is also alerted when a day on which you took payments gets no settlement.

Refusals

A settlement UniPay can't accept answers **HTTP 200** with "successful": false. Nothing is recorded; fix the request and send it again — you can reuse the transferReference.

message	Cause
Settlement date is required · Settlement date cannot be in the future · Settlement date cannot be more than 90 days in the past	Check settlementDate.
Total amount must be greater than zero · Fee amount cannot be negative	Fix the amount.
Total amount X does not equal the sum of the lines Y	With no fee, totalAmount must be the sum of the lines.
Total amount X does not equal the sum of the lines Y less the fee Z	totalAmount must be the sum of the lines less feeAmount.
Fee amount X must be less than the sum of the lines Y	Fix feeAmount.
Currency must be 3 characters	Fix currency.
Transfer reference is required · Transfer reference must not exceed 100 characters	Fix transferReference.
Credited account number must not exceed 64 characters · Biller code must not exceed 64 characters	Fix the field.
At least one line is required · A settlement may contain at most 5000 lines	Fix lines.
Every line needs an RRN · RRN 'X' must not exceed 50 characters · RRN 'X' appears more than once	Fix the line.

message	Cause
Amount for RRN 'X' must be greater than zero·Bill number for RRN 'X' must not exceed 100 characters	Fix the line.
Biller 'X' not found·Biller code 'X' is ambiguous; use the service provider number	Fix billerCode.
Every RRN in the settlement must belong to biller 'X'.	A payment you listed was made to another provider.
The RRNs belong to N service providers. A settlement covers one biller's transfer; submit one per biller.	Split the settlement by provider.
Transfer reference 'X' was already submitted with a different date, currency or total.	See Retrying .
A settlementDate that isn't a date at all — 30/09/2026, say — gets HTTP 400 , with the message '30/09/2026' is not a date. Use yyyy-MM-dd, for example "2025-01-17".	

Retrying

The transferReference identifies the settlement. If a call times out, or you get a 429 or 5xx, send the same request again:

- Same reference, same settlementDate, currency and totalAmount: you get the recorded result again, with isNewRecord: false. The lines aren't compared — the answer is always the first one recorded.
- Same reference with a different date, currency or total: refused, because that's a different transfer or a mistake.

A settlement can't be changed once recorded. If a transfer was wrong, correct it with your next transfer — the payments it matches are cleared as above. The general policy is in [Idempotency and retries](#).

Reading back

Read a settlement again at any time, for example to recover an answer you lost. Both endpoints answer in the same shape as `settle-day`, with `isNewRecord: false`, and only ever show your own settlements.

Call	Returns
GET /api/app/payment-processor/{id}/day-settlement	One settlement, by <code>settlementId</code> . An unknown ID gets 404.
GET /api/app/payment-processor/day-settlements	Your settlements, newest <code>settlementDate</code> first, as { "totalCount": ..., "items": [...] }.

The list takes `from` and `to` (settlement dates, `yyyy-MM-dd`, inclusive, both optional), `skipCount`, and `maxResultCount` — 20 by default, at most 100. A `from` after `to` is refused with the error object.

Your administrators also see every settlement and its differences in the UniPay portal, under **My settlements**.

What the service provider sees

The provider reads your settlement from its [processor settlements feed](#): your transfer reference, the account you credited and whether it recognises it, gross, fee and net amounts, whether it balanced, and every line with its outcome — including the payments you left out.

Service provider directory

The directory lists every active service provider on UniPay. Use it to build a “who can I pay?” list, to find a provider’s number or short code for account payments, or to check which currencies it accepts before offering them. Scope: `unipay:directory`.

List providers

[GET /api/app/directory/service-providers](#)

Query	Notes
<code>filter</code>	Partial, case-insensitive match on name, number, short code or TIN.
<code>country, town</code>	Exact match (case-insensitive).
<code>currency</code>	Only providers that can be settled in this currency.
<code>sorting, skipCount, maxResultCount</code>	Paging.

```
{
  "totalCount": 12,
  "items": [
    {
      "number": "4444",
      "shortCode": "MUT1",
      "name": "City of Mutare",
      "town": "Mutare",
      "country": "Zimbabwe",
      "phoneNumber": "+263...",
      "email": "billing@...",
      "tin": "2001234567",
      "defaultCurrency": "USD",
      "acceptedCurrencies": ["USD", "ZWG"]
    }
  ]
}
```

Providers change rarely: cache the list and refresh it daily rather than calling it per customer.

One provider

[GET /api/app/directory/service-providers/{providerRef}](#) — by number or short code. With the `unipay:directory.settlement` scope, the answer includes the provider's **settlement bank accounts**, with account numbers masked (**** * 1234).

If you initiate the bank transfers that settle payments, the `unipay:directory.settlement.reveal` scope lets you [reveal an account's full number](#). Every reveal is audited. Ask for this scope only if you need it.

Go-live checklist

Run through this on <https://test.unipay.co.zw>, then send the results to your Axis Solutions contact to request production credentials.

