Pago46 Documentation Center

Pago46 API technical documentation · v2026.8.2

Contents

Latin America's cash network 4
Actors 4
Services 4
- Pay-In (Payments) 4
- Pay-Out (Withdrawals) 4
How the integration works 4
Choose your guide 5
- I'm a Merchant 5
- I'm a Payment Provider 5

Countries 6
Regional Coverage 6
- Active Countries and Local Currencies (Fiat) 6
- Foreign Currency 6
Next Steps 6
- I'm a Merchant 7
- I'm a Payment Provider 7

Environments 8
Traditional integration flow 8
- Testing (Sandbox) 8
- Production 8
- Availability monitoring 8
Choose your guide 9
- I'm a Merchant 9
- I'm a Payment Provider 9

Authentication 10
Required headers 10
Building the signature 10
- 1. String to sign 10
- 2. Compute the hash 10
Implementation examples 11
Authentication errors 12
Choose your guide 13
- I'm a Merchant 13
- I'm a Payment Provider 13

OpenAPI 14
Available resources 14
Recommended usage for integrators 14
Best practices 14
Choose your guide 14
- I'm a Merchant 14
- I'm a Payment Provider 14

Build with AI 15
- Chat in your browser 15
- Use your coding agent 15
The prompts 15
How it works 16

Handling Common Errors 17
Error response format 17
Common errors 17
- 403 Forbidden 17
- 405 Method Not Allowed 18
- 406 Not Acceptable 18
- 415 Unsupported Media Type 18
- 500 Internal Server Error 18
Choose your guide 19
- I'm a Merchant 19
- I'm a Payment Provider 19

Merchant Integration 20
Available services 20
- Pay-In (Collections) 20
- Pay-Out (Disbursements) 20
- Foreign Currency 20
- Checkout 20
Integration flow 21
- Integration steps 21
Environments 21
Order lifecycle 21
Coverage 22
• Payments (Pay-In) 23
- General flow 23
- Prerequisites 24
- 1. Create a payment order 24
- 2. Query an order 26
- Order states 27
- Webhooks (notifications) 28
- Common errors 31
- Best practices 32
- Complete integration example 32
- Next steps 34
• Withdrawals (Pay-Out) 35
- General flow 35
- Prerequisites 36
- 1. Create a withdrawal order 36
- 2. Query an order 38
- Order states 39
- Webhooks (notifications) 40
- Common errors 43
- Best practices 44
- Complete integration example 44
- Next steps 46
• Foreign Currency Withdrawals (Pay-Out) 47
- General flow 47
- Prerequisites 48
- 1) Create a quote 48
- 2) Create a foreign withdrawal order 49
- 3) Sign and broadcast the on-chain transaction 51
- 4) Retrieve an order 52
- Order states (foreign pay-out) 52
- Webhooks (notifications) 53
- Integration best practices 56
• Checkout 57
- What is Checkout? 57
- Environments 57
- Iframe integration 57
- Mobile app integration 58
- Browser permissions 58
- Recommended test flow 58
- Demos 59

Payment Provider Integration 60
Available services 60
- Pay-In (Collections) 60
- Pay-Out (Disbursements) 60
Integration flow 60
- Integration steps 60
Environments 61
Order lifecycle 61
Coverage 63
• Payments (Pay-In) 63
- Payment network registration 63
- Order lifecycle 63
- 1. Verify the order 64
- 2. Start the collection (lock) 65
- 3. Confirm the collection (finalization) 66
- Cancellation 66
- Status summary 67
• Withdrawals (Pay-Out) 68
- Payment network registration 68
- Order lifecycle 68
- 1. Verify the order 69
- 2. Start the payment (lock) 70
- 3. Confirm the payment (completion) 71
- Cancellation (optional) 72
- Status summary 72

Service Status 73
What information is available? 73
Subscriptions and alerts 73
- Email 73
- Slack 73
Embedded status widget in this documentation 73
When to check it 73
Choose your guide 74
- I'm a Merchant 74
- I'm a Payment Provider 74

Design Guide 75
Visual identity 75
- Institutional colors 75
- Typography 75
Primary logo variants 75
Button catalog 76
Implementation examples 80
Restrictions and practices to avoid 81
Choose your guide 81
- I'm a Merchant 81
- I'm a Payment Provider 81

Latin America's cash network

Pago46 connects the digital world with cash across Latin America. With a single integration to the **Pago46 API**, you can process cash **payments (Pay-In)** and **withdrawals (Pay-Out)** through a network of physical points—retail chains, banks, pharmacies, and kiosks—and a network of agents who collect and deliver money at geolocated points.

Actors

The API coordinates transactions between three roles. Identify yours to know which guide to follow.

Actor	Role	Description
Merchant	Creates the order	Online businesses, e-commerce platforms, wallets, or fintechs that need to **collect** from their users or **disburse** funds.
Payment Provider	Processes the order	Physical networks (banks, pharmacies, kiosks) that **process** the cash for orders created by Merchants.
Consumer	Delivers or receives the cash	The end user who pays or collects at a point in the network.

Services

The Pago46 API handles local currency and foreign currency with the same semantics.

Pay-In (Payments)

Your users pay in cash at any point in the network.

- **Merchant:** you create an order and get a code/QR; you receive the money reconciled digitally.
- **Provider:** you validate the code, receive the cash, and confirm the operation.

Pay-Out (Withdrawals)

Your users withdraw cash or receive a disbursement at a physical point.

- **Merchant:** you create a withdrawal order for your user.
- **Provider:** you verify the order, lock it, and hand over the cash.

How the integration works

The architecture is **server-to-server**, and every request is signed with **HMAC SHA-256**.

1. **Authentication** — you sign every request with HMAC SHA-256.
2. **Order creation** — the Merchant creates the payment or withdrawal intent.
3. **Physical operation** — the user goes to a point in the network.
4. **Confirmation** — the Provider confirms the money changed hands.
5. **Webhooks** — Pago46 notifies the Merchant of the result in real time.

BEFORE YOU START

Authentication is the prerequisite for any API call. Start there.

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

Before integrating, check country coverage to confirm operational availability.

Countries

Regional Coverage

With **a single integration**, you connect to the most extensive cash payment network in Latin America. Pago46 operates natively or through partner *Payment Networks*, guaranteeing unified settlement and consistent SLAs across all markets.

Active Countries and Local Currencies (Fiat)

We currently process operations in local currency (fiat) in the following territories. Make sure you use the correct `currency_code` in your API requests for each jurisdiction.

Country	Accepted Currency	ISO Code (Currency)
🇦🇷 **Argentina**	Argentine Peso	`ARS`
🇨🇱 **Chile**	Chilean Peso	`CLP`
🇪🇨 **Ecuador**	US Dollar	`USD`
🇲🇽 **Mexico**	Mexican Peso	`MXN`
🇵🇪 **Peru**	Sol	`PEN`
🇬🇹 **Guatemala**	Quetzal	`GTQ`

> **Country Configuration**
>
> Behavior and functionality can be configured per country, allowing different payment flows, fees, and business rules adapted to local regulations.

Foreign Currency

We offer foreign-currency functionality to process transactions with currency conversion. This lets your users pay or receive funds in a currency other than the local one, making international expansion possible without multiple integrations.

Source Currency → Target Currency	Description	Available Countries
USDC → MXN	Allows users to send **USDC** via **Solana**, **Monad**, or **Stellar** and **immediately receive Mexican pesos at physical points**.	🇲🇽 Mexico

---

Next Steps

Select the guide that best fits your integration needs:

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

Questions about expanding to your country? Contact your account executive or email contacto@pago46.com.

Environments

Pago46 operates in two distinct environments: **Sandbox** for testing and **Production** for real transactions. Each environment has its own characteristics and configuration requirements.

Traditional integration flow

1. Identify whether you are a **Merchant** or a **Payment Provider**.
2. Identify the flow you want to integrate: **Collections (Pay-In)** or **Withdrawals (Pay-Out)**.
3. Send an email to `contacto@pago46.com` indicating your role (Merchant or Payment Provider), country of operation, and the flow you want to integrate.
4. You will receive credentials for the **Sandbox** environment and access to the documentation.
5. Perform the required tests in **Sandbox**.
6. When you are ready for production, request production credentials from your account manager.

Testing (Sandbox)

Pago46 Service	Base URL	Notes
Main API	`https://api.dev.pago46.io/api/v1`	HTTPS REST API integration point with HMAC signature.
Checkout (web payment application)	`https://checkout.dev.pago46.io`	Hosted page for consumers to complete payment flows.

*Amounts processed in sandbox **do not** generate real money movements.*

Production

Pago46 Service	Base URL	Notes
Main API	`https://api.prd.pago46.io/api/v1`	HTTPS REST API integration point with HMAC signature.
Checkout (web payment application)	`https://checkout.prd.pago46.io`	Hosted page for consumers to complete payment flows.
Pago46 app (web)	`https://app.pago46.io`	Works in any browser, with nothing to install. It can be added to the home screen on iPhone and Android.
Pago46 app (Android)	Google Play	The same app published on Google Play.

*Anyone can create an account in the Pago46 app; to operate as a **Socio46** partner you must complete identity verification.*

Availability monitoring

For production operational visibility, review the Service Status page, where you can find active incidents, scheduled maintenance, and subscription options for email and Slack.

> **DIFFERENT CREDENTIALS**
>
> Sandbox and production keys **are not interchangeable**. Request them from your account manager.

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

Authentication

Every request to the Pago46 API is signed with **HMAC SHA-256**. You compute a signature from your secret key and the request contents; the server recomputes the signature and compares the two values. A match confirms two things: that the request is yours and that nobody altered the contents in transit.

The scheme is identical for Merchants and Payment Providers. The only thing that changes is the name of the header where you send your public key.

Required headers

Every HTTP request must include these three headers:

Header	Description
`Merchant-Key` or `Provider-Key`	Your public key. Use `Merchant-Key` if you are a Merchant and `Provider-Key` if you are a Payment Provider.
`Message-Date`	Unix timestamp in **seconds** (integer or decimal).
`Message-Hash`	The HMAC SHA-256 signature in hexadecimal (see below).

> **VALIDITY WINDOW**
>
> The server rejects the request if `Message-Date` differs from the server time by more than **5 minutes**, to prevent replay attacks. Always use the current time, **in seconds**: sending milliseconds puts it outside the window and the request is rejected.

Building the signature

1. String to sign

Concatenate these five values separated by colons (`:`), in this exact order:

```
KEY:MESSAGE_DATE:METHOD:PATH:BODY
```

1. **Public key** — the same value as your `Merchant-Key` or `Provider-Key` header.
2. **Message-Date** — the same value you send in the header.
3. **HTTP method** — `POST`, `GET`, etc.
4. **Path** — the resource path **without the query string**, for example `/api/v1/merchants/orders/pay-in/`.
5. **Body** — the raw request body in UTF-8. If the request has no body (for example a `GET`), use an empty string.

Parameters are not sorted and the JSON is not normalized. Sign exactly the same bytes you send in the body: any extra space or reordering produces a different signature.

2. Compute the hash

Sign the string with HMAC SHA-256 using your **secret** as the key, and output the result in hexadecimal (hexdigest).

Implementation examples

The same signer works for Merchants and Providers; only the key header name changes.

Python

```
import hashlib
import hmac
import json
import time

import requests

HOST = "https://api.dev.pago46.io"


def signed_request(key, secret, key_header, method, path, body=None):
    timestamp = str(time.time())  # Unix in seconds
    body_str = json.dumps(body) if body else ""

    string_to_sign = f"{key}:{timestamp}:{method}:{path}:{body_str}"
    signature = hmac.new(
        secret.encode("utf-8"),
        string_to_sign.encode("utf-8"),
        hashlib.sha256,
    ).hexdigest()

    headers = {
        key_header: key,  # "Merchant-Key" or "Provider-Key"
        "Message-Date": timestamp,
        "Message-Hash": signature,
        "Content-Type": "application/json",
    }
    return requests.request(method, f"{HOST}{path}", headers=headers, data=body_str)


# Example: a Merchant creates a pay-in order
response = signed_request(
    key="YOUR_MERCHANT_KEY",
    secret="YOUR_MERCHANT_SECRET",
    key_header="Merchant-Key",
    method="POST",
    path="/api/v1/merchants/orders/pay-in/",
    body={"order_type": "LocalCurrencyOrder", "price": "100.00", "price_currency": "CLP"},
)
print(response.status_code)
```

Node.js

```
const crypto = require('crypto');
const axios = require('axios');

const HOST = 'https://api.dev.pago46.io';

async function signedRequest({ key, secret, keyHeader, method, path, body = null }) {
  const timestamp = (Date.now() / 1000).toString(); // Unix in seconds
  const bodyStr = body ? JSON.stringify(body) : '';

  const stringToSign = [key, timestamp, method, path, bodyStr].join(':');
  const signature = crypto
    .createHmac('sha256', secret)
    .update(stringToSign)
    .digest('hex');

  return axios({
    method,
    url: `${HOST}${path}`,
    headers: {
      [keyHeader]: key, // 'Merchant-Key' or 'Provider-Key'
      'Message-Date': timestamp,
      'Message-Hash': signature,
      'Content-Type': 'application/json',
```

```
    },
    data: bodyStr,
  });
}
```

Go

```
package main

import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/hex"
        "strconv"
        "strings"
        "time"
)

// keyHeader is "Merchant-Key" or "Provider-Key".
func authHeaders(key, secret, keyHeader, method, path, bodyStr string) map[string]string {
        timestamp := strconv.FormatInt(time.Now().Unix(), 10) // Unix in seconds

        stringToSign := strings.Join([]string{key, timestamp, method, path, bodyStr}, ":")

        h := hmac.New(sha256.New, []byte(secret))
        h.Write([]byte(stringToSign))
        signature := hex.EncodeToString(h.Sum(nil))

        return map[string]string{
                keyHeader:      key,
                "Message-Date": timestamp,
                "Message-Hash": signature,
        }
}
```

Authentication errors

When authentication fails, the API returns **`403 Forbidden`** with the standard error format:

```
{
  "type": "client_error",
  "errors": [
    {
      "code": "authentication_failed",
      "detail": "Incorrect authentication credentials.",
      "attr": null
    }
  ]
}
```

Probable cause	How to fix it
A required header is missing	Include `Merchant-Key` / `Provider-Key`, `Message-Date`, and `Message-Hash`.
`Message-Date` outside the 5-minute window	Use the current system time, in seconds.
Unknown public key	Make sure you are using the key for the correct environment (sandbox vs. production).
The signature does not match	Check the string order, the `:` separator, the path without query string, and that the body is identical to the one sent.

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

OpenAPI

Pago46's OpenAPI contract is the official reference for integrating our REST services. It defines endpoints, methods, parameters, payloads, responses, and error schemas, helping you build faster with less rework.

Available resources

- **OpenAPI Contract (YAML)**: Download specification
- **Swagger UI**: Explore and test endpoints
- **Redoc UI**: Browse the documentation

Recommended usage for integrators

1. **Start with the contract** to understand resources, operations, and models.
2. **Validate authentication and headers** before consuming business endpoints.
3. **Use Swagger UI or Redoc** to review examples and expected behavior.
4. **Generate technical artifacts** (clients, mocks, or tests) from the YAML when it fits your stack.
5. **Test in Sandbox** and move to production only after validating critical flows and error handling.

Best practices

- Treat OpenAPI as the technical source of truth across the integration lifecycle.
- Implement schema validation to prevent payload format and type errors.
- Standardize error handling based on documented status codes and responses.
- Review documentation regularly to identify updates and new capabilities.

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

Build with AI

Integrating Pago46 with the help of an AI assistant? We give models all the context they need. Our docs publish an `llms.txt` index that links to a clean **Markdown** copy of every page, so any assistant can read the pay-in, pay-out, authentication, and checkout flows without wrestling with the site's HTML.

Pick your assistant below to open it with the integration prompt already loaded, or copy a ready-to-run command for your terminal coding agent.

Our docs publish an `llms.txt` index that points any AI at a clean Markdown copy of every page. Launch your assistant with the integration prompt ready, or copy a ready-to-run command for your terminal coding agent.

Copy prompt | Open llms.txt

CHAT IN YOUR BROWSER

Opens the assistant with the prompt loaded. Gemini has no prefill — use “Copy prompt” and paste it.

- **Claude** — Anthropic →
- **ChatGPT** — OpenAI →
- **Perplexity** — Search + browse →
- **Grok** — xAI →
- **Le Chat** — Mistral AI →
- **Gemini** — Google →

USE YOUR CODING AGENT

Copy a ready-to-run command for your terminal.

