Pago46 Documentation Center

Core API technical documentation · v2026.6.1

Contents

Welcome to Pago46

Pago46 is the payment infrastructure that digitizes cash in Latin America. Our platform bridges the gap between the online world and physical money, allowing any company to process **deposits (Pay-In)** and **withdrawals (Pay-Out)** in real-time through a single integration.

By connecting to our **Core API**, you gain instant access to a hybrid network that combines:

1. **Commercial and Banking Chains:** Consolidated partners such as *CajaVecina*, *RapiPago*, *ProvinciaNet*, among others.
2. **Socio46 Network:** Our force of mobile agents ("Uber for cash") who collect or deliver money at home or at geolocated points.

Ecosystem and Actors

The API is designed to orchestrate transactions between three main actors. Identify your role to navigate this documentation:

Actor	Role in API	Description
Merchant	`Initiator`	Online businesses, E-commerce, Wallets, or Fintechs that need to **charge** their users or **disburse** funds (loans, remittances).
Payment Provider	`Fulfiller`	Physical networks (Pharmacies, Banks, Kiosks) that use our API to **process** orders created by Merchants, earning commissions for traffic.
Consumer	`End-User`	The final user who hands over or receives the cash. Interacts with the Pago46 map or the Provider's point of sale.

Core Services

Our new Core API unifies operational flows into polymorphic endpoints, allowing you to handle operations in **local** or **foreign currency** with the same semantics.

1. Pay-In (Collections)

Allows users to pay for online purchases with cash.

- **For Merchants:** You generate an order and obtain a QR/Code. You receive digitally reconciled money.
- **For Providers:** You validate the user's code, receive the cash, and notify Pago46 to release the service.

2. Pay-Out (Withdrawals)

Allows users to withdraw balance from a digital wallet or receive a cash loan.

- **For Merchants:** You create a withdrawal order for your user.
- **For Providers:** You verify the order, block the transaction (Start Payment), and hand over the cash from your register.

Integration Flow

Pago46's architecture is **Server-to-Server**. To ensure security, we use a robust digital signature scheme.

1. **Authentication:** Implements the **HMAC SHA-256** standard to sign all your requests.
2. **Order Creation:** The Merchant initiates the payment or withdrawal intention (`/orders`).
3. **Physical Interaction:** The user goes to a point (Provider or Socio46) or requests an on-demand service.
4. **Confirmation:** The Provider notifies the API that the money has changed hands.
5. **Webhooks:** Pago46 notifies the Merchant in real-time that the transaction was successful.

> **NEW HERE?**
>
> We recommend starting by reading the **Authentication** section before attempting any API calls. It is the fundamental requirement to get a response from the server.

> **OPERATIONAL MONITORING**
>
> Check the **Service Status page** to verify incidents, maintenance windows, and subscription options for email or Slack alerts.

Where do I start?

Select the guide that best fits your integration needs:

🛍️ I am a Merchant

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

- Integrate Collections (Pay-In)
- Integrate Withdrawals (Pay-Out)
- Integrate foreign currency withdrawals

🏪 I am a Provider

I have physical locations and want to process Pago46 transactions.

- Process Payments
- Process Withdrawals

Ready to begin? Check the **Countries** section to learn about 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 allied *Payment Networks*, guaranteeing unified settlement and homogeneous SLAs in all markets.

Active Countries and Local Currencies (Fiat)

Currently, we process operations in local currency (Fiat) in the following territories. Ensure 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**
>
> Specific behavior and functionality configuration can be performed for each country, allowing the definition of 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 allows your users to pay or receive funds in a currency different from the local one, facilitating international expansion without the need for 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 am a Merchant

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

- Integrate Collections (Pay-In)
- Integrate Withdrawals (Pay-Out)

🏪 I am a Provider

I have physical locations and want to process Pago46 transactions.

- Process Payments
- Process Withdrawals

Doubts about expansion to your country? Contact mcontreras@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 flow 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.
Web checkout application	`https://checkout.dev.pago46.io`	Hosted page for consumers to complete payment flows.
Mobile application	-	We do not currently provide a sandbox environment for Android or iOS mobile apps.