Authentication

- ☐ A token request returns 200 with an `access_token`.
- ☐ A wrong `client_secret` returns `invalid_client`; your system alerts rather than retrying in a loop.
- ☐ Under sustained load, your system requests about **one token per 30 minutes** per process — check your logs.
- ☐ An expired or revoked token (401) makes your client get a new token and retry once.
- ☐ The client secret is in a secrets vault, not in code or configuration files.

Lookup

- ☐ A valid bill number returns the bill; you show the customer `payableNow` and the bill's currency.
- ☐ An unknown bill number shows the customer a clear message, and no payment is offered.
- ☐ If you offer account payments: an account lookup returns status 1 (Owing), and statuses 0 and 3 are refused.
- ☐ A bill with an `expiryDate` in the past is refused before the customer is debited.

Payment

- ☐ A payment returns `"successful": true, "isNewRecord": true`.

- ☐ Sending the **same request again** returns "isNewRecord": false, and your system treats it as success.
- ☐ Each payment gets a new rrn, unique across your organisation, stored before the first attempt.
- ☐ A payment in a currency the provider doesn't accept is refused, and your system reverses the debit.
- ☐ A paymentDate in the future is refused; your paymentDate carries a UTC offset.
- ☐ A simulated timeout (e.g. block the connection after sending) leads to a retry with the same rrn — **not** a reversal.
- ☐ A 429 or 5xx leads to backoff and retry, capped at about five attempts, then an alert.

Day settlements

If you pay service providers directly:

- ☐ A settlement listing every payment of a test day returns "isBalanced": true.
- ☐ Leaving a payment out returns it in unsettledByProcessor; your system alerts rather than ignoring it.
- ☐ Sending the same settlement again returns "isNewRecord": false, and your system treats it as success.
- ☐ Each transfer is sent once per provider and currency, with its bank reference as transferReference.

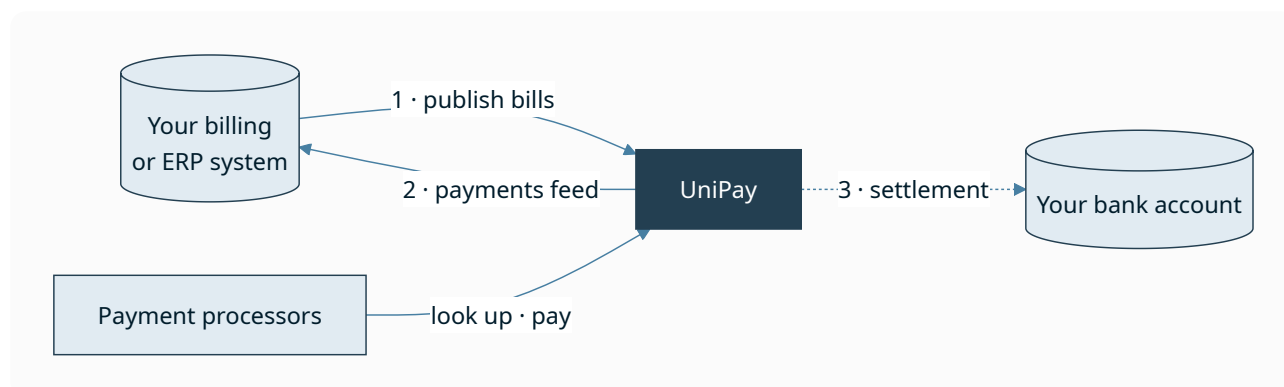
Operations

- ☐ You log every request and response with a timestamp, the rrn, and your client_id — with tokens and secrets redacted.
- ☐ You alert on the rate of "successful": false answers, not only on HTTP errors.
- ☐ Timeouts are set: 10 s to connect, 30 s to read.

- ☐ Separate credentials are ready for production; none are shared with test.

Service provider guide

As a service provider you raise bills, and UniPay lets every connected payment processor collect against them — banks, mobile-money operators, fintechs — without an integration with each. Your part of the [payment lifecycle](#):



1. **Publish your bills** so processors can find them, and republish them as they change — [Publish bills](#). Optionally, publish your account balances and exchange rates too — [Accounts and exchange rates](#).
2. **Collect the payments** made against them into your own system, and apply any [reversals](#) — [Collect payments](#).
3. The money is **settled** to your bank account, separately from the API.

UniPay never calls you: your system publishes and pulls on its own schedule.

The Axis Sync Engine

Most service providers don't call the API themselves. The **Axis Sync Engine**, installed beside your billing system (Promun, Sage and others), publishes your bills and collects your payments for you — get it and the bridge for your ERP from [Downloads](#). These pages describe what it does on your behalf — and what to build if you integrate directly.

What you can call

Everything lives under `/api/app/service-provider/` and `/api/app/bill-import/`. Your token needs `unipay:bills`, plus `unipay:payments` to read payments.