- **Claude Code** — Anthropic CLI
- **Codex CLI** — OpenAI
- **Gemini CLI** — Google
- **opencode** — Open source
- **Cursor** — cursor-agent CLI
- **Aider** — Open source

The prompts

These are the exact prompts the buttons above use. Copy and adapt them to your case — for example, name your language, framework, or whether you integrate as a merchant or as a payment provider.

For browser assistants (Claude, ChatGPT, Perplexity, Grok, Le Chat, Gemini):

```
I'm integrating Pago46 cash payments (pay-in and pay-out) for Latin America.
Read the documentation index at https://docs.pago46.io/en/llms.txt — it links to
a Markdown copy of every doc page — then help me plan and implement the
integration. Start by fetching that URL.
```

For terminal coding agents (Claude Code, Codex, Gemini CLI, OpenCode, Cursor, Aider):

```
Read https://docs.pago46.io/en/llms.txt and the pages it links, then help me
integrate Pago46 cash pay-in and pay-out.
```

> **COPY ANY PAGE INTO YOUR ASSISTANT**
>
> Every page in these docs has a **Copy Page** button (top right) that copies its Markdown, or opens it directly in ChatGPT or Claude. Use it when you have a focused question about one specific page.

How it works

- `llms.txt` — an index at the site root that lists every page with its title and a link to the Markdown version. It's the ideal starting point for an assistant: open it here.
- **Per-page Markdown** — every page also exists as `.md` (for example `authentication.md`), stripped of the site chrome, so the model reads only the content.
- **Always fresh** — these files are generated on every docs release, so the assistant reads the current version, not a stale copy.

For the full endpoint and payload reference, also point your assistant at the OpenAPI reference.

Handling Common Errors

Error response format

The error response format looks like this:

```
{
  "type": "validation_error",
  "errors": [
    {
      "code": "required",
      "detail": "This field is required.",
      "attr": "name"
    }
  ]
}
```

- `type`: can be `validation_error`, `client_error`, or `server_error`.
- `code`: A short string describing the error. You can use it to adapt your integration's behavior.
- `detail`: user-friendly text describing the error. Usually in English, but may be localized.
- `attr`: the field name when the error applies to a specific field (for example, in a `validation_error`), and `null` for non-validation errors when not applicable.

Common errors

Some errors may appear on most of our endpoints. The following sections describe the most common ones. For more details, refer to the linked resources in OpenAPI.

403 Forbidden

These errors are returned with status code 403 whenever authentication fails or a request is made to an endpoint without authentication information. These are the two possible errors:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "authentication_failed",
            "detail": "Incorrect authentication credentials.",
            "attr": null
        }
    ]
}
```

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "not_authenticated",
            "detail": "Authentication credentials were not provided.",
            "attr": null
        }
    ]
}
```

405 Method Not Allowed

This is returned when an endpoint is called with an unexpected HTTP method. For example, if updating a user requires a POST request and a PATCH is issued instead, this error is returned. Here's what it looks like:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "method_not_allowed",
            "detail": "Method \"PATCH\" not allowed.",
            "attr": null
        }
    ]
}
```

406 Not Acceptable

This is returned if the `Accept` header is sent with a value other than `application/json`. Here's how the response would look:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "not_acceptable",
            "detail": "Could not satisfy the request Accept header.",
            "attr": null
        }
    ]
}
```

415 Unsupported Media Type

This is returned when the request content type is not JSON. Here's how the response would look:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "unsupported_media_type",
            "detail": "Unsupported media type \"application/xml\" in request.",
            "attr": null
        }
    ]
}
```

500 Internal Server Error

This is returned when the API server encounters an unexpected error. Here's how the response would look:

```
{
    "type": "server_error",
    "errors": [
        {
            "code": "error",
            "detail": "A server error occurred.",
            "attr": null
        }
    ]
}
```

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

Merchant Integration

As a merchant on Pago46, you access a hybrid network of physical points where your users can pay or receive cash. You manage everything digitally through our API.

Available services

Pay-In (Collections)

Allow your users to pay with cash at any point in the network.

Use cases: Wallet top-ups, subscriptions, e-commerce, services.

View Pay-In documentation

Pay-Out (Disbursements)

Allow your users to withdraw cash or receive payments at physical points.

Use cases: Balance withdrawals, payments to workers, remittances, refunds.

View Pay-Out documentation

Foreign Currency

Operations with currency conversion.

View Foreign Currency documentation

Checkout

Embed the web Checkout application in an iframe for Pay-In and Pay-Out flows.

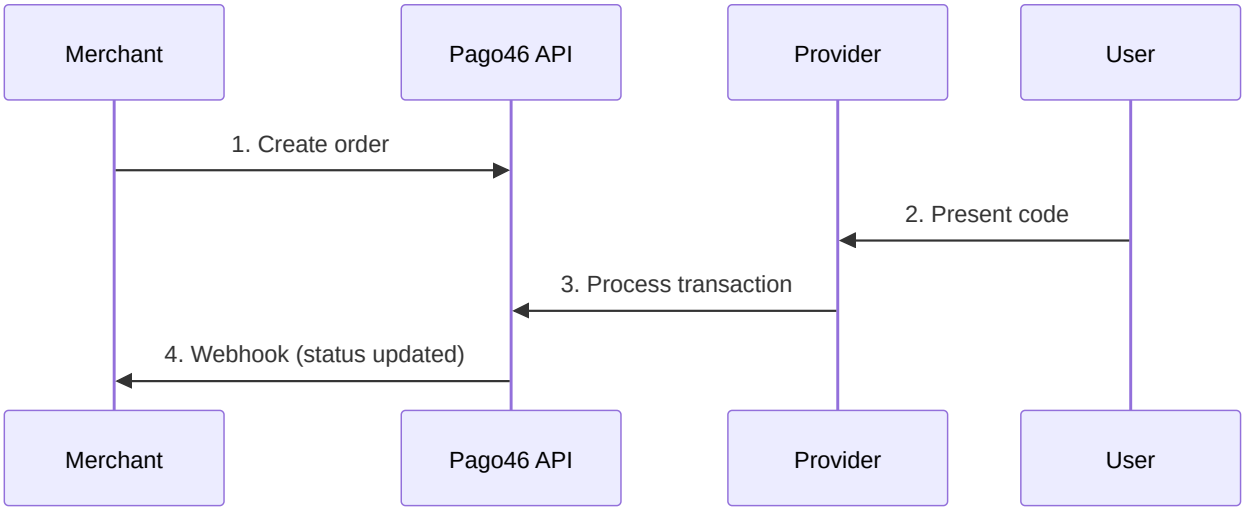
View Checkout documentation

> **PAGO46 CHECKOUT**
>
> Orders contain the `redirect_url` field, which is the Checkout link your user uses to continue the flow.
>
> - Sandbox: `https://checkout.dev.pago46.io`
> - Production: `https://checkout.prd.pago46.io`
> - Per-order format: `https://checkout.{dev|prd}.pago46.io/{UUID}`
>
> You can also open the Checkout home page (without a UUID) and enter the order id manually for iframe integration testing.

View embedded content online: https://drive.google.com/file/d/19XG84b29aGuBPA_VWpPrI1J4QknkosVe/preview

Integration flow

The architecture is server-to-server with HMAC SHA-256 authentication.



Integration steps

1. **Get your credentials** - Receive your `Merchant-Key` and `Merchant-Secret`
2. **Implement HMAC authentication** - View guide
3. **Create orders** - Use `POST /merchants/orders/pay-in` or `/pay-out`
4. **Receive webhooks** - Configure your HTTPS endpoint for notifications

> **FUNDAMENTAL REQUIREMENT**
>
> All requests require HMAC authentication. Review the authentication documentation before starting.

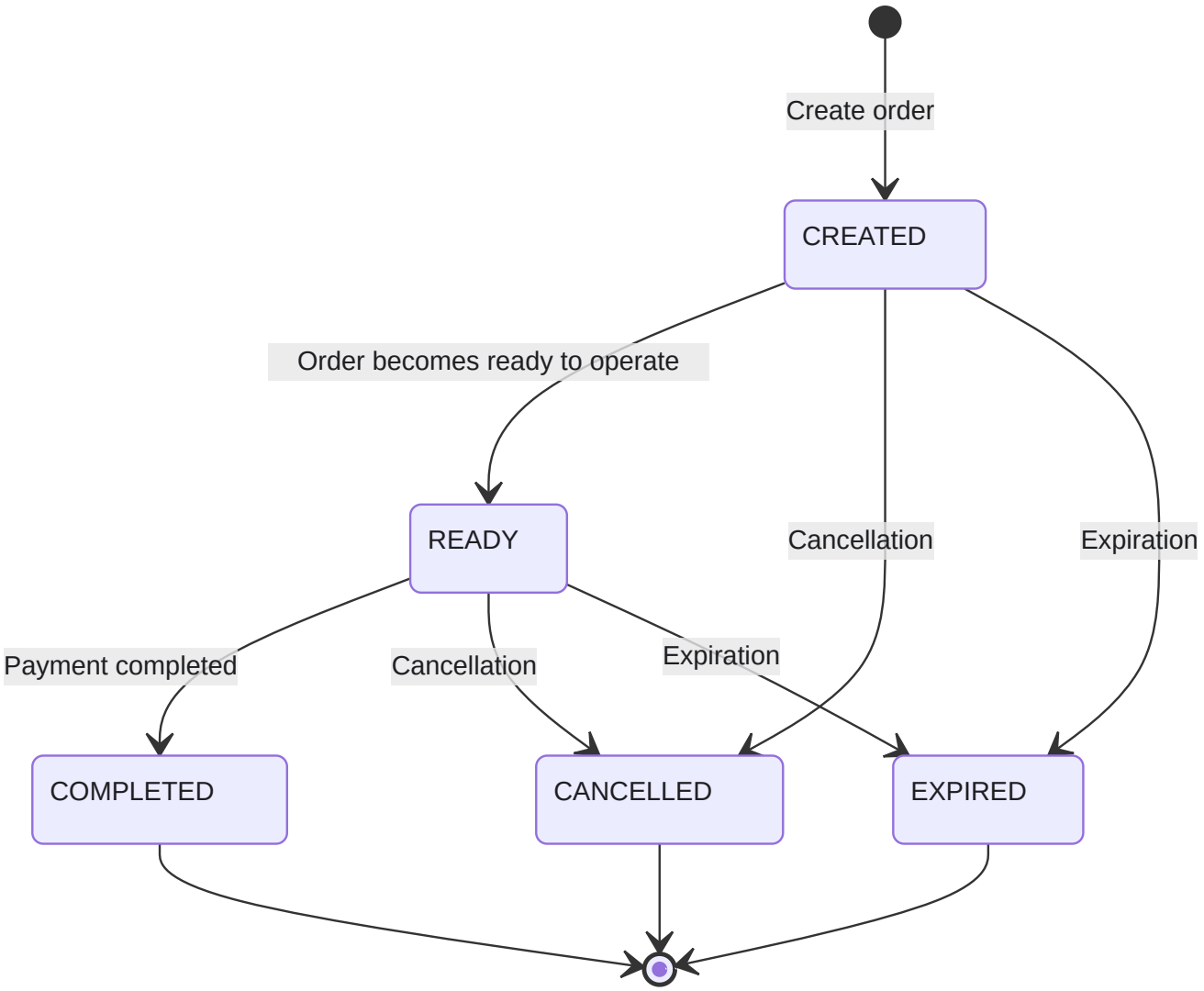
Environments

Environment	URL	Usage
Sandbox	`api.dev.pago46.io`	Development and testing
Production	`api.prd.pago46.io`	Real operations

→ View environment details

Order lifecycle

As a merchant, your integration reacts to the order's **final states** (`COMPLETED`, `CANCELLED`, `EXPIRED`), which we report via webhooks. The intermediate processing (a provider taking the order and collecting or handing out the cash) is resolved internally and requires no action on your side.



FOREIGN-CURRENCY ORDER NOTIFICATIONS

Merchants receive webhooks when an order reaches a final state. For **Foreign-Currency Pay-Out**, you also receive two earlier notifications: a `CREATED` update that adds the transaction you must sign, and a `READY` webhook when we confirm the on-chain funds reached Pago46 and the order is ready to be disbursed in cash to the end user.

WHO CAN CANCEL?

Merchants **cannot** cancel orders. In **Pay-In** the cancellation is performed by the end user or the Pago46 team; in **Pay-Out** (local and foreign currency) **only the Pago46 team** can cancel. We prefer unattended orders to expire naturally.

You receive an HTTP POST notification at `notify_url` when the order reaches a final state. The Foreign-Currency Pay-Out flow also sends the `CREATED` and `READY` notifications described above, so you may receive more than one notification with the same `status` in that flow.

Coverage

Pago46 operates in multiple countries across Latin America. View complete list.

Payments (Pay-In)

This module allows Merchants to create payment orders (pay-in) to receive funds from their users or customers. Created orders become available for processing by Payment Providers integrated into the Pago46 network.

BASE ENDPOINT

All routes described below are relative to the API base URL: `/api/v1`

PAGO46 CHECKOUT

Orders contain the `redirect_url` field, which is the Checkout link used to continue the user flow.

- Sandbox: `https://checkout.dev.pago46.io`
- Production: `https://checkout.prd.pago46.io`
- Per-order format: `https://checkout.{dev|prd}.pago46.io/{UUID}`

You can also open the base URL without a UUID and manually enter the order ID for iframe integration testing.

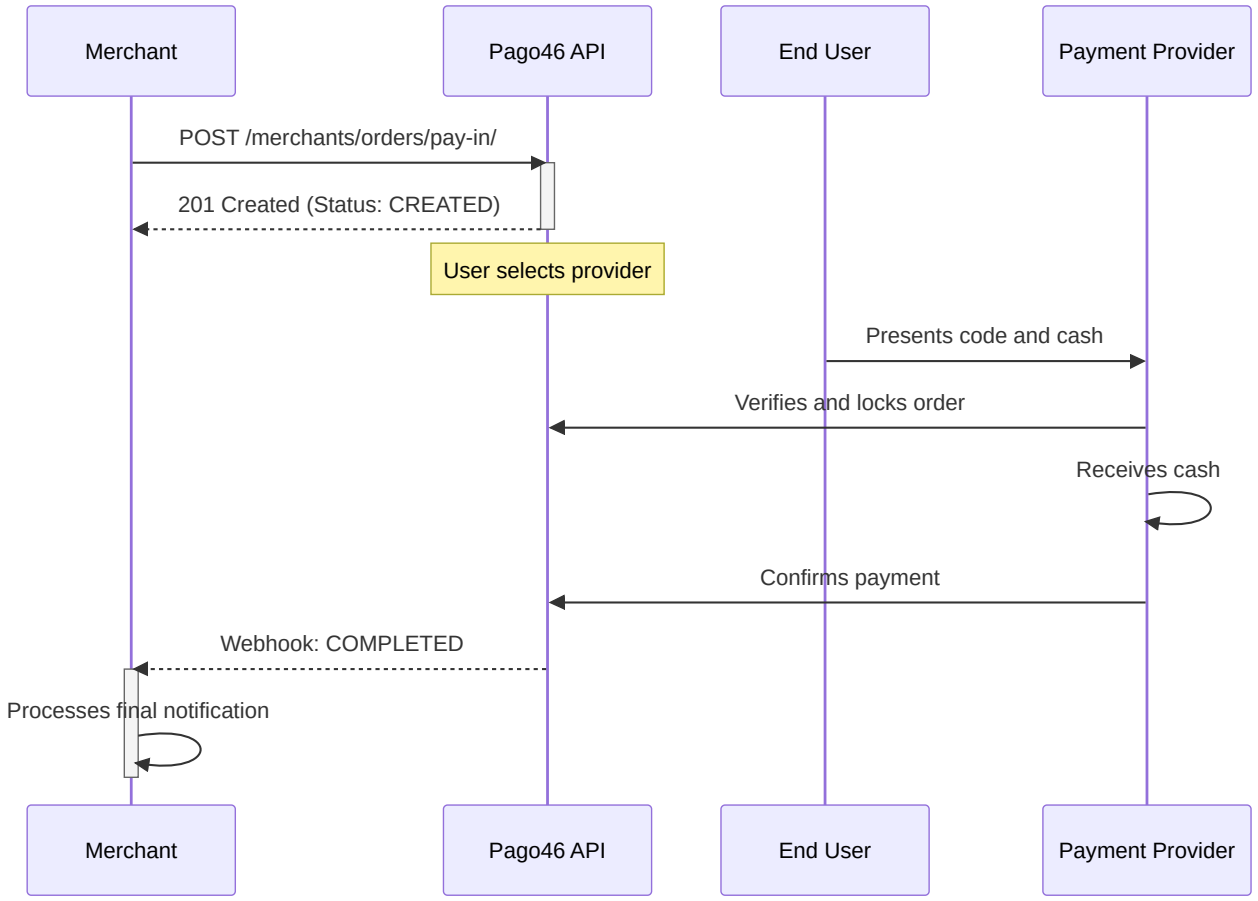
> View embedded content online: https://drive.google.com/file/d/19XG84b29aGuBPA_VWpPrI1J4QknkosVe/preview

More details in Checkout.

General flow

The payment process for merchants is divided into three main stages:

1. **Order creation:** The merchant creates a payment order specifying the amount, country, payer data, and notification URLs.
2. **Processing:** The order is processed by a Payment Provider in the network, who receives cash from the end user.
3. **Notifications (webhooks):** The merchant receives the order's final status through the configured webhook.



Prerequisites

Before starting, make sure you have:

- **API credentials:** Your `Merchant-Key` and `Merchant-Secret` provided by Pago46.
- **HMAC authentication:** Familiarize yourself with the authentication scheme described in the Authentication section.
- **Webhook endpoint:** A public HTTPS URL where you'll receive status change notifications.

> **SECURITY**
>
> All requests must include HMAC authentication headers: `Merchant-Key`, `Message-Date`, and `Message-Hash`.

---

1. Create a payment order

To initiate a collection, you must create an order providing amount information, country, payer, and notification configuration.

Endpoint: `POST /merchants/orders/pay-in/`

Request parameters

Required fields

Field	Type	Description
`order_type`	String	Order type: `LocalCurrencyOrder`
`country`	String	Country ISO code (e.g., `MX`, `CL`, `AR`)
`price`	Decimal	Payment amount (format: `"1500.00"`)
`price_currency`	String	Currency ISO code (e.g., `MXN`, `CLP`, `ARS`)
`description`	String	Transaction description
`merchant_order_id`	String	Unique ID from your system (max 127 characters)
`notify_url`	String (URL)	URL to receive status change webhooks
`return_url`	String (URL)	Return URL for the user
`expiry`	DateTime	Expiration date and time (ISO 8601 format)

Optional fields

Field	Type	Description
`consumer_email`	String	Payer email
`consumer_phone_number`	String	Payer phone (max 128 characters)

CONTACT DETAILS

For pay-in, if you send contact data, you must send both fields, only `consumer_email`, or neither. `consumer_phone_number` without email is not supported. Phone-only support will be available soon.

COUNTRY AND CURRENCY LIMITS

Your account must be enabled for the `country` + `price_currency` combination you send; otherwise you get a `400` with the detail `No matching configuration found for merchant, country, and currency`.

The `price` must also fall within the **minimum and maximum** amounts configured for your account in that country and currency. Out of range, the API returns `400` with `Value is too low.` or `Value is too high.` on the `price` field. See the available countries and currencies; for your limits, check with your account executive.

Request example

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/orders/pay-in/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "order_type": "LocalCurrencyOrder",
    "country": "MX",
    "price": "1500.00",
    "price_currency": "MXN",
    "description": "Subscription payment - User ABC123",
    "merchant_order_id": "ORDER-2024-001234",
```

```
    "notify_url": "https://your-merchant.com/webhooks/pago46",
    "return_url": "https://your-merchant.com/payment/return",
    "consumer_email": "user@example.com",
    "consumer_phone_number": "+525512345678",
    "expiry": "2026-06-27T23:59:59Z"
  }'
```