*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.
Web checkout application	`https://checkout.prd.pago46.io`	Hosted page for consumers to complete payment flows.
Mobile application	Link to Apple AppStore or Link to Google Play Store	To use the application, you must be registered as a Socio46 partner.

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.

Authentication

The new Pago46 Core API uses a security scheme based on **HMAC SHA-256** to guarantee the integrity and authenticity of each transaction.

This mechanism ensures that whoever sends the request possesses the correct credentials and that the message has not been altered en route nor is it a repetition of an old request.

Mandatory Headers

Every HTTP request you make to our API must include the following headers:

Header	Type	Description
`Provider-Key`	String	Your unique provider identifier (API Key).
`Message-Date`	Timestamp	Unix Timestamp (floating point seconds or milliseconds).
`Message-Hash`	String	The hexadecimal result of the calculated HMAC signature.

> **IMPORTANT**
>
> The server will validate that the difference between the sent `Message-Date` and the current server time is not greater than **24 hours**. If the time exceeds this range, the request will be rejected to prevent *replay attacks*.

Signature Algorithm

To generate a valid `Message-Hash`, you must strictly follow the algorithm below to construct the signature string.

1. String Construction (String to Sign)

Unlike previous versions, this flow **does not require** sorting parameters alphabetically. The string is constructed by concatenating the following values separated by colons (`:`):

1. **Provider Key**
2. **Message Date** (The same value sent in the header)
3. **HTTP Method** (E.g., `POST`, `GET`)
4. **Path** (The resource path, e.g., `/api/v1/payments`)
5. **Body** (The raw body of the request in UTF-8)

Format:

```
PROVIDER_KEY:MESSAGE_DATE:METHOD:PATH:BODY
```

> **NOTE ON BODY**

If the request is `GET` and has no body, the `BODY` value must be an empty string. If it is a `POST` with JSON, make sure to use the exact JSON string you will send in the request, without extra spaces or modifications.

2. Hash Generation

Once the string is built, you must sign it using:

- **Algorithm:** HMAC SHA-256
- **Secret:** Your `Provider Secret` (provided by Pago46)
- **Message:** The string built in step 1.
- **Output:** Hexdigest (hexadecimal string).

Implementation Examples

Below, we present how to generate this signature in different languages.

Python **Node.js** **Go**

```python
import hmac
import hashlib
import time
import requests
import json

def send_authenticated_request(provider_key, provider_secret, method, path, body_dict=None):
    # 1. Prepare data
    host = "https://api.dev.pago46.io"
    timestamp = str(time.time()) # Current Timestamp (float as string)

    # If there is a body, convert to JSON string, otherwise, it is empty
    body_str = json.dumps(body_dict) if body_dict else ""

    # 2. Build the signature string (IMPORTANT: Separator is ":")
    # Format: KEY:DATE:METHOD:PATH:BODY
    string_to_sign = f"{provider_key}:{timestamp}:{method}:{path}:{body_str}"

    # 3. Calculate HMAC SHA-256
    calculated_hmac = hmac.new(
        provider_secret.encode("utf-8"),
        string_to_sign.encode("utf-8"),
        hashlib.sha256
    ).hexdigest()

    # 4. Send Request
    headers = {
        "Provider-Key": provider_key,
        "Message-Date": timestamp,
        "Message-Hash": calculated_hmac,
        "Content-Type": "application/json"
    }

    response = requests.request(
        method=method,
        url=f"{host}{path}",
        headers=headers,
        data=body_str
    )

    return response

# Usage
response = send_authenticated_request(
    provider_key="PK_12345",
    provider_secret="SECRET_XYZ",
    method="POST",
```

```
    path="/api/v1/payments/",
    body_dict={"amount": 100, "currency": "CLP"}
)
print(response.status_code)
```

Common Errors

HTTP Code	Message	Probable Cause
`403 Forbidden`	`Invalid authentication credentials`	The `Provider-Key` header does not exist or was not found in the database.
`403 Forbidden`	`Possible replay attack`	The `Message-Date` has a difference of more than 24 hours with the server.
`403 Forbidden`	`Hash mismatch`	The signature does not match. Verify that the separator is `:`, that the body is exact, and that the path is correct.