Task	Endpoint	Reference
Publish one bill	POST <code>/api/app/service-provider/bill</code>	<u>publishBill</u>
Publish up to 500 bills	POST <code>/api/app/service-provider/bills</code>	<u>publishBills</u>
Import up to 5,000 bills	POST <code>/api/app/bill-import/submit</code>	<u>submitBillImport</u>
Publish up to 500 account balances	POST <code>/api/app/service-provider/accounts</code>	<u>publishAccounts</u>
Finish an account balance snapshot	POST <code>/api/app/service-provider/finalize-account-sync</code>	<u>finalizeAccountSync</u>
Publish up to 1,000 exchange rates	POST <code>/api/app/service-provider/currency-rates</code>	<u>publishCurrencyRates</u>
Read your exchange rates	GET <code>/api/app/service-provider/currency-rates</code>	<u>listCurrencyRates</u>
Currencies you accept	GET <code>/api/app/service-provider/accepted-currencies</code>	<u>acceptedCurrencies</u>
Read new payments	GET <code>/api/app/service-provider/payments</code>	<u>listPayments</u>
Read reversals	GET <code>/api/app/service-provider/payment-reversals</code>	<u>listPaymentReversals</u>
Read processors' day settlements	GET <code>/api/app/service-provider/processor-</code>	<u>listProcessorSettlements</u>

Task	Endpoint	Reference
	settlements	

The [Service Provider API reference](#) lists every endpoint and field, including some used only by the Axis Sync Engine.

Publish bills

Publish a bill with `POST /api/app/service-provider/bill`, scope `unipay:bills`.

```
curl -sS --fail-with-body -X POST "$BASE_URL/api/app/service-
provider/bill" \
-H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
-d "{
  \"billNo\": \"$BILL_NO\",
  \"uniqueNo\": \"$BILL_NO\",
  \"accountNo\": \"$ACCOUNT_NO\",
  \"name\": \"Example Customer\",
  \"description\": \"Rates - October 2026\",
  \"billDate\": \"$(date +%Y-%m-%d)\",
  \"dueDate\": \"$(date -d '+30 days' +%Y-%m-%d)\",
  \"currency\": \"$CURRENCY\",
  \"amount\": $AMOUNT
}"
# →
{"successful":true,"message":"Added","isNewRecord":true,"status":"Create
d",...}
```

The bill

```
{
  "billNo": "4444INV0009",
  "uniqueNo": "INV0009",
  "accountNo": "100-2345",
  "name": "J. Moyo",
  "description": "Rates - September 2026",
  "billDate": "2026-09-01",
  "dueDate": "2026-09-30",
  "currency": "USD",
  "amount": 220.00,
  "expiryDate": null
}
```

Field	Required	Rules
billNo	Yes	What processors look the bill up by. 4–50 letters and digits only — no spaces or dashes. Unique within your organisation per currency; prefixing your provider number works well (4444INV0009).
uniqueNo	Yes	Your own internal reference: invoice, statement or case number. Needn't be unique.
accountNo	Recommended	The customer's account. Without it the bill can't be paid <u>by account</u> .
name		The account holder, as the customer will recognise it.
description	Yes	What the bill is for.
billDate, dueDate	Yes	Dates. Paying after dueDate is still accepted.
expiryDate		After this date the bill can't be paid.

Field	Required	Rules
currency	Yes	Three-letter ISO 4217 code: the bill's native currency.
amount	Yes	Signed. Positive: the account owes. Zero: settled. Negative: in credit.

Your organisation comes from your token; there's no provider field to send.

Publish a bill for every account, whatever its balance — zero and credit balances included. That's what lets any customer pay any account, and lets a lookup say "nothing owing" instead of "not found".

The answer

```
{ "successful": true, "message": "Added", "isNewRecord": true,
  "status": "Created" }
```

status	Meaning
Created	A new bill.
Updated	The bill existed and has been updated with what you sent.
AlreadyExists	The bill existed, unchanged. Nothing to do — publishing again is safe.
Blocked	successful: false. The bill already has payments against it, so it can't be changed automatically (errorCode: bill_has_payments). Raise a new bill instead.

A malformed bill — a billNo with a dash, a missing description — is rejected with **HTTP 400** listing each problem:

```
{
  "error": {
    "message": "Your request is not valid!",
    "validationErrors": [
      { "message": "Bill number can only contain letters and numbers",
    "members": [{"billNo"} ]
    ]
  }
}
```

Many bills at once

Volume	Endpoint	How
Up to 500	POST /api/app/service-provider/bills	Synchronous. Send {"bills": [...]}; the answer has a result per bill.
Up to 5,000	POST /api/app/bill-import/submit	Asynchronous. Send {"bills": [...], "idempotencyKey": "..."}; you get a batchId back.

A batch answers with a count and one result per bill; each result's status is a number here — 1 Created, 2 AlreadyExists, 3 Updated, 4 Blocked, 5 Failed:

```
{
  "total": 2, "succeeded": 2, "failed": 0,
  "results": [
    { "billNo": "4444INV0009", "successful": true, "message": "Added",
    "status": 1 },
    { "billNo": "4444INV0010", "successful": true, "message": "Bill
number '4444INV0010' already exists for currency USD.", "status": 2 }
  ]
}
```

For an **import**, give each submission its own idempotencyKey (up to 64 characters) and reuse it if you retry the submit. Then poll [GET /api/app/bill-](#)

import/status/{batchId} until status is 2 (Completed), 3 (CompletedWithErrors) or 4 (Failed) — 0 is Pending and 1 Processing — and fetch the failed rows from GET /api/app/bill-import/errors/{batchId}.