Successful response (201 Created)

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Subscription payment - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/payment/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": null
}
```

SAVE THE ID

Save the `id` of the order returned in the response. You'll need it to query the order status later.

---

2. Query an order

You can query the current status of an order at any time using its ID.

Endpoint: `GET /merchants/orders/pay-in/{id}/`

Parameters

Parameter	Location	Description
`id`	Path	Order UUID

Request example

```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-in/123e4567-e89b-12d3-a456-426614174000/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Response (200 OK)

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
```

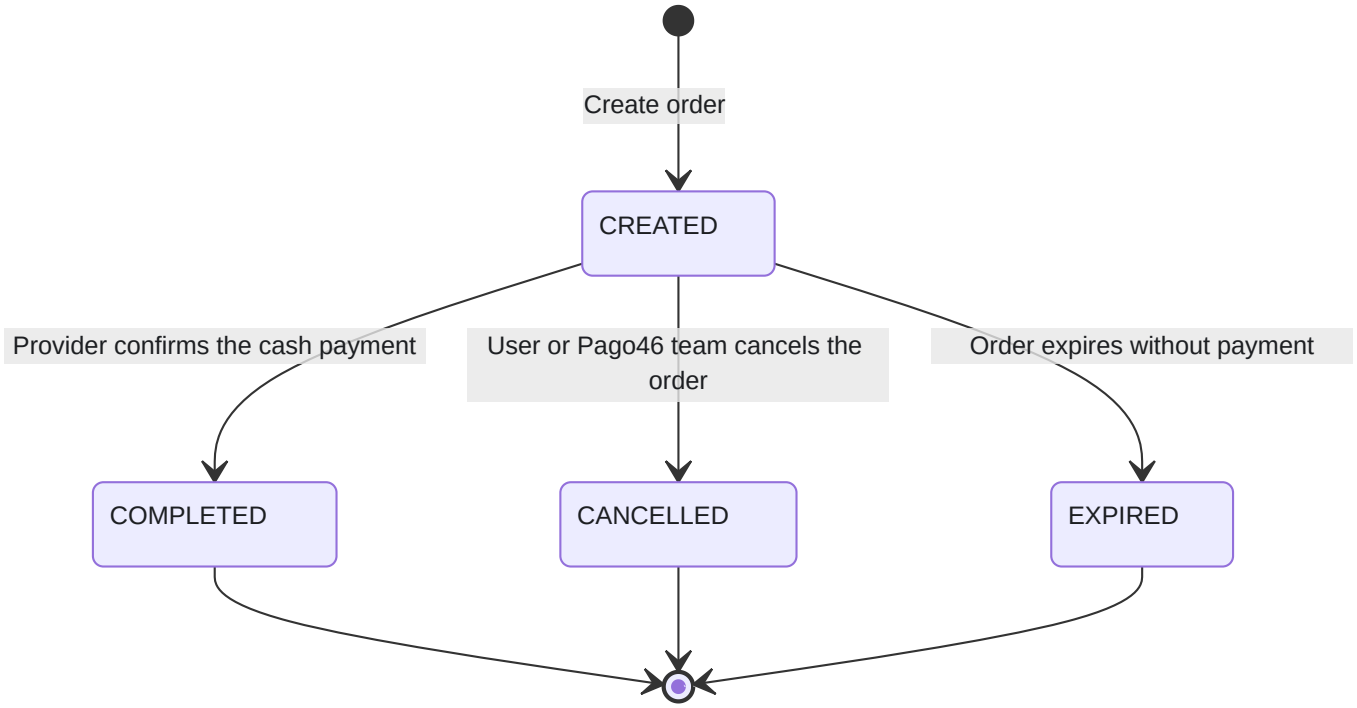
```
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Subscription payment - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/payment/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": null
}
```

Order states

As a merchant you only need to react to the **final states** (`COMPLETED`, `CANCELLED`, `EXPIRED`). Intermediate states are resolved internally and, in Pay-In, are not notified.

State	Notified via webhook?	What does it mean for the merchant?
`CREATED`	Order creation response	The order has been registered and is pending processing
`READY`	No (internal in Pay-In)	The order can now be taken by a provider
`PAYMENT_STARTED`	No (internal)	Intermediate step: the user is making the physical payment at the point of sale
`COMPLETED`	Yes — final state	The payment completed successfully. The provider received the cash from the user
`CANCELLED`	Yes — final state	The order was cancelled (by the user or the Pago46 team) and will not be processed
`EXPIRED`	Yes — final state	The order expired without payment and will not be processed

State transition diagram



Webhooks (notifications)

When the order reaches a final state (`COMPLETED`, `CANCELLED`, or `EXPIRED`), Pago46 sends an HTTP POST notification to the URL specified in `notify_url`.

Webhook structure

The webhook will be sent with HMAC authentication. You must verify the signature to ensure the notification comes from Pago46.

Webhook headers

```
POST /webhooks/pago46 HTTP/1.1
Host: your-merchant.com
Content-Type: application/json
Merchant-Key: <YOUR_MERCHANT_KEY>
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Webhook payload

State: COMPLETED

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Subscription payment - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "COMPLETED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
```

```
  "return_url": "https://your-merchant.com/payment/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": "2026-06-25T14:30:00Z"
}
```

State: CANCELLED

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Subscription payment - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CANCELLED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/payment/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": null
}
```

Webhook verification

It's **critical** that you verify the authenticity of each received webhook to prevent processing fraudulent notifications.

Python verification example

```python
import hmac
import hashlib
import json
from flask import Flask, request, jsonify

app = Flask(__name__)

# Your Merchant Secret (obtained from Pago46)
MERCHANT_SECRET = "your_merchant_secret_here"

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    # 1. Extract headers
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')

    # 2. Get raw body
    body_str = request.get_data(as_text=True)

    # 3. Build string to sign
    # Format: MERCHANT_KEY:MESSAGE_DATE:METHOD:PATH:BODY
    method = request.method  # "POST"
    path = request.path      # "/webhooks/pago46"
    string_to_sign = f"{merchant_key}:{message_date}:{method}:{path}:{body_str}"

    # 4. Calculate HMAC
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    # 5. Verify
    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # 6. Process notification
```

```
    order_data = json.loads(body_str)
    order_id = order_data.get('id')
    order_status = order_data.get('status')
    merchant_order_id = order_data.get('merchant_order_id')

    print(f"Order {merchant_order_id} ({order_id}) changed to status: {order_status}")

    # Update your database
    if order_status == 'COMPLETED':
        # Mark as completed
        paid_at = order_data.get('paid')
        print(f"Payment completed on: {paid_at}")
        # Activate service, deliver product, etc.
    elif order_status == 'CANCELLED':
        # Mark as cancelled
        print("Payment cancelled")

    # 7. Respond with 200 OK
    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    app.run(port=5000)
```

Node.js verification example

```
const express = require('express');
const crypto = require('crypto');
const app = express();

const MERCHANT_SECRET = 'your_merchant_secret_here';

app.post('/webhooks/pago46', express.text({ type: '*/*' }), (req, res) => {
  // 1. Extract headers
  const merchantKey = req.headers['merchant-key'];
  const messageDate = req.headers['message-date'];
  const receivedHash = req.headers['message-hash'];

  // 2. Body as string
  const bodyStr = req.body;

  // 3. Build string to sign
  const method = req.method;
  const path = req.path;
  const stringToSign = `${merchantKey}:${messageDate}:${method}:${path}:${bodyStr}`;

  // 4. Calculate HMAC
  const calculatedHash = crypto
    .createHmac('sha256', MERCHANT_SECRET)
    .update(stringToSign)
    .digest('hex');

  // 5. Verify
  if (calculatedHash !== receivedHash) {
    return res.status(403).json({ error: 'Invalid signature' });
  }

  // 6. Process notification
  const orderData = JSON.parse(bodyStr);
  const { id, status, merchant_order_id, paid } = orderData;

  console.log(`Order ${merchant_order_id} (${id}) changed to status: ${status}`);

  if (status === 'COMPLETED') {
    console.log(`Payment completed on: ${paid}`);
    // Activate service, deliver product, etc.
  } else if (status === 'CANCELLED') {
    console.log('Payment cancelled');
  }

  // 7. Respond
  res.status(200).json({ status: 'received' });
});

app.listen(5000, () => {
  console.log('Webhook server listening on port 5000');
});
```

WEBHOOK RESPONSE

You must respond with an HTTP `200` or `201` status code to confirm reception. If you don't respond successfully, we will retry sending the notification.

We retry on any `5XX`, `408`, or `429` response, and also on timeouts or connection errors (no response received), using exponential backoff. In **production**, up to 600 attempts over roughly 4 days, capped at 10 minutes between attempts. In **sandbox**, only 3 attempts over a few seconds, following the same backoff pattern.

If the notification for an already-completed order fails in a non-retryable way (for example, retries are exhausted), we will reach out to you through one of your already-established communication channels to help resolve any issues your users may be experiencing with their order.

Common errors

See Handling common errors for the standard error format and examples covering limit validations, required fields, and other business errors.

Field validation

If you send invalid or incomplete data, you will receive a `400 Bad Request` error with specific details:


```
{
    "type": "validation_error",
    "errors": [
        {
            "code": "invalid",
            "detail": "No matching configuration found for merchant, country, and currency.",
            "attr": "merchant_country_order_setting"
        },
        {
            "code": "required",
            "detail": "This field is required.",
            "attr": "country"
        },
        {
            "code": "required",
            "detail": "This field is required.",
            "attr": "price"
        },
        {
            "code": "required",
            "detail": "This field is required.",
            "attr": "price_currency"
        }
    ]
}
```


HTTP error codes

HTTP Code	Description	Solution
`400 Bad Request`	Invalid data or missing fields	Verify that all required fields are present and correctly formatted
`403 Forbidden`	Authentication failed	Check your credentials and HMAC signature
`404 Not Found`	Order not found	Ensure the order ID is correct
`422 Unprocessable Entity`	Business logic error	Review the specific error message

Best practices

1. Order identifier

Assign a unique `merchant_order_id` per operation. The API enforces a uniqueness constraint on `(merchant_order_id, merchant)`: creating another order with an already-used `merchant_order_id` is rejected rather than duplicated. Use it to reconcile each order with your system. If you're unsure whether a create succeeded, **look the order up** instead of retrying the `POST`.

2. Webhook handling

- **Process webhooks asynchronously:** Don't block the HTTP response while processing business logic.
- **Implement retries:** If your webhook server is down, Pago46 will retry sending.
- **Always validate HMAC signature:** Never trust webhooks without verifying their authenticity.

3. Order expiration

Set `expiry` between 24 and 72 hours after creating the order. This range reduces the risk of a user keeping a screenshot of the code and attempting to pay after the operation should no longer be valid. Avoid orders that remain valid for several days or weeks.

4. Order monitoring

- Query orders that do not reach a final state within the expected time.
- Implement alerts to detect orders that remain pending for too long.
- Check the Service Status page when unusual errors appear, to confirm active incidents or maintenance.

5. Notification URLs

- Use HTTPS URLs for `notify_url`.
- Ensure the endpoint is always available.
- Implement logging for debugging.

Complete integration example

This complete Python example shows how to create an order and handle webhooks:

```
import hmac
import hashlib
import time
import requests
import json
from flask import Flask, request, jsonify

# Configuration
API_BASE_URL = "https://api.dev.pago46.io"
MERCHANT_KEY = "your_merchant_key"
MERCHANT_SECRET = "your_merchant_secret"

app = Flask(__name__)

def generate_hmac(method, path, body_dict=None):
    """Generates HMAC signature for authentication"""
    timestamp = str(time.time())
    body_str = json.dumps(body_dict) if body_dict else ""

    string_to_sign = f"{MERCHANT_KEY}:{timestamp}:{method}:{path}:{body_str}"
```

```
    signature = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    return {
        "Merchant-Key": MERCHANT_KEY,
        "Message-Date": timestamp,
        "Message-Hash": signature,
        "Content-Type": "application/json"
    }

def create_payin_order(amount, user_email, user_phone, merchant_order_id):
    """Creates a payment order"""
    path = "/api/v1/merchants/orders/pay-in/"

    order_data = {
        "order_type": "LocalCurrencyOrder",
        "country": "MX",
        "price": str(amount),
        "price_currency": "MXN",
        "description": f"User payment {user_email}",
        "merchant_order_id": merchant_order_id,
        "notify_url": "https://your-merchant.com/webhooks/pago46",
        "return_url": "https://your-merchant.com/payment/return",
        "consumer_email": user_email,
        "consumer_phone_number": user_phone,
        "expiry": "2026-06-27T23:59:59Z"
    }

    headers = generate_hmac("POST", path, order_data)

    response = requests.post(
        f"{API_BASE_URL}{path}",
        headers=headers,
        json=order_data
    )

    if response.status_code == 201:
        order = response.json()
        print(f"Order created successfully: {order['id']}")
        return order
    else:
        print(f"Error creating order: {response.status_code}")
        print(response.text)
        return None

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    """Handles Pago46 webhooks"""
    # Verify HMAC signature
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')
    body_str = request.get_data(as_text=True)

    string_to_sign = f"{merchant_key}:{message_date}:{request.method}:{request.path}:{body_str}"
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # Process webhook
    order = json.loads(body_str)

    print(f"Webhook received for order: {order['merchant_order_id']}")
    print(f"   Status: {order['status']}")

    # Merchant webhooks only contain final states
    if order['status'] == 'COMPLETED':
        print(f"   Payment completed on {order['paid']}")
        # Activate service, deliver product, etc.
    elif order['status'] == 'CANCELLED':
        print("   Payment cancelled")
    elif order['status'] == 'EXPIRED':
```

```
        print("   Order expired")

    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    # Example: Create a payment order
    order = create_payin_order(
        amount=1500.00,
        user_email="user@example.com",
        user_phone="+525512345678",
        merchant_order_id="ORDER-2024-" + str(int(time.time()))
    )

    # Start webhook server
    print("\nStarting webhook server...")
    app.run(port=5000)
```

Next steps

- **Authentication:** Review the HMAC authentication guide to understand the security mechanism in detail.
- **Environments:** Familiarize yourself with development and production environments.