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.

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`: short string describing the error. Can be used by API consumers to customize their 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. In the next sections, we describe the most common ones. For more details, please refer to the linked resources in OpenAPI.

401 Unauthorized

These errors are returned with the status code 401 whenever the authentication fails or a request is made to an endpoint without providing the authentication information as part of the request. Here are the 2 possible errors that can be returned.

```
{
    "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 submitted and contains 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
        }
    ]
}
```

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

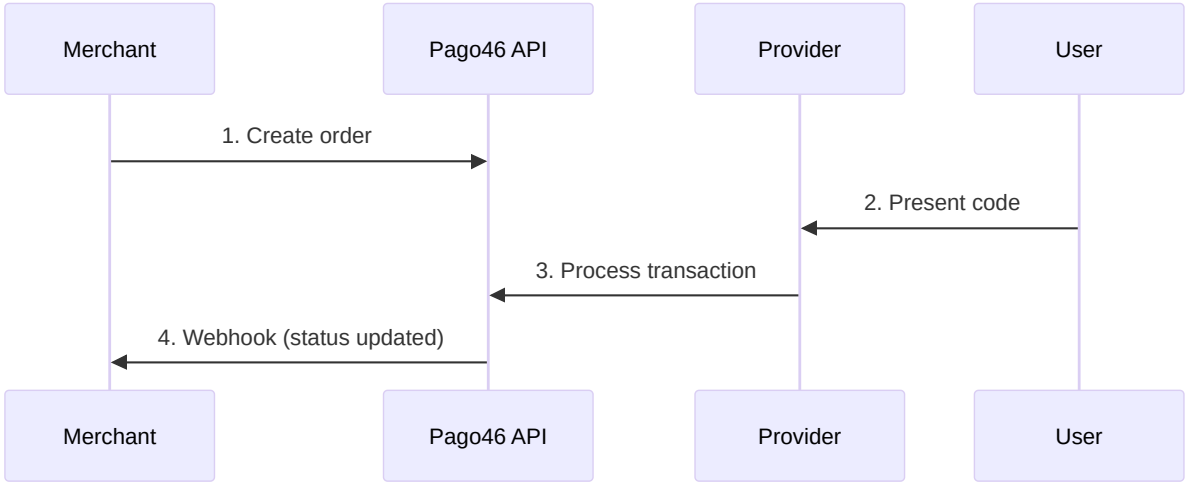
Embed the web Checkout application in an iframe for Pay-In and Pay-Out flows.

→ 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 Checkout home (without UUID) and enter the order id manually for iframe integration testing.

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.