Accounts and exchange rates

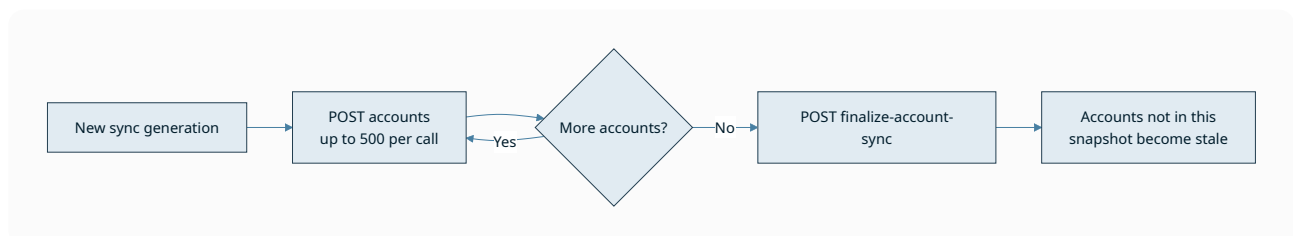
Bills say what a customer was charged. Two more things make UniPay's answers match your books between bill runs:

- **Account balances** — what each account owes right now, including what was paid at your own counters. Lookups then show processors today's figure, and tell them when an account has closed.
- **Exchange rates** — your own rates, by day. UniPay uses them to show a bill in another currency and to settle a payment made in one.

The Axis Sync Engine publishes both from your billing system. This page is for integrating directly. Every call here needs the `unipay:bills` scope.

Account balances

Publish a **complete snapshot** of your accounts in chunks, then tell UniPay the snapshot is finished:



Publish a chunk

[POST /api/app/service-provider/accounts](https://unipay.com/api/app/service-provider/accounts)

```
{
  "syncGeneration": 20260930,
  "accounts": [
    { "accountNumber": "100-2345", "name": "J. Moyo", "balance": 150.00,
"currency": "USD" },
    { "accountNumber": "100-2346", "name": "T. Dube", "balance": -20.00,
"currency": "USD", "customerReference": "CUST-881" }
  ]
}
```

```
curl -X POST https://test.unipay.co.zw/api/app/service-
provider/accounts \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d @accounts.json
```

Field	Required	Rules
syncGeneration	Yes	A number identifying this snapshot. Use a higher number for each new snapshot, and the same number on every chunk of it and on the finalize call. A timestamp such as 20260930 or 1759219200 works well.
accounts	Yes	1 to 500 accounts.
accountNumber	Yes	As on your bills. Spaces, dashes, dots and case are ignored, as in lookups.
currency	Yes	Three-letter ISO 4217 code. An account with balances in two currencies is sent once per currency.
balance	Yes	Signed , like a bill's amount: positive owes, zero settled, negative in credit.

Field	Required	Rules
name		The account holder. Left unchanged if you don't send it.
customerReference		Your own customer ID, stored as sent. Never shown to processors.

Each account is added, or updated if UniPay already has that account number in that currency:

```
{
  "inserted": 1,
  "updated": 1,
  "failed": 0,
  "errors": []
}
```

A row UniPay can't store is counted in failed and listed in errors, with its accountNumber and one of these messages; the rest of the chunk is stored.

message	Cause
Account number must contain letters or digits	Nothing left after spaces, dashes and dots are removed.
Currency is required	Empty currency.

An empty chunk, or more than 500 accounts, is refused with **HTTP 400** and At least one account is required. or Too many accounts for a synchronous bulk upsert; submit in smaller chunks. Sending a chunk again is safe: the same accounts are updated to the same figures.

Finish the snapshot

When **every** chunk has been accepted, call [POST /api/app/service-provider/finalize-account-sync](#) with the same generation:

```
{ "syncGeneration": 20260930 }
```

```
{ "successful": true, "message": "Reconciled; 12 account(s) marked stale." }
```

Every account you published in an **earlier** snapshot but not in this one is marked **stale**: closed, transferred, or otherwise no longer on your books. Calling it again is safe (0 account(s) marked stale).

Don't finalize until the whole snapshot is in: accounts you haven't sent yet would be marked stale too. If that happens, they come back as soon as you publish them.

What processors see

When a processor looks up an account:

- **amountDueNow** is your published balance in the currency being quoted, less any payments UniPay has recorded since you published it. So a balance of 150.00, then a 13.69 payment through UniPay, shows 136.31 until your next snapshot.
- **balanceAsAt** is when you last published that balance, and **accountBalances** lists every currency you publish for the account.
- **A stale account** looks up as status 3 (Unknown) with **successful**: false, and processors won't take payment. When a bill still exists for it, **balanceIsStale**: true warns them instead.
- An account with no live bill still answers by its **balance**: **NotOwing** for zero, **InCredit** for a credit.

Without account balances, lookups work out what's owed from your bills and the payments made through UniPay alone. See Bills and accounts for how processors use these figures.