SUPPORT

If you have questions or need help with your integration, contact your account executive or email contacto@pago46.com.

Withdrawals (Pay-Out)

This module allows Merchants to create withdrawal orders (pay-out) to disburse funds to their users or customers. Created orders become available for processing by Payment Providers integrated into the Pago46 network.

BASE ENDPOINT

All routes described below are relative to the API base URL: `/api/v1`

PAGO46 CHECKOUT

Orders contain the `redirect_url` field, which is the Checkout link used to continue the user flow.

- Sandbox: `https://checkout.dev.pago46.io`
- Production: `https://checkout.prd.pago46.io`
- Per-order format: `https://checkout.{dev|prd}.pago46.io/{UUID}`

You can also open the base URL without a UUID and manually enter the order ID for iframe integration testing.

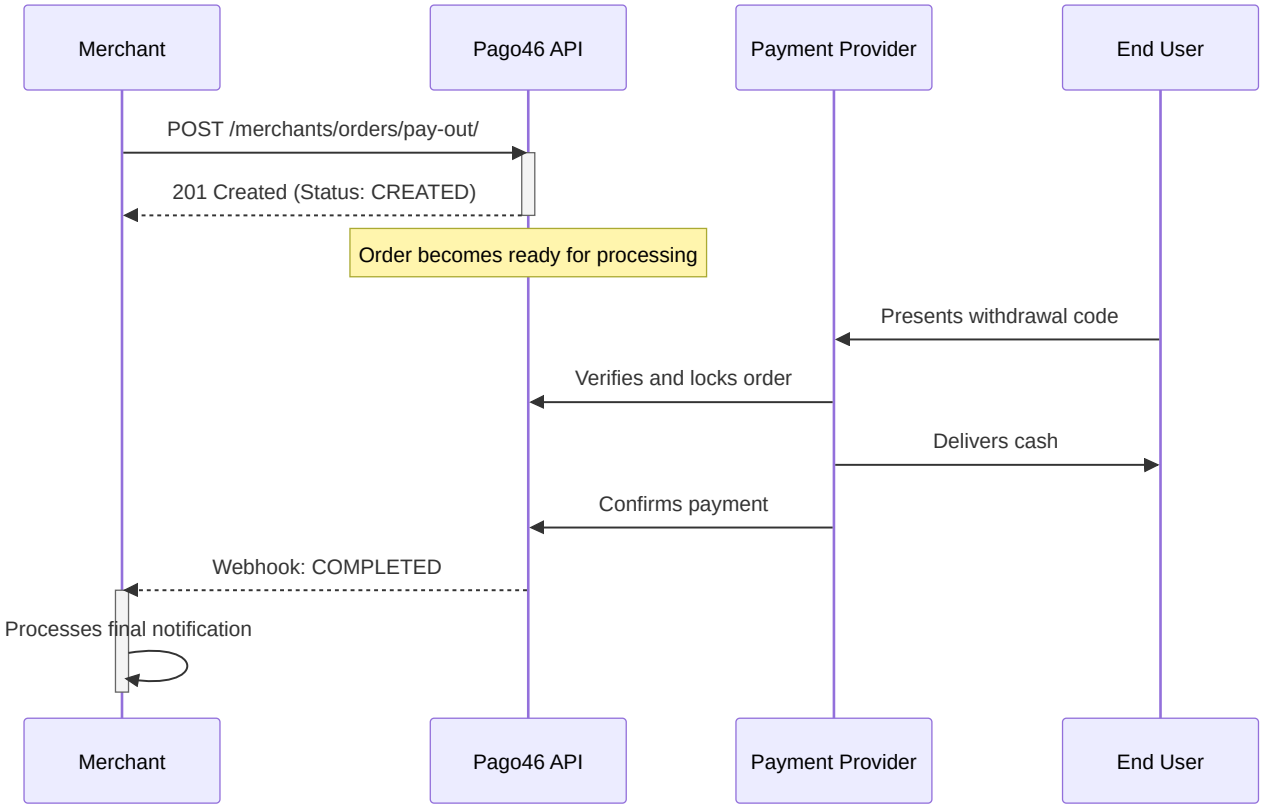
> View embedded content online: https://drive.google.com/file/d/1iSXarQQocBATZjLsEGcDEajYLfvm6Aon/preview

More details in Checkout.

General flow

The withdrawal process for merchants is divided into three main stages:

1. **Order creation:** The merchant creates a withdrawal order specifying the amount, country, beneficiary data, and notification URLs.
2. **Processing:** The order is processed by a Payment Provider in the network, who delivers cash to the end user.
3. **Notifications (webhooks):** The merchant receives the order's final status through the configured webhook.



Prerequisites

Before starting, make sure you have:

- **API credentials:** Your `Merchant-Key` and `Merchant-Secret` provided by Pago46.
- **HMAC authentication:** Familiarize yourself with the authentication scheme described in the Authentication section.
- **Webhook endpoint:** A public HTTPS URL where you'll receive status change notifications.

> **SECURITY**
>
> All requests must include HMAC authentication headers: `Merchant-Key`, `Message-Date`, and `Message-Hash`.

1. Create a withdrawal order

To initiate a withdrawal, you must create an order providing amount information, country, beneficiary, and notification configuration.

Endpoint: `POST /merchants/orders/pay-out/`

Request parameters

Required fields

Field	Type	Description
`order_type`	String	Order type: `LocalCurrencyOrder`

Field	Type	Description
`country`	String	Country ISO code (e.g., `MX`, `CL`, `AR`)
`price`	Decimal	Withdrawal amount (format: `"1500.00"`)
`price_currency`	String	Currency ISO code (e.g., `MXN`, `CLP`, `ARS`)
`description`	String	Transaction description
`merchant_order_id`	String	Unique ID from your system (max 127 characters)
`notify_url`	String (URL)	URL to receive status change webhooks
`return_url`	String (URL)	Return URL for the user
`expiry`	DateTime	Expiration date and time (ISO 8601 format)

Optional fields

Field	Type	Description
`consumer_email`	String	Beneficiary email
`consumer_phone_number`	String	Beneficiary phone (max 128 characters)

BENEFICIARY CONTACT

For pay-out you must send **at least one** of `consumer_email` or `consumer_phone_number`.

FOREIGN CURRENCY WITHDRAWALS

For international withdrawals or currency conversion, see the Foreign Currency Withdrawals documentation.

COUNTRY AND CURRENCY LIMITS

Your account must be enabled for the `country` + `price_currency` combination you send; otherwise you get a `400` with the detail `No matching configuration found for merchant, country, and currency`.

The `price` must also fall within the **minimum and maximum** amounts configured for your account in that country and currency. Out of range, the API returns `400` with `Value is too low.` or `Value is too high.` on the `price` field. See the available countries and currencies; for your limits, check with your account executive.

Request example

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "order_type": "LocalCurrencyOrder",
    "country": "MX",
    "price": "1500.00",
    "price_currency": "MXN",
    "description": "Balance withdrawal - User ABC123",
    "merchant_order_id": "ORDER-2024-001234",
    "notify_url": "https://your-merchant.com/webhooks/pago46",
    "return_url": "https://your-merchant.com/withdrawal/return",
    "consumer_email": "user@example.com",
```

```
    "consumer_phone_number": "+525512345678",
    "expiry": "2026-06-27T23:59:59Z"
  }'
```

Successful response (201 Created)

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Balance withdrawal - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/withdrawal/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": null
}
```

SAVE THE ID

Save the `id` of the order returned in the response. You'll need it to query the order status later.

---

2. Query an order

You can query the current status of an order at any time using its ID.

Endpoint: `GET /merchants/orders/pay-out/{id}/`

Parameters

Parameter	Location	Description
`id`	Path	Order UUID

Request example

```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/123e4567-e89b-12d3-a456-426614174000/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Response (200 OK)

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Balance withdrawal - User ABC123",
```

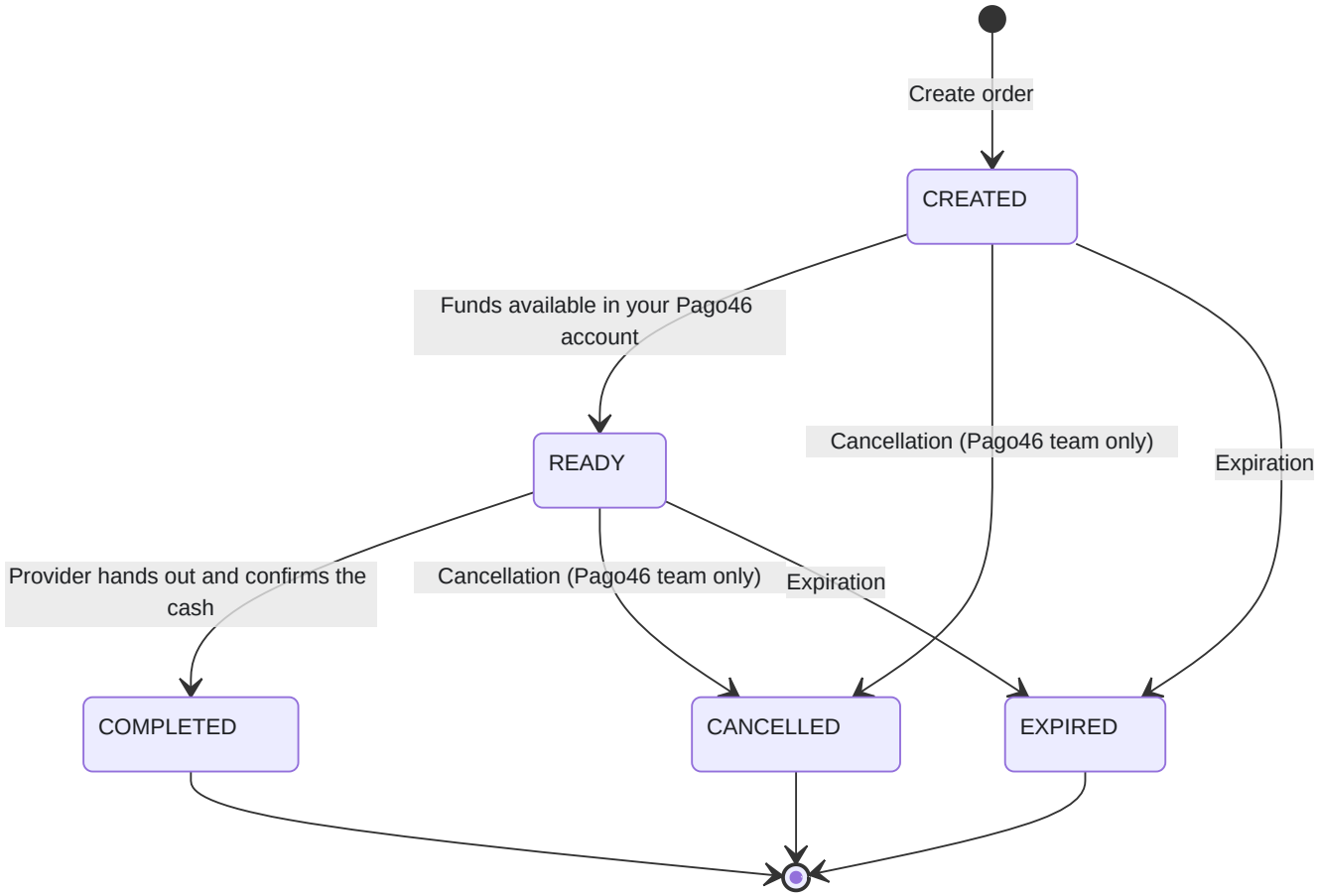
```
  "merchant_order_id": "ORDER-2024-001234",
  "status": "READY",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/withdrawal/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": null
}
```

Order states

As a merchant, you only receive webhooks for final states (`COMPLETED` , `CANCELLED` , and `EXPIRED`). Intermediate states are handled internally.

State	Notified via webhook?	What does it mean for the merchant?
`CREATED`	Order creation response	The order has been registered; pending available funds
`READY`	No (internal)	Funds are available in your Pago46 account; the order can proceed
`PAYMENT_STARTED`	No (internal)	Intermediate step: a provider is handing the cash to the beneficiary
`COMPLETED`	Yes — final state	The withdrawal completed successfully. The beneficiary received the cash
`CANCELLED`	Yes — final state	The order was cancelled by the Pago46 team (neither the merchant nor the user can cancel) and will not be processed
`EXPIRED`	Yes — final state	The order expired and will not be processed

State transition diagram



Webhooks (notifications)

When the order reaches a final state (`COMPLETED` , `CANCELLED` , or `EXPIRED`), Pago46 sends an HTTP POST notification to the URL specified in `notify_url` .

Webhook structure

The webhook will be sent with HMAC authentication. You must verify the signature to ensure the notification comes from Pago46.

Webhook headers

```
POST /webhooks/pago46 HTTP/1.1
Host: your-merchant.com
Content-Type: application/json
Merchant-Key: <YOUR_MERCHANT_KEY>
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Webhook payload

State: COMPLETED

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
```

```
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Balance withdrawal - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "COMPLETED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/withdrawal/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": "2026-06-25T14:30:00Z"
}
```

State: CANCELLED

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Balance withdrawal - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CANCELLED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://your-merchant.com/withdrawal/return",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2026-06-27T23:59:59Z",
  "paid": null
}
```

Webhook verification

It's **critical** that you verify the authenticity of each received webhook to prevent processing fraudulent notifications.

Python verification example

```python
import hmac
import hashlib
import json
from flask import Flask, request, jsonify

app = Flask(__name__)

# Your Merchant Secret (obtained from Pago46)
MERCHANT_SECRET = "your_merchant_secret_here"

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    # 1. Extract headers
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')

    # 2. Get raw body
    body_str = request.get_data(as_text=True)

    # 3. Build string to sign
    # Format: MERCHANT_KEY:MESSAGE_DATE:METHOD:PATH:BODY
    method = request.method  # "POST"
    path = request.path      # "/webhooks/pago46"
    string_to_sign = f"{merchant_key}:{message_date}:{method}:{path}:{body_str}"

    # 4. Calculate HMAC
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
```

```
    ).hexdigest()

    # 5. Verify
    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # 6. Process notification
    order_data = json.loads(body_str)
    order_id = order_data.get('id')
    order_status = order_data.get('status')
    merchant_order_id = order_data.get('merchant_order_id')

    print(f"Order {merchant_order_id} ({order_id}) changed to status: {order_status}")

    # Update your database
    if order_status == 'COMPLETED':
        # Mark as completed
        paid_at = order_data.get('paid')
        print(f"Withdrawal completed on: {paid_at}")
    elif order_status == 'CANCELLED':
        # Mark as cancelled
        print("Withdrawal cancelled")

    # 7. Respond with 200 OK
    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    app.run(port=5000)
```

Node.js verification example

```
const express = require('express');
const crypto = require('crypto');
const app = express();

const MERCHANT_SECRET = 'your_merchant_secret_here';

app.post('/webhooks/pago46', express.text({ type: '*/*' }), (req, res) => {
  // 1. Extract headers
  const merchantKey = req.headers['merchant-key'];
  const messageDate = req.headers['message-date'];
  const receivedHash = req.headers['message-hash'];

  // 2. Body as string
  const bodyStr = req.body;

  // 3. Build string to sign
  const method = req.method;
  const path = req.path;
  const stringToSign = `${merchantKey}:${messageDate}:${method}:${path}:${bodyStr}`;

  // 4. Calculate HMAC
  const calculatedHash = crypto
    .createHmac('sha256', MERCHANT_SECRET)
    .update(stringToSign)
    .digest('hex');

  // 5. Verify
  if (calculatedHash !== receivedHash) {
    return res.status(403).json({ error: 'Invalid signature' });
  }

  // 6. Process notification
  const orderData = JSON.parse(bodyStr);
  const { id, status, merchant_order_id, paid } = orderData;

  console.log(`Order ${merchant_order_id} (${id}) changed to status: ${status}`);

  if (status === 'COMPLETED') {
    console.log(`Withdrawal completed on: ${paid}`);
    // Update database
  } else if (status === 'CANCELLED') {
    console.log('Withdrawal cancelled');
    // Update database
  }