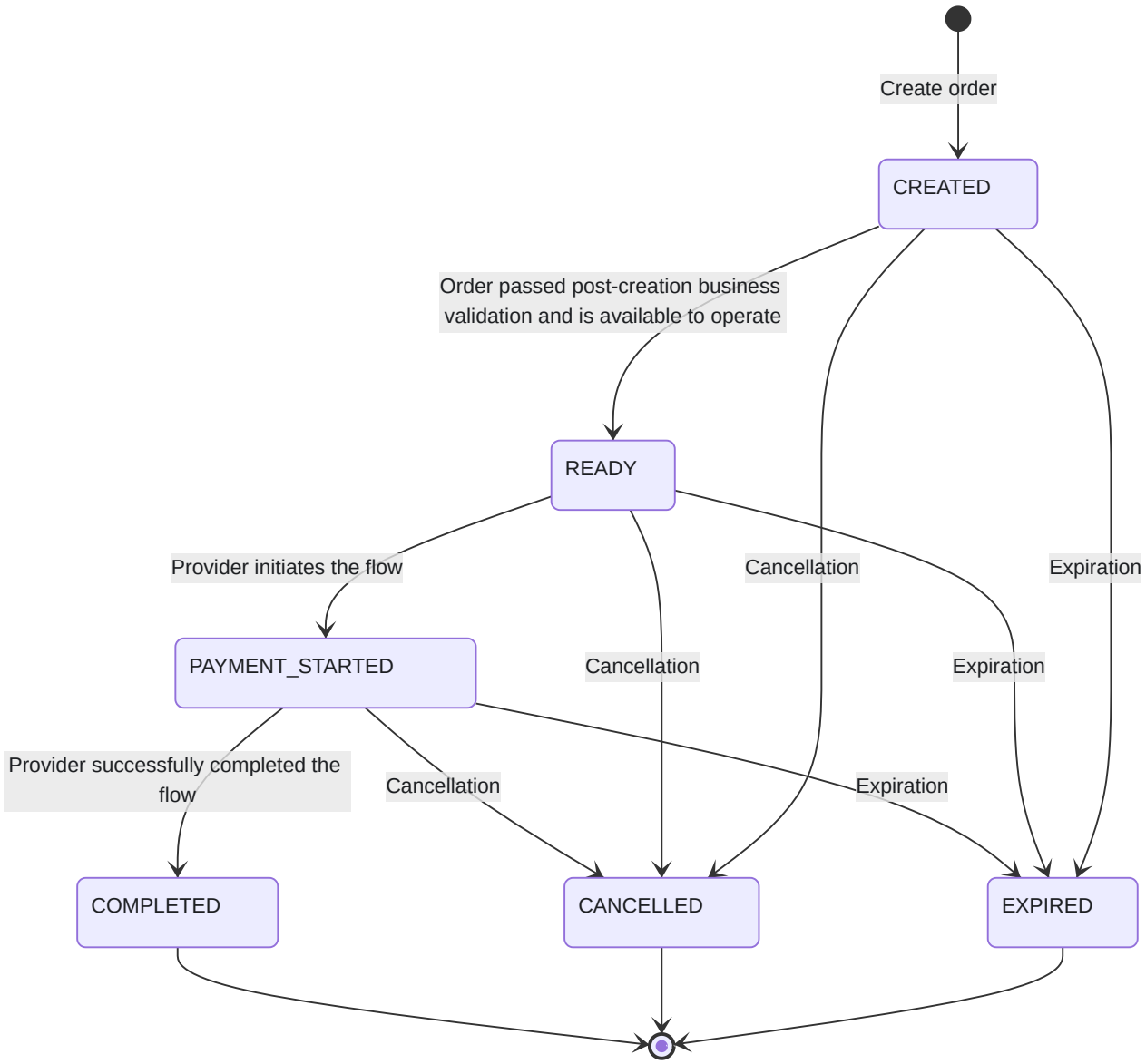
Environments

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

→ View environment details

Order Lifecycle

Orders transition through states that we report via webhooks:



You will receive an HTTP POST notification to your `notify_url` whenever the order is updated, including when the status changes or new information (such as transaction details) becomes available.