Publish a snapshot at least once a day — after your day-end is a good time — so payments made at your own counters show up quickly.

Exchange rates

Publish rates

[POST /api/app/service-provider/currency-rates](#)

```
{
  "rates": [
    { "currencyCode": "USD", "baseCurrency": "USD", "rate": 1,
      "effectiveDate": "2026-09-30" },
    { "currencyCode": "ZWG", "baseCurrency": "USD", "rate": 0.037,
      "effectiveDate": "2026-09-30" }
  ]
}
```

Field	Rules
currencyCode	The currency the rate is for.
baseCurrency	Your base currency.
rate	How many units of baseCurrency one unit of currencyCode is worth. Above: 1 ZWG = 0.037 USD. Greater than zero.
effectiveDate	The day the rate applies from.

- **Send your base currency too, at rate 1**, for every date. UniPay converts between any two currencies through the base, so it needs a row for both.
- Send 1 to 1,000 rates per call. An empty list, or more than 1,000, is refused with **HTTP 400**.
- The answer is the number of rates stored, e.g. 2. A rate for a currency and date UniPay already has replaces it; rows with an empty code or a rate of zero or less are skipped and not counted.
- Rates are kept by date and add up over time. Publish each day's rates, every day.

A rate someone at your organisation keys by hand in the UniPay portal, under **Exchange rates**, wins for that day: your published rate for the same currency and date is skipped. Keying a rate by hand is done only in the portal, not through the API.

How rates are used

Where	Rate used
A processor <u>asks for a bill in another currency</u>	Your latest rate on or before today, up to 31 days old. The converted figure is marked indicative.
A payment in another currency <u>settles a bill</u>	Your own rate for the payment date — by default, one no older than the day before. Without one, the payment is recorded but not matched to a bill.

With the rates above, a customer who pays ZWG 370.00 against a USD bill settles USD 13.69 of it. The payments feed shows both, with `allocationRate 0.037`.

Read your rates

[GET /api/app/service-provider/currency-rates?days=30](#) lists your rates from the last days days (1 to 366, default 30), newest first. Each row has source — 1 published through the API, 2 keyed in the portal — and, for keyed rates, who entered it.

Accepted currencies

[GET /api/app/service-provider/accepted-currencies](#) returns the currencies processors may take payment in for your bills — what they see as `acceptedCurrencies`:

```
[ "USD", "ZWG" ]
```

Axis sets the list up with you; ask your Axis contact to change it. Where none has been set, it's every currency UniPay supports plus any you bill in or publish rates for.

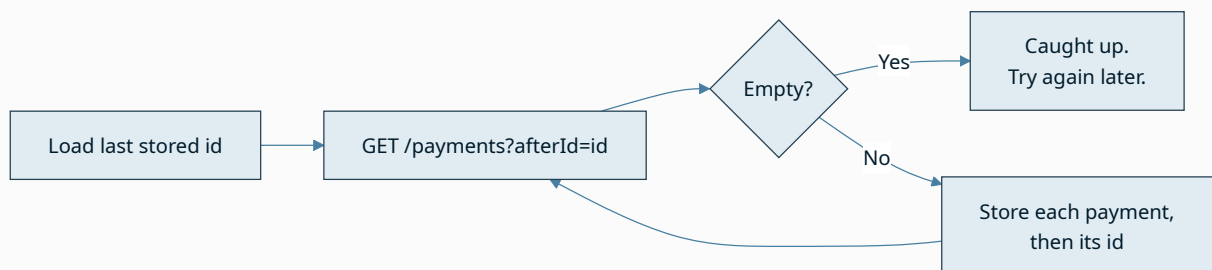
Collect payments

UniPay doesn't push payments to you. Your system **pulls** them from the payments feed ([GET /api/app/service-provider/payments](#)) as often as suits you, resuming from where it last stopped. Scopes: `unipay:bills` **and** `unipay:payments`.



```
# Omit afterId the first time; afterwards pass the id of the last
payment you stored.
curl -sS --fail-with-body "$BASE_URL/api/app/service-
provider/payments${AFTER_ID:+?afterId=$AFTER_ID}" \
  -H "Authorization: Bearer $TOKEN"
# → [{"payments":
[{"id":"01a0ee4f-...", "rrn":"...", "billNo":"...", "amount":1.4, "currency":"ZWG"
,
#
"allocatedAmount":1.4, "allocatedCurrency":"ZWG", ...}], "successful":true, ...}
, ...]
#      An empty array [] means you're up to date.
```

How the feed works



- **afterId** is the id of the last payment you stored. You get only payments **after** it, oldest first, up to 100 per call.
- Leave `afterId` out on your first call to start from the beginning.

- Keep calling until you get an empty array `[]`: you're caught up. Poll again later — every few minutes is plenty.
- **Store the payment, then the id**, in that order and ideally in one transaction. If you crash in between, you re-read that payment next time — so make storing it idempotent on id.