  // 7. Respond
  res.status(200).json({ status: 'received' });
```

```
});

app.listen(5000, () => {
  console.log('Webhook server listening on port 5000');
});
```

⚠ WEBHOOK RESPONSE

You must respond with an HTTP `200` or `201` status code to confirm reception. If you don't respond successfully, we will retry sending the notification.

We retry on any `5XX`, `408`, or `429` response, and also on timeouts or connection errors (no response received), using exponential backoff. In **production**, up to 600 attempts over roughly 4 days, capped at 10 minutes between attempts. In **sandbox**, only 3 attempts over a few seconds, following the same backoff pattern.

If the notification for an already-completed order fails in a non-retryable way (for example, retries are exhausted), we will reach out to you through one of your already-established communication channels to help resolve any issues your users may be experiencing with their order.

Common errors

See Handling common errors for the standard error format and examples covering limit validations, required fields, and other business errors.

Field validation example

If you provide invalid or incomplete data, you will receive a `400 Bad Request` error structured as shown below (see "Common Errors" for more details).

Example response for invalid or missing fields:

```
{
  "type": "validation_error",
  "errors": [
    {
      "attr": "country",
      "code": "required",
      "detail": "This field is required."
    },
    {
      "attr": "price",
      "code": "invalid",
      "detail": "A valid number is required."
    },
    {
      "attr": "expiry",
      "code": "invalid",
      "detail": "Datetime has wrong format. Use one of these formats instead: YYYY-MM-DDThh:mm[:ss[.uuuuuu]][+HH:MM|-
HH:MM|Z]."
    }
  ]
}
```

HTTP error codes

HTTP Code	Description	Solution
`400 Bad Request`	Invalid data or missing fields	Check that all required fields are provided and correctly formatted
`403 Forbidden`	Authentication failed	Check your credentials and HMAC signature
`404 Not Found`	Order not found	Verify the order ID is correct
`422 Unprocessable Entity`	Business logic error	Review the specific error message

Best practices

1. Order identifier

Assign a unique `merchant_order_id` per operation. The API enforces a uniqueness constraint on `(merchant_order_id, merchant)`: creating another order with an already-used `merchant_order_id` is rejected rather than duplicated. Use it to reconcile each order with your system. If you're unsure whether a create succeeded, **look the order up** instead of retrying the `POST`.

2. Webhook handling

- **Process webhooks asynchronously:** Don't block the HTTP response while processing business logic.
- **Implement retries:** If your webhook server is down, Pago46 will retry sending.
- **Always validate HMAC signature:** Never trust webhooks without verifying their authenticity.

3. Order expiration

Set `expiry` between 24 and 72 hours after creating the order. This range reduces the risk of a user keeping a screenshot of the code and attempting to withdraw cash after the operation should no longer be valid. Avoid orders that remain valid for several days or weeks.

4. Order monitoring

- Query orders that do not reach a final state within the expected time.
- Implement alerts to detect orders that remain pending for too long.
- Check the Service Status page when unusual errors appear, to confirm active incidents or maintenance.

5. Notification URLs

- Use HTTPS URLs for `notify_url`.
- Ensure the endpoint is always available.
- Implement logging for debugging.

Complete integration example

Below is a complete Python example showing how to create an order and handle webhooks:

```python
import hmac
import hashlib
import time
import requests
import json
from flask import Flask, request, jsonify

# Configuration
API_BASE_URL = "https://api.dev.pago46.io"
MERCHANT_KEY = "your_merchant_key"
MERCHANT_SECRET = "your_merchant_secret"

app = Flask(__name__)

def generate_hmac(method, path, body_dict=None):
    """Generates HMAC signature for authentication"""
    timestamp = str(time.time())
    body_str = json.dumps(body_dict) if body_dict else ""

    string_to_sign = f"{MERCHANT_KEY}:{timestamp}:{method}:{path}:{body_str}"

    signature = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    return {
        "Merchant-Key": MERCHANT_KEY,
        "Message-Date": timestamp,
        "Message-Hash": signature,
        "Content-Type": "application/json"
    }

def create_payout_order(amount, user_email, user_phone, merchant_order_id):
    """Creates a withdrawal order"""
    path = "/api/v1/merchants/orders/pay-out/"

    order_data = {
        "order_type": "LocalCurrencyOrder",
        "country": "MX",
        "price": str(amount),
        "price_currency": "MXN",
        "description": f"User withdrawal {user_email}",
        "merchant_order_id": merchant_order_id,
        "notify_url": "https://your-merchant.com/webhooks/pago46",
        "return_url": "https://your-merchant.com/withdrawal/return",
        "consumer_email": user_email,
        "consumer_phone_number": user_phone,
        "expiry": "2026-06-27T23:59:59Z"
    }

    headers = generate_hmac("POST", path, order_data)

    response = requests.post(
        f"{API_BASE_URL}{path}",
        headers=headers,
        json=order_data
    )

    if response.status_code == 201:
        order = response.json()
        print(f"Order created successfully: {order['id']}")
        return order
    else:
        print(f"Error creating order: {response.status_code}")
        print(response.text)
        return None

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    """Handles Pago46 webhooks"""
    # Verify HMAC signature
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')
    body_str = request.get_data(as_text=True)

    string_to_sign = f"{merchant_key}:{message_date}:{request.method}:{request.path}:{body_str}"
    calculated_hash = hmac.new(
```

```
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # Process webhook
    order = json.loads(body_str)

    print(f"Webhook received for order: {order['merchant_order_id']}")
    print(f"   Status: {order['status']}")

    # Merchant webhooks only contain final states
    if order['status'] == 'COMPLETED':
        print(f"   Withdrawal completed on {order['paid']}")
        # Update user balance, send notification, etc.
    elif order['status'] == 'CANCELLED':
        print("   Withdrawal cancelled")
        # Refund balance, notify user, etc.
    elif order['status'] == 'EXPIRED':
        print("   Order expired")

    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    # Example: Create a withdrawal order
    order = create_payout_order(
        amount=1500.00,
        user_email="user@example.com",
        user_phone="+525512345678",
        merchant_order_id="ORDER-2024-" + str(int(time.time()))
    )

    # Start webhook server
    print("\nStarting webhook server...")
    app.run(port=5000)
```

Next steps

- **Foreign currency withdrawals:** See the Foreign Currency Withdrawals documentation to learn how to create international orders.
- **Authentication:** Review the HMAC authentication guide to understand the security mechanism in detail.
- **Environments:** Familiarize yourself with development and production environments.

SUPPORT

If you have questions or need help with your integration, contact your account executive or email contacto@pago46.com.

Foreign Currency Withdrawals (Pay-Out)

This module allows you to create withdrawals (pay-out) in local currency using source funds in foreign currency (for example, USDC → MXN).

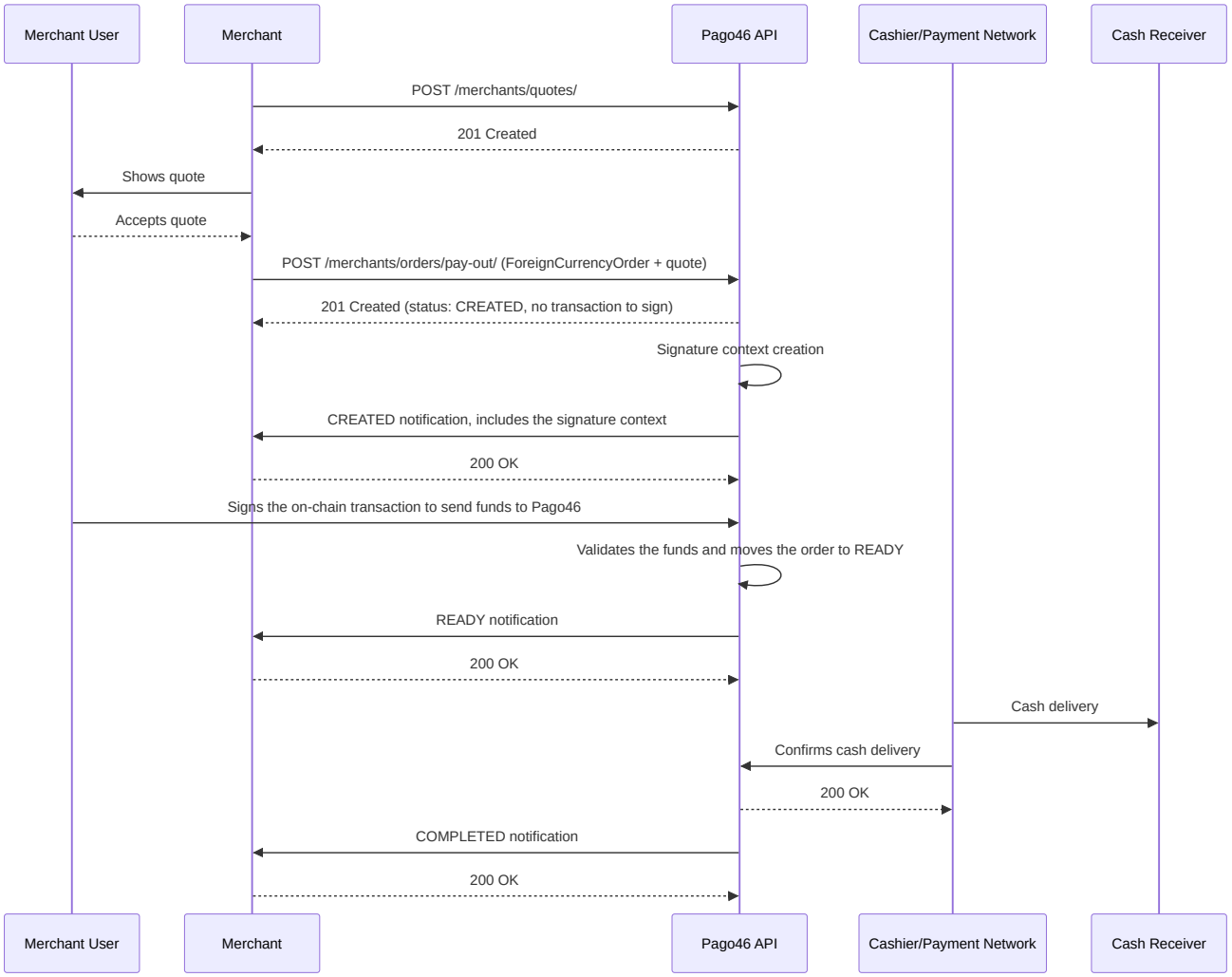
As a Merchant, your integration communicates exclusively with Pago46. Pago46 internally manages the quote, monitors the receipt of funds, and enables the cash withdrawal.

See the available pairs and countries for foreign currency, and sign every request following the Authentication guide.

> **BASE ENDPOINT**
>
> All routes described below are relative to the API base URL: `/api/v1`

General flow

1. You create a quote for an asset pair (`POST /merchants/quotes/`).
2. Your user accepts the quote in your application.
3. You create a foreign withdrawal order associated with that quote.
4. Your user sends the funds to the address indicated by Pago46.
5. If the funding arrives within the valid window, the order advances to `READY` and can continue the normal withdrawal flow.



Prerequisites

- Merchant credentials (`Merchant-Key` and `Merchant-Secret`).
- HMAC signature on every request (`Message-Date`, `Message-Hash`).
- Public `notify_url` via HTTPS for status changes.
- At least one of `consumer_email` or `consumer_phone_number` when creating the order.

Check the Authentication section for the signature.

---

1) Create a quote

Request a quote for the desired asset pair.

Endpoint: `POST /merchants/quotes/`

Request fields

Field	Type	Required	Description
`blockchain`	String	Yes	Funding network (`SOLANA`, `STELLAR`, `MONAD`).
`send`	Decimal	Yes	Amount the user will send.
`send_currency`	String	Yes	Source currency (e.g., `USDC`).
`target_country`	String	Yes	Destination country (`MX`, `CL`, etc.).
`target_currency`	String	Yes	Target local currency (e.g., `MXN`).

Request example

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/quotes/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "blockchain": "SOLANA",
    "send": "150.00",
    "send_currency": "USDC",
    "target_country": "MX",
    "target_currency": "MXN"
  }'
```

Quote response (201 Created)

```
{
  "id": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "blockchain": "SOLANA",
  "send": "150.00",
  "send_currency": "USDC",
  "target_country": "MX",
  "target_currency": "MXN",
  "fee": "2.00",
  "fee_currency": "USDC",
  "order_price": "2895.00",
```

```
  "order_price_currency": "MXN",
  "rate": "19.56",
  "expires": "2030-01-01T11:05:00Z"
}
```

> **TIP**
>
> Save the quote `id`. You will use it when creating the foreign order.

> **QUOTE VALIDATION**
>
> Creating a quote can fail with `400` if:
>
> - your account is not configured for the destination country or currency (`No matching configuration found for merchant, country, and currency`),
> - the pair is not valid for the blockchain (`<send_currency>:<target_currency> is not a valid pair for <blockchain> blockchain`),
> - the amount to send is too low (`The amount to send is too low.`).

> **EXCHANGE-RATE PROTECTION (`424`)**
>
> If we can't secure a fair exchange rate at that moment, the quote returns `424` with `A dependent operation failed.`. This is a safeguard, not a bug in your integration: we only issue a quote when the rate meets our guarantees, so your user is never left with an unfavorable rate.
>
> It's a **temporary** condition and can affect one blockchain in particular while liquidity rebalances. If you hit it, **retry in a few minutes**; if one network keeps returning `424`, offer your user another supported blockchain (`SOLANA`, `STELLAR`, `MONAD`) as an alternative path.

---

2) Create a foreign withdrawal order

When your user accepts the quote, create the foreign pay-out order using the `quote`.

Endpoint: `POST /merchants/orders/pay-out/`

Request fields

Field	Type	Required	Description
`order_type`	String	Yes	Must be `ForeignCurrencyOrder`.
`quote`	UUID	Yes	ID of the previously created quote.
`description`	String	Yes	Transaction description.
`merchant_order_id`	String	Yes	Unique ID from your system (unique per merchant).
`notify_url`	URL	Yes	URL for status notifications.
`return_url`	URL	Yes	Return URL.

Field	Type	Required	Description
`expiry`	DateTime	Yes	Order expiration (ISO 8601).
`consumer_email`	String	Conditional	Required if you don't send a phone number.
`consumer_phone_number`	String	Conditional	Required if you don't send an email.
`wallet`	String	Yes	Wallet that will sign the on-chain transaction.