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}`

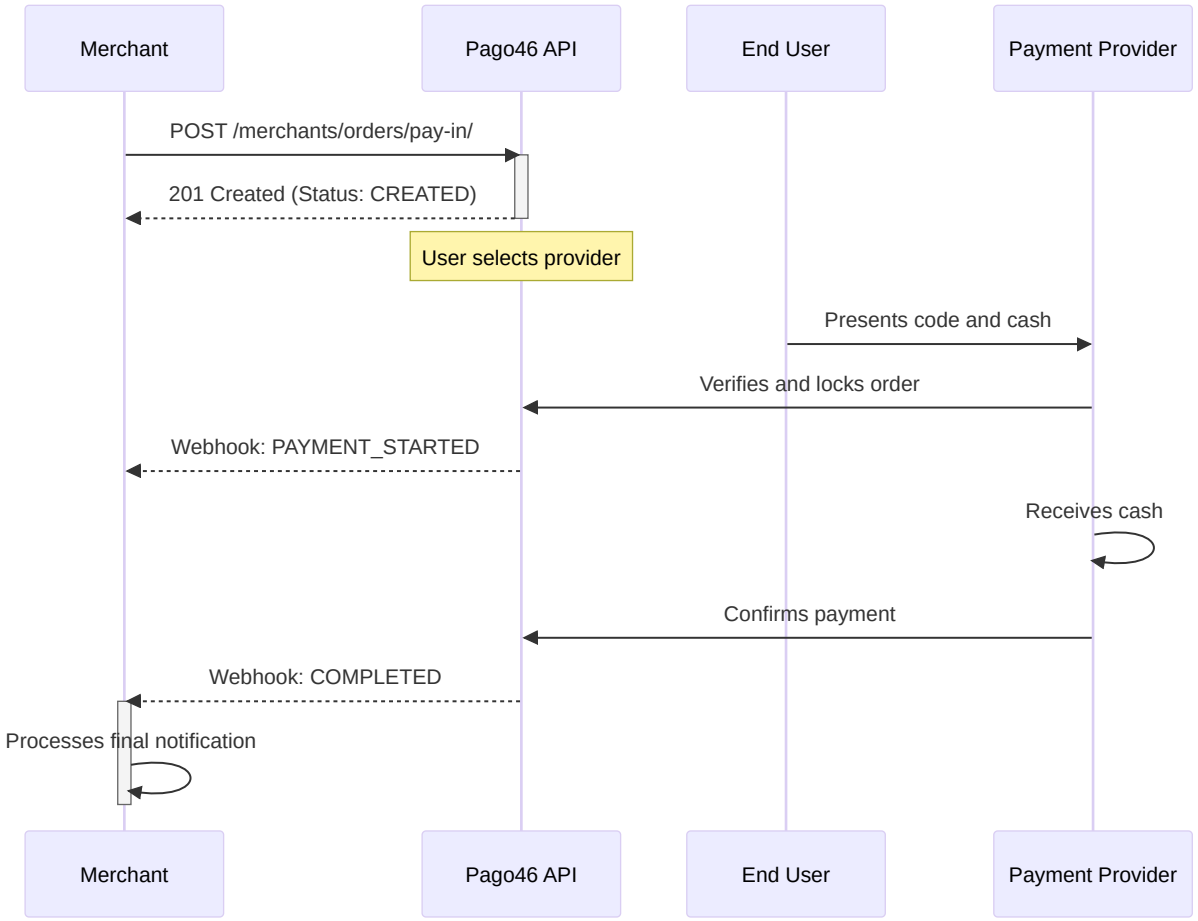
You can also open the base URL without UUID and manually enter the order id for iframe integration testing.

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 order status updates via configured webhooks.



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 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`, `CO`)
`price`	Decimal	Payment amount (format: `"1500.00"`)
`price_currency`	String	Currency ISO code (e.g., `MXN`, `CLP`, `COP`)
`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.

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": "2024-12-31T23:59:59Z"
  }'
```

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": "2024-12-31T23:59:59Z",
  "paid": null
}
```


SAVING THE ID

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

---

2. Query 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-a456-42661417400
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Response (200 OK)