What you get

The feed is a JSON array. Each element wraps **one** payment:

```
[
  {
    "successful": true,
    "message": "Found",
    "payments": [
      {
        "id": "01a0ee4f-b160-75ef-9036-c51fd40af1d1",
        "rrn": "PX20260929000123",
        "billNo": "4444INV0009",
        "uniqueNo": "INV0009",
        "accountNumber": "100-2345",
        "serviceProvider": "4444",
        "paymentDate": "2026-09-29T08:15:00Z",
        "amount": 170.00,
        "currency": "USD",
        "allocatedAmount": 170.00,
        "allocatedCurrency": "USD",
        "allocationRate": null
      }
    ]
  }
]
```

Field	Meaning
id	UniPay's ID for the payment. Use it as your checkpoint and your de-duplication key.

Field	Meaning
rrn	The paying processor's reference.
billNo, accountNumber	What was paid.
uniqueNo	Your reference for the bill on its first payment; the rrn on later ones.
paymentDate	When the customer paid, in UTC.
amount, currency	What the customer tendered .
allocatedAmount, allocatedCurrency	What it settled against the bill, in the bill's currency.
allocationRate	The exchange rate used, when the tender was converted.

allocatedAmount and allocatedCurrency are **null** when the payment couldn't be matched to a bill — for example, paid in a currency the account has no bill in, with no exchange rate available. Record those as unallocated and resolve them through your exceptions process; see [Currencies](#).

Reversals

A payment can be reversed after you've collected it. Reversals have their own feed, read exactly the same way: [GET /api/app/service-provider/payment-reversals?afterId=...](#)

```
[
  {
    "id": "01a0f0c2-...",
    "paymentId": "01a0ee4f-b160-75ef-9036-c51fd40af1d1",
    "rrn": "PX20260929000123",
    "billNo": "4444INV0009",
    "reversedAt": "2026-09-30T09:00:00Z",
    "category": "...",
    "reason": "..."
  }
]
```

Match `paymentId` to the payment you stored and reverse it in your system. Keep a separate checkpoint for this feed. Only approved reversals appear; [Reversals](#) explains how a payment comes to be reversed.

Processors' day settlements

When a processor transfers a day's takings to you, it [reports the transfer](#) to UniPay. The [processor settlements feed](#) lists those reports — processor, settlement date, transfer reference, the account credited, gross, fee and net amounts, and the payments included — for reconciling your bank statement. Read it the same way, with its own checkpoint; it returns up to 50 per call.

Downloads

Service providers usually connect to UniPay through the **Axis Sync Engine**, installed beside their billing system. It publishes your bills to UniPay and posts the payments it collects into your ERP. It reaches your ERP through a **bridge**: one for Promun, one for Sage Pastel. Install the engine and the bridge for your ERP on the same Windows server.

Installer	What it does	Latest stable	Every release
Axis Sync Engine	Publishes bills to UniPay and posts payments into your ERP. Windows service with a local dashboard.	Download MSI	Releases
Promun bridge	Connects the Sync Engine to Promun (OpenEdge). Windows service, 32-bit.	Download MSI	Releases
Sage Pastel bridge	Connects the Sync Engine to Sage Pastel (v10.1). Windows service, 32-bit; a setup wizard finds your Sage companies.	Download MSI	Releases

Stable and newer builds

Download MSI always gets the current **stable** release — the one Axis Solutions has promoted for general use. The releases page lists every build, newest first, with the stable one marked at the top. Install a newer build only when your Axis Solutions contact asks you to.

A specific version is always at the same address, so you can reinstall exactly what you had.

Before you install

- A Windows server that can reach both your billing system and `https://online.unipay.co.zw` (or `https://test.unipay.co.zw` while testing).
- Administrator rights to install Windows services.
- Your UniPay **service provider credentials** — see [Environments and access](#). The Sync Engine uses them to connect to UniPay.
- For Promun: the OpenEdge client on the same server. For Sage Pastel: Sage Pastel v10.1 installed there.

Your Axis Solutions contact normally installs and configures these with you the first time. Before upgrading, check with them — and back up the bridge's configuration.

Integrating directly instead? You don't need any of these — see [Publish bills](#) and [Collect payments](#).

Go-live checklist

Run through this on `https://test.unipay.co.zw`, then send the results to your Axis Solutions contact to request production credentials. If the Axis Sync Engine integrates for you, your Axis contact runs these checks with you.

Authentication

- ☐ A token request with scope `unipay:bills unipay:payments` returns 200.
- ☐ Your system requests about one token per 30 minutes per process, and gets a new one on 401.
- ☐ The client secret is in a secrets vault, not in code or configuration files.

Publishing bills

- ☐ A new bill returns `Created`; publishing it again unchanged returns `AlreadyExists`.
- ☐ Changing a bill's amount and publishing again returns `Updated`.
- ☐ A bill number with a dash or space is rejected — your system strips separators before publishing.
- ☐ Accounts with zero and credit balances get bills too (amount 0 or negative).
- ☐ A processor on the test environment can find your bill by number and by account.

Account balances and rates

If you publish them:

- ☐ A snapshot published in chunks, then finalized, makes an account lookup show your balance as `amountDueNow`.
- ☐ An account left out of the next snapshot looks up as status 3 (Unknown) after finalizing.