Request example

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
    "description": "Cash withdrawal from USDC balance",
    "merchant_order_id": "FX-PO-2026-0001",
    "notify_url": "https://your-merchant.com/webhooks/pago46",
    "return_url": "https://your-merchant.com/withdrawal/back",
    "consumer_email": "user@example.com",
    "expiry": "2030-01-03T12:05:00Z",
    "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV",
    "order_type": "ForeignCurrencyOrder"
  }'
```

⚠ IMPORTANT

Do not assume a successful (201 Created) POST to `/pay-out/` means the withdrawal is actually available!

- The order has been created, but it **DOES NOT INCLUDE** the blockchain transaction to sign.
- Pago46 will build the transaction asynchronously.
- When it's ready, you will receive a webhook (`CREATED` status) that includes the `transaction` field in the order.
- You may receive multiple webhooks with the same `status` (`CREATED`) but different content; treat the one containing `transaction` as the **"transaction ready"** event.
- **Do not attempt to sign or show blockchain data to the user until you receive this webhook!**

Immediate POST response (201 Created, without a transaction)

```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "order_type": "ForeignCurrencyOrder",
  "country": "MX",
  "price": "2895.00",
  "price_currency": "MXN",
  "description": "Cash withdrawal from USDC balance",
  "merchant_order_id": "FX-PO-2026-0001",
  "status": "CREATED",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "redirect_url": "https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122",
  "return_url": "https://your-merchant.com/withdrawal/back",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "",
  "expiry": "2030-01-03T12:05:00Z",
  "paid": null,
  "transaction": "",
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV"
```

```
}
```

Webhook payload example (with a transaction)


```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "order_type": "ForeignCurrencyOrder",
  "country": "MX",
  "price": "2895.00",
  "price_currency": "MXN",
  "description": "Cash withdrawal from USDC balance",
  "merchant_order_id": "FX-PO-2026-0001",
  "status": "CREATED",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "redirect_url": "https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122",
  "return_url": "https://your-merchant.com/withdrawal/back",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "",
  "expiry": "2030-01-03T12:05:00Z",
  "paid": null,
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "transaction":
"AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADyTBSt8EsV++3pX0uVJVEfDjDp8mnxoqBQocHm
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV"
}
```


3) Sign and broadcast the on-chain transaction

When the webhook with the `transaction` field arrives, that value is the **unsigned** transaction that Pago46 already built. The format depends on the quote's network (base64, XDR, JSON, etc.) — see the per-network detail below. Your user must sign it with the `wallet` you provided when creating the order and broadcast it to the network to deposit the funds to the Pago46 address.

The procedure is each network's standard one; you don't need any Pago46-specific logic:

1. **Decode and parse** the `transaction` field according to the network's format (see detail below).
2. **Deserialize** it with the corresponding blockchain SDK.
3. **Sign** it with the user's `wallet`.
4. **Submit the signed transaction** to the network and wait for confirmation.

Once the transaction is confirmed on-chain and Pago46 validates the receipt of funds, the order advances to `READY` and you receive a webhook. This state confirms that the on-chain funds reached Pago46 within the expected window and that the order is ready to be disbursed in cash to the end user.

By the quote's network:

- **Solana** — the transaction is a `VersionedTransaction`. Deserialize it from base64, sign it with the wallet keypair and send it through a Solana RPC. See @solana/web3.js and the Solana docs.
- **Stellar** — the transaction is an XDR `TransactionEnvelope`. Decode it, sign it with the account key and submit it to Horizon. See the Stellar JS SDK and the Stellar docs.
- **Monad** — it is EVM-compatible. The `transaction` field arrives as a JSON object with the transaction parameters (not base64). Pass it to your usual library (ethers.js or viem) to sign and broadcast it like any Ethereum transaction. See the Monad docs.

SANDBOX USES TESTNET; PRODUCTION USES MAINNET

Sandbox runs on the **testnet** of all three blockchains (Solana, Stellar and Monad); **Production** runs on **mainnet**. Point the wallet and the RPC to the correct network for each environment. See Environments for each one's base URLs.

⚠ TEST TOKEN IN SANDBOX

In Sandbox we use a **non-mintable** test token for the source assets (for example, `USDC`). To test on testnet you need us to fund you with that token: request it from your account executive or email contacto@pago46.com.

---

4) Retrieve an order

You can retrieve the withdrawal order whenever you need to verify its status.

Endpoint: `GET /merchants/orders/pay-out/{id}/`

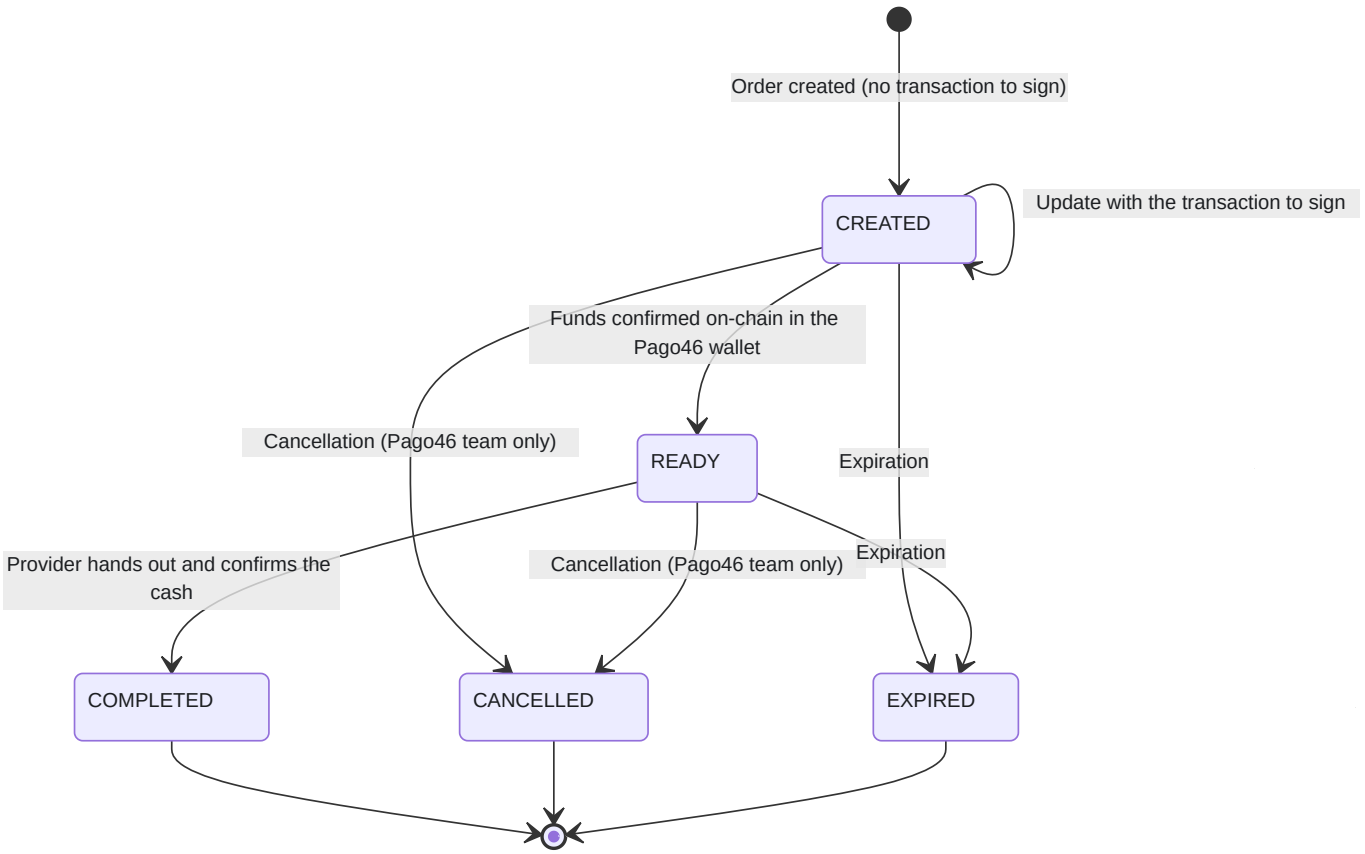
```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/01952d91-a0ff-7f57-8f4e-68d95be01122/" \
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Order states (foreign pay-out)

As a merchant, you receive webhooks when the order reaches a final state. In addition, you receive a `CREATED` update containing the transaction you must sign and a `READY` webhook when we confirm the on-chain funds. The intermediate cash-handout step is handled internally.

Status	Notified via webhook?	What does it mean for the merchant?
`CREATED`	Response and update	The initial response registers the order without `transaction`; a later update, still in `CREATED`, includes the information you must sign
`READY`	Yes	We confirmed receipt of the funds on-chain in the Pago46 wallet; the order is ready to be disbursed in cash
`PAYMENT_STARTED`	No (internal)	Intermediate step: a provider is handing the cash to the beneficiary
`COMPLETED`	Yes — final state	The withdrawal completed; the beneficiary received the cash
`CANCELLED`	Yes — final state	The order was cancelled by the Pago46 team (neither the merchant nor the user can cancel) and will not be processed
`EXPIRED`	Yes — final state	The order expired and will not be processed

State transition diagram



Webhooks (notifications)

Throughout this flow you'll receive several webhook notifications at the `notify_url` you provided when creating the order: the `CREATED` update with the transaction to sign, `READY` when the on-chain funds are confirmed, and the final state (`COMPLETED`, `CANCELLED`, or `EXPIRED`). See the table in the previous section for the detail on each.

Webhook structure

The webhook will be sent with HMAC authentication. You must verify the signature to ensure the notification comes from Pago46.

Webhook headers

```
POST /webhooks/pago46 HTTP/1.1
Host: your-merchant.com
Content-Type: application/json
Merchant-Key: <YOUR_MERCHANT_KEY>
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Webhook payload

State: COMPLETED

```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "order_type": "ForeignCurrencyOrder",
```

```
  "country": "MX",
  "price": "2895.00",
  "price_currency": "MXN",
  "description": "Cash withdrawal from USDC balance",
  "merchant_order_id": "FX-PO-2026-0001",
  "status": "COMPLETED",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "redirect_url": "https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122",
  "return_url": "https://your-merchant.com/withdrawal/back",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "",
  "expiry": "2030-01-03T12:05:00Z",
  "paid": "2030-01-01T12:10:00Z",
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "transaction":
"AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADyTBSt8EsV++3pX0uVJVEfDjDp8mnxoqBQocHm
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV"
}
```


State: CANCELLED


```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "order_type": "ForeignCurrencyOrder",
  "country": "MX",
  "price": "2895.00",
  "price_currency": "MXN",
  "description": "Cash withdrawal from USDC balance",
  "merchant_order_id": "FX-PO-2026-0001",
  "status": "CANCELLED",
  "notify_url": "https://your-merchant.com/webhooks/pago46",
  "redirect_url": "https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122",
  "return_url": "https://your-merchant.com/withdrawal/back",
  "consumer_email": "user@example.com",
  "consumer_phone_number": "",
  "expiry": "2030-01-03T12:05:00Z",
  "paid": null,
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "transaction":
"AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADyTBSt8EsV++3pX0uVJVEfDjDp8mnxoqBQocHm
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV"
}
```


Webhook verification

It's **critical** that you verify the authenticity of each received webhook to prevent processing fraudulent notifications.

Python verification example

```python
import hmac
import hashlib
import json
from flask import Flask, request, jsonify

app = Flask(__name__)

# Your Merchant Secret (obtained from Pago46)
MERCHANT_SECRET = "your_merchant_secret_here"

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    # 1. Extract headers
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')

    # 2. Get raw body
    body_str = request.get_data(as_text=True)

    # 3. Build string to sign
    # Format: MERCHANT_KEY:MESSAGE_DATE:METHOD:PATH:BODY
    method = request.method  # "POST"
```

```
    path = request.path      # "/webhooks/pago46"
    string_to_sign = f"{merchant_key}:{message_date}:{method}:{path}:{body_str}"

    # 4. Calculate HMAC
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    # 5. Verify
    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # 6. Process notification
    order_data = json.loads(body_str)
    order_id = order_data.get('id')
    order_status = order_data.get('status')
    merchant_order_id = order_data.get('merchant_order_id')

    print(f"Order {merchant_order_id} ({order_id}) changed to status: {order_status}")

    # Update your database
    if order_status == 'COMPLETED':
        # Mark as completed
        paid_at = order_data.get('paid')
        print(f"Withdrawal completed on: {paid_at}")
    elif order_status == 'CANCELLED':
        # Mark as cancelled
        print("Withdrawal cancelled")

    # 7. Respond with 200 OK
    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    app.run(port=5000)
```

Node.js verification example

```
const express = require('express');
const crypto = require('crypto');
const app = express();

const MERCHANT_SECRET = 'your_merchant_secret_here';

app.post('/webhooks/pago46', express.text({ type: '*/*' }), (req, res) => {
  // 1. Extract headers
  const merchantKey = req.headers['merchant-key'];
  const messageDate = req.headers['message-date'];
  const receivedHash = req.headers['message-hash'];

  // 2. Body as string
  const bodyStr = req.body;

  // 3. Build string to sign
  const method = req.method;
  const path = req.path;
  const stringToSign = `${merchantKey}:${messageDate}:${method}:${path}:${bodyStr}`;

  // 4. Calculate HMAC
  const calculatedHash = crypto
    .createHmac('sha256', MERCHANT_SECRET)
    .update(stringToSign)
    .digest('hex');

  // 5. Verify
  if (calculatedHash !== receivedHash) {
    return res.status(403).json({ error: 'Invalid signature' });
  }

  // 6. Process notification
  const orderData = JSON.parse(bodyStr);
  const { id, status, merchant_order_id, paid } = orderData;

  console.log(`Order ${merchant_order_id} (${id}) changed to status: ${status}`);

  if (status === 'COMPLETED') {
    console.log(`Withdrawal completed on: ${paid}`);
```

```
    // Update database
  } else if (status === 'CANCELLED') {
    console.log('Withdrawal cancelled');
    // Update database
  }

  // 7. Respond
  res.status(200).json({ status: 'received' });
});

app.listen(5000, () => {
  console.log('Webhook server listening on port 5000');
});
```

⚠ WEBHOOK RESPONSE

You must respond with an HTTP `200` or `201` status code to confirm reception. If you don't respond successfully, we will retry sending the notification.

We retry on any `5XX`, `408`, or `429` response, and also on timeouts or connection errors (no response received), using exponential backoff. In **production**, up to 600 attempts over roughly 4 days, capped at 10 minutes between attempts. In **sandbox**, only 3 attempts over a few seconds, following the same backoff pattern.

If the notification for an already-completed order fails in a non-retryable way (for example, retries are exhausted), we will reach out to you through one of your already-established communication channels to help resolve any issues your users may be experiencing with their order.

Integration best practices

- Use an idempotent `merchant_order_id` per business operation.
- Handle status changes in a retry-tolerant manner.
- Do not assume `COMPLETED` immediately after creating the order.
- Show the user a countdown based on the quote window.
- After a POST to `/pay-out/`, maintain a status like "Waiting for transaction" until you receive the webhook containing the `transaction` field. Only then proceed to collect a signature from the user or advance the flow.
- Store the returned order `id` so you can correlate the initial POST and the webhook notification.
- In case of validation errors, always request a new quote before retrying. See Common Errors for the exact error formats returned by the API for validation and logic issues.
- **Implement retries:** If your webhook server is down, Pago46 will retry the delivery.

If you need support for new asset pairs (for example, additional source currencies or destination countries), contact your account executive or email contacto@pago46.com.

Checkout

Checkout is Pago46's web application designed to be embedded in an iframe in your application. From there, your users can complete Pay-In and Pay-Out flows with an interface optimized for mobile and desktop.

What is Checkout?

Checkout standardizes the final user journey in one place, regardless of the operation type or entry channel. This helps you:

- reduce implementation time for new flows,
- keep a consistent UX across web and mobile,
- accelerate end-to-end testing,
- simplify operational support with a shared execution path.

Internally, Checkout includes technical monitoring and performance tracking to support operations while the flow runs.

Environments

Environment	Base URL
Sandbox	`https://checkout.dev.pago46.io`
Production	`https://checkout.prd.pago46.io`

Per-order format:

- Sandbox: `https://checkout.dev.pago46.io/{UUID}`
- Production: `https://checkout.prd.pago46.io/{UUID}`

You can also open the base URL (without a UUID) and manually enter the order ID.

`redirect_url` is returned in order create/retrieve responses and points directly to the Checkout URL for that specific order.

Iframe integration

Recommended flow:

1. Create the order in Pago46.
2. Take `redirect_url` from the order response.
3. Load that `redirect_url` in an iframe inside your app.
4. For QA, also validate manual UUID input from the Checkout home page.

Base iframe example:

```
<iframe
  src="https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122"
  width="100%"
  height="720"
  style="border: 0"
  allow="clipboard-write"
></iframe>
```

Practical recommendations for web iframe integration:

- Use full width and enough height to avoid excessive inner scrolling.
- Do not block cookies, storage, or in-domain navigation for Checkout.
- Allow `clipboard-write` and standard browser capabilities when needed.
- Handle session expiration and reloads without breaking your parent screen.
- For QA, also test base URL access with manual UUID entry.

Mobile app integration

Checkout is integrated as a WebView, with no additional SDK required.

Mobile app with WebView

- Load `redirect_url` directly in the WebView.
- Avoid intercepting Checkout navigation unless strictly needed.
- Keep a stable User-Agent to reduce rendering inconsistencies.

Native app (iOS/Android)

- Embed Checkout as a WebView in your payout/collection flow.
- Handle the back action carefully to avoid accidental flow exits.
- Define clear rules for opening external links outside the WebView.

React Native app

- Integrate Checkout via WebView (no proprietary SDK required).
- Keep the same contract: receive `redirect_url` and load it in the view.
- Test full foreground/background cycles to validate continuity.

Browser permissions

Checkout uses the clipboard so users can copy payment codes. If you embed it in an iframe, enable `clipboard-write` (see the example above). It also needs a stable internet connection: the order status is kept up to date by periodic polling while the screen is open.

Recommended test flow

1. Create a sandbox order.
2. Take `redirect_url` and load it in an iframe or WebView.
3. Complete the flow with the preloaded order URL.
4. Repeat from the Checkout home page, entering the UUID manually.
5. Validate behavior on desktop, Android, and iOS.

Demos

Android - Pay-In

View embedded content online: https://drive.google.com/file/d/19XG84b29aGuBPA_VWpPrI1J4QknkosVe/preview

Android - Pay-Out

View embedded content online: https://drive.google.com/file/d/1iSXarQQocBATZjLsEGcDEajYLfvm6Aon/preview

iOS

View embedded content online: https://drive.google.com/file/d/1Dy4FT7jsURd4IItC0MKwaOFUdeZG4dcU/preview

Desktop - Pay-In

View embedded content online: https://drive.google.com/file/d/1mhqbKd0hJ0yt9YpZrglx5ORVV8M8umiY/preview

Desktop - Pay-Out

View embedded content online: https://drive.google.com/file/d/1dD6Ma2HMlS9_X3YZVJf9D84wG2XIRZVU/preview

Payment Provider Integration

As a payment provider, you integrate your network of physical points (stores, kiosks, agents) with Pago46 to process cash collections and disbursements generated by merchants.

Available services

Pay-In (Collections)

Process cash payments from end users at your physical points.

Flow: The user presents a code → you validate and lock the order → you receive the cash → you confirm the transaction.

View Pay-In documentation

Pay-Out (Disbursements)

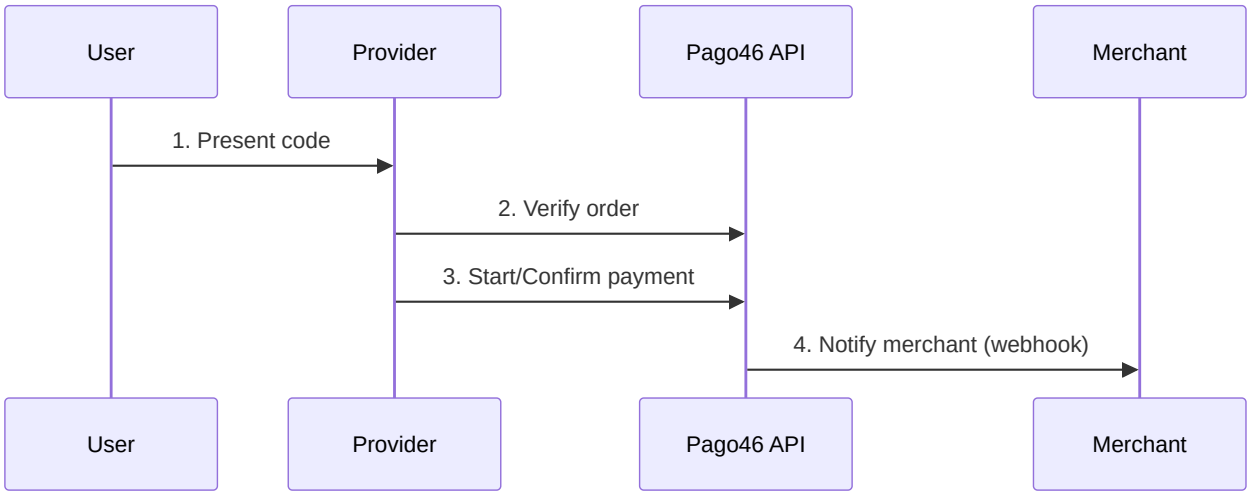
Disburse cash to end users at your physical points.

Flow: The user presents a code → you validate and lock the order → you deliver the cash → you confirm the transaction.

View Pay-Out documentation

Integration flow

The architecture is server-to-server with HMAC SHA-256 authentication.



Integration steps

1. **Get your credentials** - Receive your `Provider-Key` and `Provider-Secret`
2. **Register your networks** - Send us your list of networks/subnetworks to get `network_id`
3. **Implement HMAC authentication** - View guide
4. **Query orders** - Use `GET /providers/orders/{type}/{code}/`
5. **Process transactions** - Use `start-payment` and `confirm-payment`

LOCKING MECHANISM

Locking applies to both Pay-In and Pay-Out. When you call `start-payment`, the order is reserved for your network until you confirm or cancel.

NETWORKS AND SUBNETWORKS

To call the endpoints, you need a valid `network_id`. Send us your list of networks/subnetworks so we can register them and assign the IDs.

Environments

Environment	URL	Usage
Sandbox	`api.dev.pago46.io`	Development and testing
Production	`api.prd.pago46.io`	Real operations

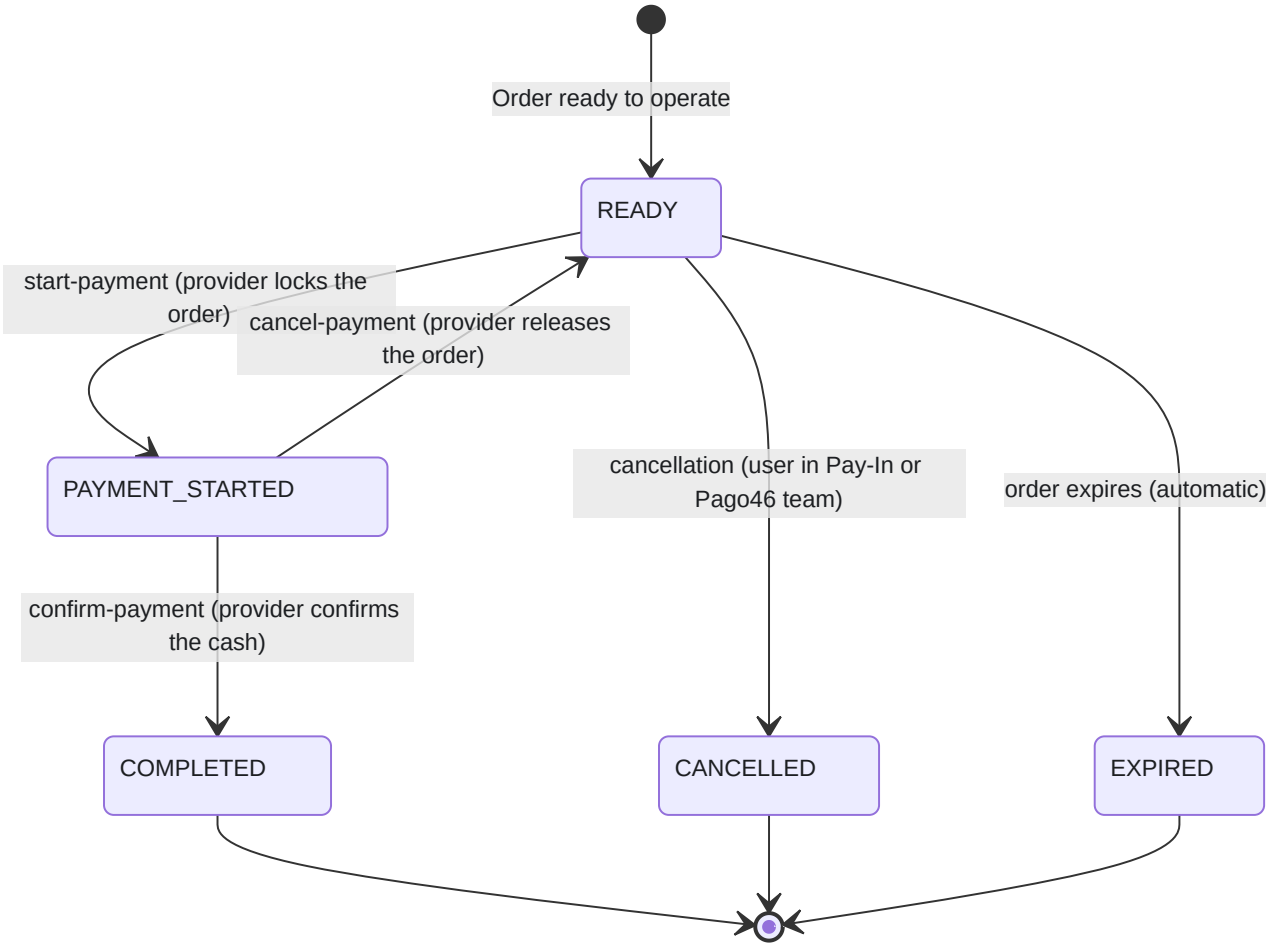
→ View environment details

OPERATIONAL MONITORING

For day-to-day operations in cashier/support teams, review the Service Status page and subscribe to email or Slack alerts to stay informed about incidents and maintenance.

Order lifecycle

The state diagram is the same for Pay-In and Pay-Out. Each arrow shows the **action that triggers the transition**: the ones labeled with an endpoint name (`start-payment`, `confirm-payment`, `cancel-payment`) are the ones you, as a provider, call via API; the rest happen automatically or are triggered by the merchant.



FROM WHICH STATE CAN I ACT?

The action you can take depends on the order's current state:

- **Take an order** → `start-payment`, available **only from** **`READY`**.
- **Release / "return" an order** → `cancel-payment`, available **only from** **`PAYMENT_STARTED`**, **before confirming**, and only to the provider that locked the order. The order goes back to `READY` and is available again. If another provider locked it, you cannot act on it.
- **Confirm the transaction** → `confirm-payment`, available **only from** **`PAYMENT_STARTED`**. The order moves to `COMPLETED`. Confirm **only once the cash has changed hands**: for Pay-In when you received it, for Pay-Out when you handed it out.
- `COMPLETED`, `CANCELLED`, and `EXPIRED` are **final** states: they allow no further transitions.

NO API REVERSALS

This is a cash operation and confirmation is immediate: **an order in** **`COMPLETED`** **cannot be returned or reversed** through the API. Any later adjustment is resolved in the settlement process agreed upon with the Pago46 team. That's why `confirm-payment` must always reflect a real movement of cash.

WHO CAN CANCEL AN ORDER?

As a provider you **never** move an order to `CANCELLED`: `cancel-payment` only releases it back to `READY`. A definitive cancellation is triggered by:

- In **Pay-In**: the end user or the Pago46 team.

- In **Pay-Out: only the Pago46 team**.

Merchants cannot cancel orders.

Coverage

Pago46 operates in multiple countries across Latin America. View complete list.

Payments (Pay-In)

This module allows Payment Providers to process **cash payment** orders created by merchants for their users. Like withdrawals, this flow uses a locking mechanism to prevent the same order from being charged twice by different providers simultaneously.

Sign every request with HMAC SHA-256, see the error format for handling non-successful responses, and check the supported countries and currencies. New here? Start with the Payment Providers introduction.

> **BASE ENDPOINT**
>
> All routes described below are relative to the API base URL: `/api/v1`

Payment network registration

Before processing transactions, providers must register the payment networks they use. Each network must have a unique identifier (`network_id`) that will be used in all state transitions.

Example network IDs:

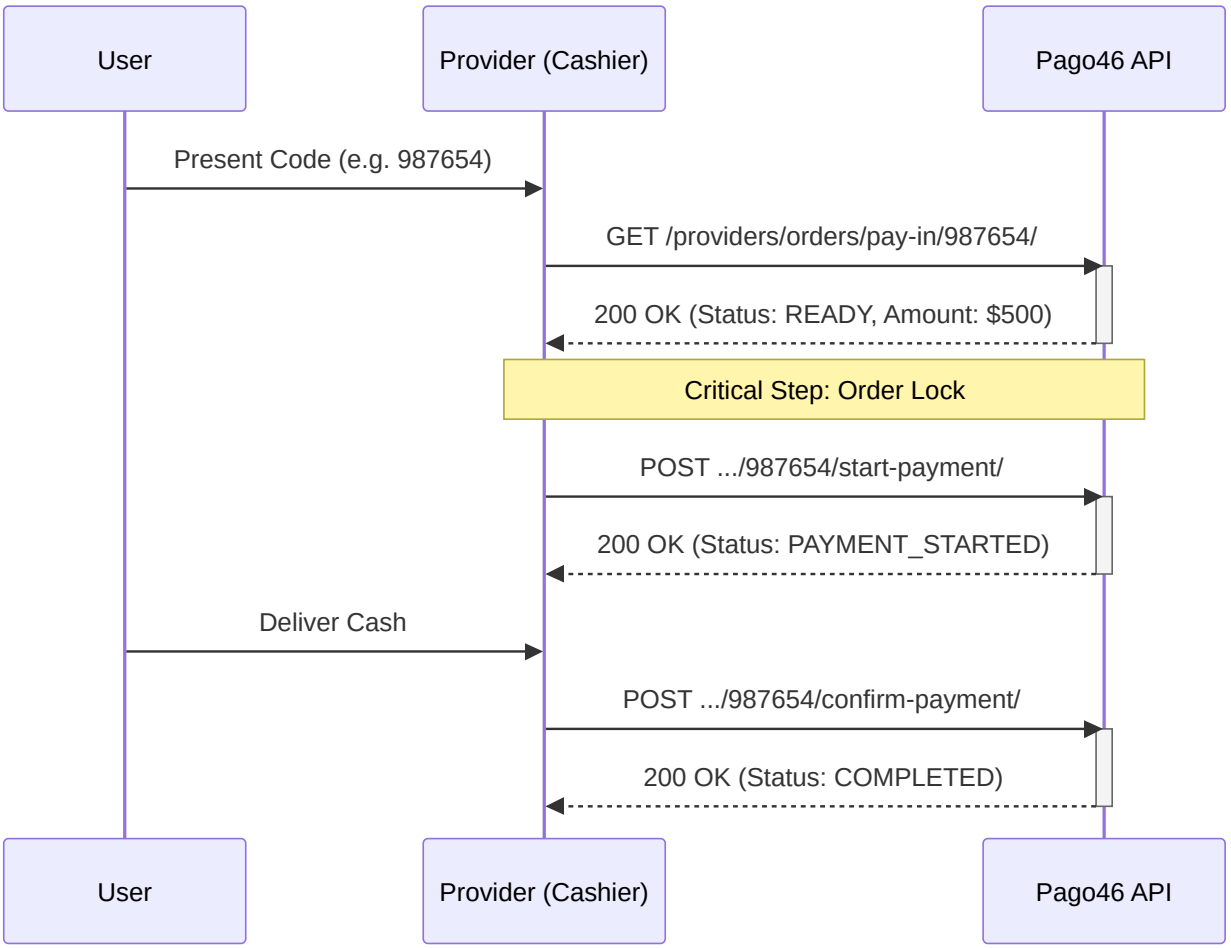
- `01`
- `oxxo_network`
- `seven_eleven_mx`
- `farmacia_abc`
- `network_123`

> **CONTACT**
>
> To register your payment networks, contact your account executive or email contacto@pago46.com. The IDs you provide will be the ones you must use in payment requests.

Order lifecycle

The process ensures that money is collected and reconciled correctly before releasing the service to the end user.

1. **Check:** The provider scans or enters the user's code to see the amount to collect and validate that the status is `READY`.
2. **Lock (Start Payment):** The provider "takes" the order. The status changes to `PAYMENT_STARTED`. **This ensures collection intent.**
3. **Confirmation (Confirm Payment):** Once the user delivers the cash and it enters the cash register, the provider confirms the transaction (`COMPLETED` status).



---

1. Verify the order

When the customer approaches the cashier to pay, you must query the code to report the exact amount to collect.

Endpoint: `GET /providers/orders/pay-in/{code}/`

Parameter	Location	Description
`code`	Path	The payment code generated by the user (numeric or QR).

Request

```
curl -X GET "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Response (200 OK) - Order Ready

```
{
  "network_id": null,
  "price": "500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T14:00:00Z",
  "modified": "2026-06-25T14:00:00Z",
```

```
  "status": "READY",
  "expiry": "2026-06-26T14:00:00Z"
}
```

> **STATUS VALIDATION**
>
> Always verify that the `status` is **`READY`**. If the order is `EXPIRED` or `COMPLETED`, you should not receive money from the user.

---

2. Start the collection (lock)

Before receiving the money, you must inform Pago46 that you are processing this order. This prevents the user from attempting to pay the same code at another provider simultaneously.

Endpoint: `POST /providers/orders/pay-in/{code}/start-payment/`

Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/start-payment/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "500.00",
    "price_currency": "MXN"
  }'
```

Response (200 OK)

```
{
  "network_id": "network_123",
  "price": "500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T14:00:00Z",
  "modified": "2026-06-25T14:05:00Z",
  "status": "PAYMENT_STARTED",
  "expiry": "2026-06-26T14:00:00Z"
}
```

Error (400) - Order Not Available

```
{
  "type": "validation_error",
  "errors": [
    {
      "code": "invalid",
      "detail": "Transition is not allowed",
      "attr": null
    }
  ]
}
```

Field	Type	Description
`network_id`	string	ID of the payment network used by the provider. **Required**.

Field	Type	Description
`price`	string	Optional. If sent, it must match the order's `price`.
`price_currency`	string	Optional. If sent, it must match the order's `price_currency`.

SUBNETWORKS FOR PROVIDERS

As a provider, you must supply the `network_id` that identifies the specific network used to process this transaction. This allows Pago46 to maintain a detailed record of which network processed each payment and provide this information to the end user.

WHEN TO COLLECT

Once you receive the `PAYMENT_STARTED` status, you can safely proceed to request and receive the customer's money.

---

3. Confirm the collection (finalization)

Once the money is secure in your cash register, confirm the transaction. At this point, Pago46 will notify the original merchant to release the product or service to the user.

Endpoint: `POST /providers/orders/pay-in/{code}/confirm-payment/`

Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/confirm-payment/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "500.00",
    "price_currency": "MXN"
  }'
```

Response (200 OK)

```
{
  "network_id": "network_123",
  "price": "500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T14:00:00Z",
  "modified": "2026-06-25T14:06:00Z",
  "status": "COMPLETED",
  "expiry": "2026-06-26T14:00:00Z"
}
```

Cancellation

If the user decides not to pay at the last moment or there is a problem with the cash, you must release the order. You can only do so when the order is in `PAYMENT_STARTED` and your provider locked it. If another provider locked it, you cannot act on it. Releasing the

order returns it to `READY` so it can be processed again.