- ☐ Each snapshot uses a higher `syncGeneration` than the last, and the same one on every chunk.
- ☐ Your rates, including your base currency at 1, appear in `GET currency-rates` for each day.

Collecting payments

- ☐ The first read, without `afterId`, returns the test payments made against your bills.
- ☐ Your checkpoint survives a restart: the next read returns only newer payments.
- ☐ Killing your process between storing a payment and storing its `id` leads to one stored payment, not two.
- ☐ A payment with `allocatedAmount: null` lands in your exceptions process, not in an account.
- ☐ A reversal from the reversals feed is applied to the right payment.

Operations

- ☐ You log each call with a timestamp and your `client_id`, with tokens and secrets redacted.
- ☐ You alert if the feed hasn't been read successfully for longer than you'd accept.
- ☐ Separate credentials are ready for production; none are shared with test.

Errors, limits and operations

Two kinds of failure

UniPay understood the request but won't carry it out — a bill that doesn't exist, a currency the provider doesn't accept, a payment date out of range. These answer **HTTP 200** with `successful: false` and a message:

```
{ "successful": false, "message": "Bill 'INV10023' not found. Cannot process payment.", "isNewRecord": true }
```

Always check `successful`, not just the HTTP status. Show or log the message, but don't match on its exact wording, which may be improved over time. Sending the same request again gets the same answer.

The request itself failed — reported with an HTTP status:

Status	Body	Meaning	What to do
400	Error object	The request couldn't be read, or a field is invalid.	Fix the request.
401	Empty	Token missing, invalid or expired.	Get a new token and retry once .
403	Empty	Your token lacks the <u>scope</u> this call needs.	Request the scope, or ask your Axis contact to enable it.
403	Error object	Your organisation can't use this endpoint. code <code>Authorization:PartnerTypeMismatch</code>	Check you're using the right credentials.

Status	Body	Meaning	What to do
		means payment-processor credentials called a service-provider endpoint, or the reverse.	
404	Empty	Wrong URL.	Check the path.
429	Empty	<u>Rate limit</u> exceeded.	Wait, then retry with backoff.
500, 502, 503, 504	Varies	UniPay can't complete the request right now.	Retry with backoff — a payment retried with the same rrn is <u>never recorded twice</u> .

The **error object** looks like this; validationErrors lists each invalid field:

```
{
  "error": {
    "code": null,
    "message": "Your request is not valid!",
    "details": "The following errors were detected during
validation.\r\n - The Description field is required.\r\n",
    "validationErrors": [
      { "message": "The Description field is required.", "members":
["description"] }
    ]
  }
}
```

Business messages

Every successful: false message the partner endpoints return. X stands for the value you sent.

Looking up a bill

Message	Cause
Bill 'X' not found	No bill with that number.
Bill number is required	Empty billNo.
Bill number must be at least 4 characters · Bill number cannot exceed 50 characters	Wrong length.
Bill number can only contain letters and numbers	Other characters after spaces, dashes and dots are removed.
Currency must be a 3-letter ISO 4217 code	Malformed currency.
Provide either a bill number or a service provider and account number, not both	lookup got both.
Provide a bill number, or a service provider and account number	lookup got neither.
Service provider is required · Account number is required	Account lookup missing a field.
Service provider cannot exceed 50 characters · Account number cannot exceed 64 characters · Account number must contain letters or digits	Malformed account lookup.
Service provider 'X' not found	Unknown provider number or short code.
Service provider short code 'X' is ambiguous; use the provider number	Use the number.

Message	Cause
No outstanding bill found and no account on record for this number	Status 0 — NotFound.
Account standing is stale; no current information for this number	Status 3 — Unknown. Don't take payment.

Paying a bill

Message	Cause
Bill 'X' not found. Cannot process payment.	No bill with that number.
No bill found for that account. Cannot process payment.	Account payment: no bill on the account.
Currency 'X' is not accepted by this service provider. Accepted currencies: ...	See Currencies .
Invoice has expired	The bill's expiryDate has passed.
Payment would exceed the remaining balance on this invoice	Overpayment, where the provider refuses them.
Payment date is required · Payment date cannot be in the future · Payment date cannot be more than 90 days in the past	Check paymentDate and its offset.
Bill number or account number is required · Bill number can only contain letters and numbers · Account number must not exceed 50 characters	Fix the bill or account.

Message	Cause
Service provider is required when paying by account number	Add serviceProvider.
Service provider 'X' not found · Service provider short code 'X' is ambiguous; use the provider number	Fix serviceProvider.
RRN 'X' was already used for a different payment	One of your own rrn values reused for a different bill, account, amount or currency.
Retrieval Reference Number is required · RRN must not exceed 50 characters	Fix rrn.
Payment amount must be greater than zero · Currency must be 3 characters	Fix the field.

Success messages: Payment Saved for a new payment; Duplicate notification; already processed. when UniPay already had the rrn (isNewRecord: false) — both are success.

Reporting a day settlement

Every refusal is listed in [Day settlements](#). Success messages: Settlement balanced — all payments matched. and Settlement accepted with N difference(s). — both mean the settlement was recorded.

Publishing a bill

Message	status	Cause
Added	Created	New bill.
Updated	Updated	Existing bill changed.

Message	status	Cause
Bill number 'X' already exists for currency C.	AlreadyExists	Unchanged; nothing to do. successful is true.
Bill number 'X' already has payments and cannot be updated automatically.	Blocked	Raise a new bill instead.
Too many bills for a synchronous bulk create; submit in smaller chunks.		More than 500 in one bills call — use an import.

Rate limits

Endpoint	Limit
POST /connect/token	10 requests a minute
/api/app/payment-processor/*	60 requests a minute
All other /api/*	300 requests a minute

Limits are counted **per client IP address**: every process calling from the same address shares them. Over the limit you get 429 with no body — pause and retry with backoff. Caching your access token for its full 30 minutes keeps you well clear of the token limit. If you need more throughput, talk to your Axis contact.

Timeouts

Set a **10-second connect timeout** and a **30-second read timeout**. UniPay normally answers in well under a second.

Health

Endpoint	Answers
<u>GET</u> <u>/api/app/health/live</u>	200 {"status": "UP"} while UniPay is running.
<u>GET</u> <u>/api/app/health/ready</u>	200 with status HEALTHY or DEGRADED, or 503 with UNHEALTHY, after checking its database and services.

Neither needs a token. Use them to tell “UniPay is down” apart from “my request is wrong” — not as a substitute for handling errors on each call.

Versioning and change policy

The API version

Every UniPay endpoint is **version 1**. Within v1:

- **Additive changes can ship at any time** — new endpoints, new optional request fields, new response fields, new error codes. Build your integration to ignore response fields it does not recognise.
- **Breaking changes are avoided**. When one is unavoidable, it is marked **⚠ Breaking** in What's changed, with what you need to do, and it reaches the test environment before production.

A future v2 would run alongside v1, with its own docs.

Test first, then production

A change always reaches the test environment first. The build that is promoted to production is byte-for-byte the build that was tested — the API **and these docs** — so what you certified against in test is what you get in production.

Pages marked **Preview** describe something present in the build but not yet switched on for everyone. Don't depend on it in production until the preview marker is gone.

What's changed

Changes that affect how you integrate with UniPay, newest first. Entries marked **Breaking** need action from you. Changes that don't alter the API contract aren't listed here.

2026-10-01 — A fiscal receipt per payment

- **POST /api/app/service-provider/fiscalization-result** takes an **optional paymentId** (the Unipay payment id, as in `erp-posting-result`). When you send it, the receipt is recorded on that payment — every payment on a bill can carry its own — and the bill keeps the first payment's receipt as its QR. Leave it out and nothing changes: a second payment's receipt on the same bill is still refused with 409.

2026-09-30 — Payment references, lookup and credentials

- **Payment references (rrn) are per processor.** A payment whose rrn another processor has already used is now recorded normally. Re-sending one of your own payments still returns `"isNewRecord": false`. **Re-using one of your own rrn values for a different bill, account, amount or currency is now refused** with RRN 'X' was already used for a different payment — previously it was answered as a success and nothing was recorded. See [Idempotency and retries](#).
- **POST /lookup by bill number** now returns `acceptedCurrencies`, `requestedCurrencyAccepted`, `rateIsStale` and `errorCode`, as `GET /bill` does. Previously they came back empty. Whenever a bill exists it's included for `NotOwing` and `InCredit` too — check `bill` rather than inferring it from `status`.
- **Credentials stay on their own side.** Payment-processor endpoints refuse service-provider credentials, and service-provider endpoints refuse payment-processor credentials, with 403 and code

Authorization:PartnerTypeMismatch. New credentials carry only their own side's scopes and the `client_credentials` grant; existing credentials are unchanged.

2026-09 — Developer docs

These docs replace the *UNIPAY Payment Processor Integration Guide* PDF, and now cover service providers too. They're served by every UniPay environment at `/docs/` and always match that environment's build.

If you integrated from the PDF, note where the API has moved on since it was written:

- **Lookup:** `POST /lookup` finds a bill by number **or by account** in one call. `GET /bill` still works.
- **Currencies:** a payment no longer has to be in the bill's currency. Any of the provider's `acceptedCurrencies` is accepted; see [Currencies](#). The message *Payment currency does not match invoice currency* no longer occurs.
- **Overpayments** are accepted unless the service provider has chosen to refuse them.
- **Rate limits** are 10 / 60 / 300 requests a minute, counted per client IP — see [Rate limits](#).
- **Error bodies:** 401, 403 and 429 responses have an empty body; other errors use the error object described in [Errors, limits and operations](#).
- **The legacy IP-based test address** has been retired. Use `https://test.unipay.co.zw`.

Support

Contact your Axis Solutions integration contact. To get an answer quickly, include:

1. **The environment** — test or production — and the **build** shown at the foot of these pages.
2. **Your client_id** — never the secret.
3. **When** the call was made, in UTC, to the second.
4. **The request and response**: method, path, body and HTTP status. Remove tokens, secrets and card numbers first.
5. For a payment, its **rrn**; for a bill, its **billNo**.
6. **What you expected**, and what happened instead, in one sentence each.