Endpoint: `POST /providers/orders/pay-in/{code}/cancel-payment/`

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/cancel-payment/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "500.00",
    "price_currency": "MXN"
  }'
```

Status summary

Status	Description	Required provider action
`READY`	The order is pending payment.	Verify amount and call `start-payment`.
`PAYMENT_STARTED`	The order is being collected.	Receive money and call `confirm-payment`.
`COMPLETED`	Money was successfully collected.	Provide receipt to user.
`CANCELLED`	The order was cancelled.	Do not receive money.

Withdrawals (Pay-Out)

This module allows Payment Providers to process cash withdrawal orders created by merchants for their users. The flow is designed to guarantee transaction atomicity and prevent duplicate fund delivery through a locking mechanism.

Sign every request with HMAC SHA-256, see the error format for handling non-successful responses, and check the supported countries and currencies. New here? Start with the Payment Providers introduction.

> **BASE ENDPOINT**
>
> All routes described below are relative to the API base URL: `/api/v1`

Payment network registration

Before processing transactions, providers must register the payment networks they use. Each network must have a unique identifier (`network_id`) that will be used in all state transitions.

Example network IDs:

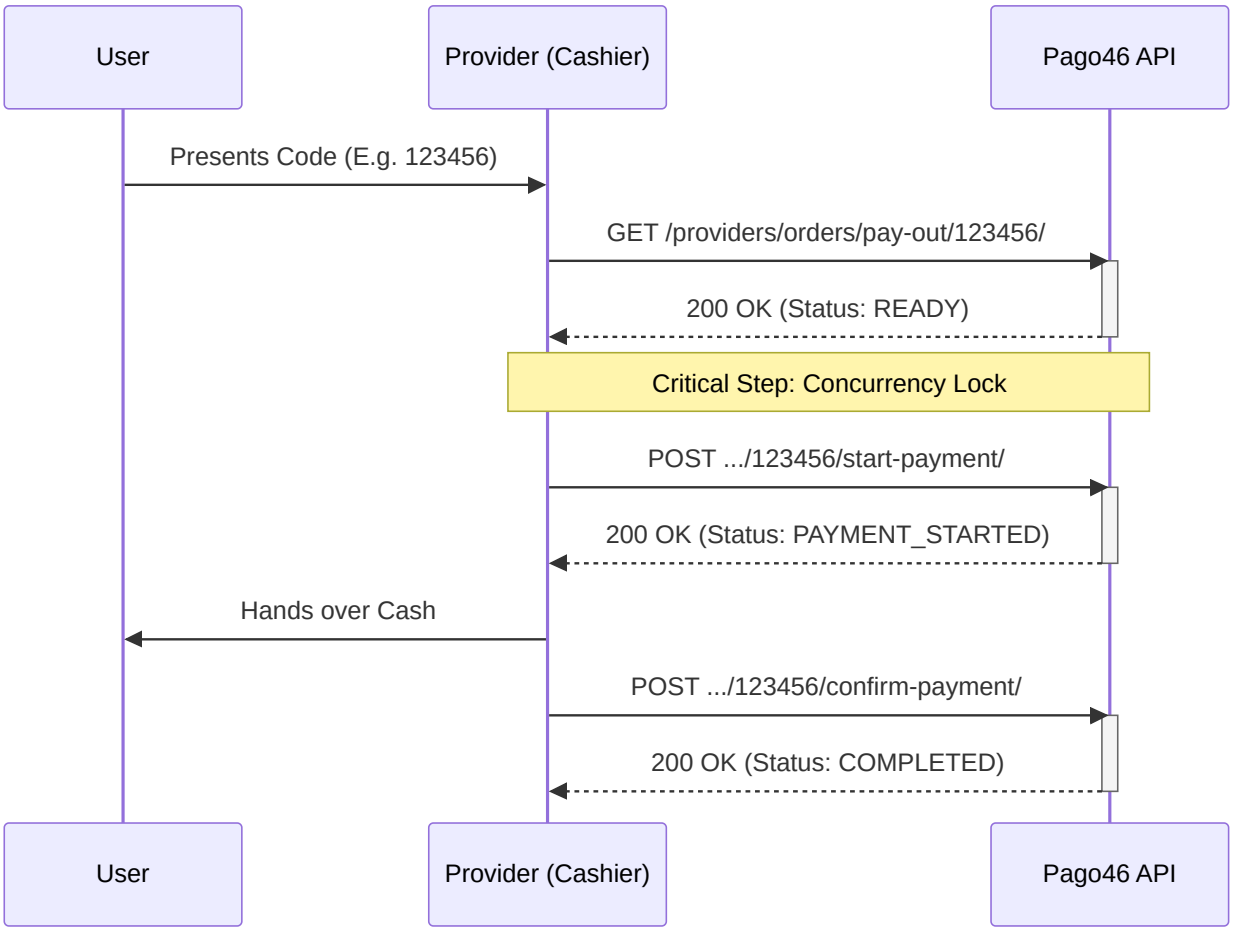
- `01`
- `oxxo_network`
- `seven_eleven_mx`
- `farmacia_abc`
- `network_123`

> **CONTACT**
>
> To register your payment networks, contact your account executive or email contacto@pago46.com. The IDs you provide will be the ones you must use in withdrawal requests.

Order lifecycle

The process is governed by strict state changes to ensure money is delivered only once.

1. **Check:** The provider verifies if the code presented by the user is valid and the order is in the `READY` state.
2. **Start Payment (Lock):** The provider "claims" the order. This changes the status to `PAYMENT_STARTED`. **At this point, no other provider can process this order.**
3. **Confirm Payment:** Once the cash is handed over, the provider confirms the transaction, moving the order to `COMPLETED`.



---

1. Verify the order

The first step occurs when the user presents their withdrawal code at the physical location. You must query the order details to validate the amount and its availability.

Endpoint: `GET /providers/orders/pay-out/{code}/`

Parameter	Location	Description
`code`	Path	The numeric or alphanumeric code presented by the end user.

Request

```
curl -X GET "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Response (200 OK) - Order Ready

```
{
  "network_id": null,
  "price": "1500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T10:00:00Z",
```

```
  "modified": "2026-06-25T10:05:00Z",
  "status": "READY",
  "expiry": "2026-06-26T10:00:00Z"
}
```

Response (200 OK) - Order Already In Progress

```
{
  "network_id": "network_123",
  "price": "1500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T10:00:00Z",
  "modified": "2026-06-25T10:05:00Z",
  "status": "PAYMENT_STARTED",
  "expiry": "2026-06-26T10:00:00Z"
}
```

> **STATUS VALIDATION**
>
> You must only proceed to the next step if the `status` is **`READY`**. If you receive `CANCELLED`, `COMPLETED`, or `EXPIRED`, you must inform the user that the order cannot be processed.

---

2. Start the payment (lock)

This is the most critical step. By executing this endpoint, you indicate the intention to pay the order. The system will **lock** the order for your provider and change its status to `PAYMENT_STARTED`.

If another provider attempts to start payment for the same order while it is locked, they will receive a `400` error indicating the transition is not allowed.

Endpoint: `POST /providers/orders/pay-out/{code}/start-payment/`

Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/start-payment/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "1500.00",
    "price_currency": "MXN"
  }'
```

Response (200 OK)

```
{
  "network_id": "network_123",
  "price": "1500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T10:00:00Z",
  "modified": "2026-06-25T10:06:00Z",
  "status": "PAYMENT_STARTED",
  "expiry": "2026-06-26T10:00:00Z"
}
```

Error (400) - Order Not Available

```
{
  "type": "validation_error",
  "errors": [
    {
      "code": "invalid",
      "detail": "Transition is not allowed",
      "attr": null
    }
  ]
}
```

Field	Type	Description
`network_id`	string	ID of the payment network used by the provider. **Required**.
`price`	string	Optional. If sent, it must match the order's `price`.
`price_currency`	string	Optional. If sent, it must match the order's `price_currency`.

> **SUBNETWORKS FOR PROVIDERS**
>
> As a provider, you must supply the `network_id` that identifies the specific network used to process this transaction. This allows Pago46 to maintain a detailed record of which network processed each payment and provide this information to the end user.

> **SAFE OPERATION**
>
> Once you receive the `200 OK` with status `PAYMENT_STARTED`, it is safe to proceed with the physical handover of money or internal fund management. The order is reserved for you.

---

3. Confirm the payment (completion)

Once the money has been successfully handed to the user (or the internal transaction has finished), you must confirm the operation to definitively close the order.

Endpoint: `POST /providers/orders/pay-out/{code}/confirm-payment/`

Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/confirm-payment/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "1500.00",
    "price_currency": "MXN"
  }'
```

Response (200 OK)

```
{
```

```
  "network_id": "network_123",
  "price": "1500.00",
  "price_currency": "MXN",
  "created": "2026-06-25T10:00:00Z",
  "modified": "2026-06-25T10:08:00Z",
  "status": "COMPLETED",
  "expiry": "2026-06-26T10:00:00Z"
}
```

After receiving this response, Pago46 notifies the merchant that created the order through the configured webhook.

Cancellation (optional)

If for any reason (insufficient funds in the register, operational error) you cannot complete the payment after calling `start-payment`, you **must** release the order so it is not left locked indefinitely. You can only do so when the order is in `PAYMENT_STARTED` and your provider locked it; if another provider locked it, you cannot act on it.

Endpoint: `POST /providers/orders/pay-out/{code}/cancel-payment/`

This returns the order to `READY`, where it becomes available for processing again.

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/cancel-payment/" \
  -H "Provider-Key: <YOUR_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "1500.00",
    "price_currency": "MXN"
  }'
```

Status summary

Status	Description	Required provider action
`READY`	The order is ready to be paid.	Can call `start-payment`.
`PAYMENT_STARTED`	The order is locked by a provider.	Must hand over money and call `confirm-payment`.
`COMPLETED`	The flow finished successfully.	No action required. Historical record.
`CANCELLED`	The order was cancelled.	Do not hand over money.

Service Status

The **Pago46 Status Page** centralizes our operational status so you can check service availability, incident updates, and scheduled maintenance in real time.

Official URL: `https://pago46.statuspage.io`

What information is available?

- **Current component status:** Fast health overview of key services.
- **Active and historical incidents:** Updates on degradations, outages, and resolution progress.
- **Scheduled maintenance:** Planned windows and expected impact.

Subscriptions and alerts

If you operate in production, we recommend subscribing to receive proactive notifications.

Email

1. Open `https://pago46.statuspage.io`.
2. Click **Subscribe to updates**.
3. Select **Email**.
4. Enter your email and confirm your subscription.

Slack

1. Open `https://pago46.statuspage.io`.
2. Click **Subscribe to updates**.
3. Select **Slack**.
4. Authorize the integration and choose the channel for alerts.

> **OPERATIONAL RECOMMENDATION**
>
> Subscribe at least one support/on-call email address and one shared Slack channel so your team has visibility during incidents or maintenance.

Embedded status widget in this documentation

In addition to the Status Page itself, this documentation site already includes the official Statuspage widget.

If there is **downtime**, degradation, or **maintenance**, you can also see status signals directly here without leaving the docs.

When to check it

- If you detect unusual API errors or webhook disruptions.
- Before escalating an incident, to verify whether there is an official update.
- When planning critical production deployments.

If you need extra help, contact us at contacto@pago46.com.

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals

Design Guide

This guide provides the visual and technical resources needed to integrate Pago46 into your platform professionally and consistently.

Visual identity

Maintaining visual consistency helps build trust with the end user during the payment process.

Institutional colors

We have prepared exact values in different formats so you can copy and paste them directly into your stylesheet or design software.

Color	Primary Use	HEX	RGB	HSL	CMYK
Green	Action buttons	`#28C339`	`rgb(40, 195, 57)`	`hsl(127, 66%, 46%)`	`79, 0, 71, 24`
Dark Blue	Text and backgrounds	`#07214F`	`rgb(7, 33, 79)`	`hsl(218, 84%, 17%)`	`91, 58, 0, 69`
White	Light backgrounds	`#FFFFFF`	`rgb(255, 255, 255)`	`hsl(0, 0%, 100%)`	`0, 0, 0, 0`

Typography

Our official font is **Lato**. It improves readability in payment interfaces and on mobile devices.

- **Use in buttons:** Bold, **16px**, centered, letter-spacing: 0px.
- **Fallback:** If **Lato** cannot be loaded, fall back to sans-serif.

> **GET THE FONT**
>
> If you don't have the font in your project, you can download or import it for free via Google Fonts or find it on Adobe Fonts.

Primary logo variants

Official assets with transparent backgrounds. Use the **SVG** version whenever possible to ensure maximum sharpness.

Version	Color	SVG	PNG
Full	**Green**	**Download:** SVG	**Download:** sm · md · lg
	Blue	**Download:** SVG	**Download:** sm · md · lg
	White	**Download:** SVG	**Download:** sm · md · lg
Isotype	**Green**	**Download:** SVG	**Download:** sm · md · lg

Version	Color	SVG	PNG
	Blue	**Download:** SVG	**Download:** sm · md · lg
	White	**Download:** SVG	**Download:** sm · md · lg

Button catalog

We offer ready-to-use buttons in multiple languages, sizes, and color schemes. Select your preference below to get direct download links.

English

Variants with **"Cash"** and **"Pay with"** labels.

Green / Blue

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Green / White

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Blue / Green

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Blue / White

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

White / Blue

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

White / Green

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Size	Full (Cash)	Full (Pay with)	Iso (Cash)	Iso (Pay with)
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Español

Variantes con los textos **"Efectivo"** y **"Pagar con"**.

Verde / Azul

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Verde / Blanco

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Azul / Verde

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Azul / Blanco

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Blanco / Azul

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Blanco / Verde

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

BULK DOWNLOAD

You can download the complete package with all variants and languages at the following link.

📦 Download All Assets (.zip)

Implementation examples

Copy and paste these code snippets to quickly integrate the button into your frontend. These examples use the **Cash + Full Logo (Green Background / Dark Blue Text and Logo)** variant in its SVG version.

HTML & CSS

```
<button class="p46-btn-pay">
  <!-- We recommend using the SVG directly for better quality -->
  <img
    src="/img/buttons/green-and-dark-blue-contrast/logo-full/md/button-cash-EN.svg"
    alt="Pay with Pago46"
    class="p46-button-img"
  />
</button>

<style>
  .p46-btn-pay {
    background: none;
    border: none;
    padding: 0;
    cursor: pointer;
    transition: transform 0.2s ease, filter 0.2s ease;
  }

  .p46-btn-pay:hover {
    filter: brightness(90%);
  }

  .p46-btn-pay:active {
    transform: scale(0.98);
  }

  .p46-btn-pay:disabled {
    opacity: 0.5;
    cursor: not-allowed;
  }

  .p46-button-img {
    height: 48px; /* Adjust according to your design */
    display: block;
  }
</style>
```

React (Tailwind)

```
import React from "react";

const Pago46Button = ({ onClick, disabled }) => {
  return (
    <button
```

```
      onClick={onClick}
      disabled={disabled}
      className="transition-all hover:brightness-90 active:scale-95 disabled:opacity-50 disabled:cursor-not-allowed"
    >
      <img
        src="/img/buttons/green-and-dark-blue-contrast/logo-full/md/button-cash-EN.svg"
        alt="Pay with Pago46"
        className="h-[48px] block"
      />
    </button>
  );
};

export default Pago46Button;
```

Restrictions and practices to avoid

To avoid integration rejection during the certification process, **do not** perform the following actions:

- **Modify proportions:** Do not stretch or compress the logo or isotype.
- **Alter colors:** Do not change the institutional green to other green tones or non-brand colors.
- **Incorrect typography:** Do not use serif fonts like Times New Roman on the button.
- **Poor contrast:** Do not use the green logo on backgrounds of similar colors that make reading difficult.

CERTIFICATION

Before moving to production, our team will verify that your button implementation follows these guidelines to ensure the best conversion rate for your business.

Choose your guide

I'm a Merchant

I want to collect payments or send money to my users.

- Payments (Pay-In)
- Withdrawals (Pay-Out)
- Foreign-currency withdrawals

I'm a Payment Provider

I have physical points and want to process Pago46 transactions.

- Process payments
- Process withdrawals