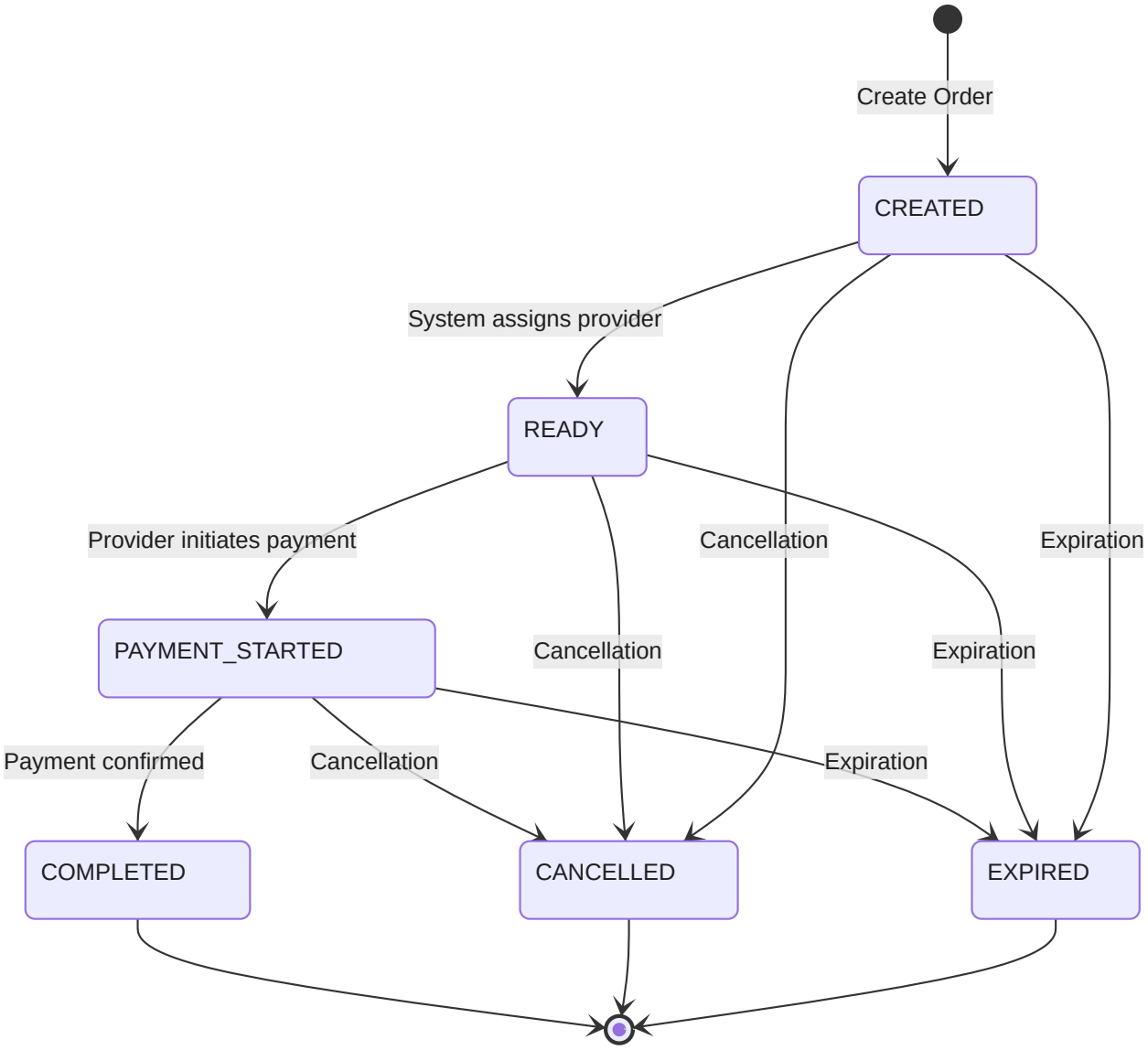

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Subscription payment - User ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "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": "2024-12-31T23:59:59Z",
  "paid": null
}
```

Order States

Each payment order goes through different states during its lifecycle. It's important that your system properly handles each of these states.

State	Description	What does it mean for the merchant?
`CREATED`	Order successfully created	The order has been registered and is pending provider assignment
`READY`	Ready for processing	A provider can begin processing the payment
`PAYMENT_STARTED`	Payment in progress	A provider has locked the order and is processing the payment
`COMPLETED`	Completed	The payment completed successfully. The provider received the cash from the user
`CANCELLED`	Cancelled	The order was cancelled and will not be processed
`EXPIRED`	Expired	The order expired and will not be processed

State Transition Diagram



Webhooks (Notifications)

Every time an order's status changes, Pago46 will send an HTTP POST notification to the URL specified in your order's `notify_url` field.

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: PAGO46_SYSTEM
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Webhook Payload

State: PAYMENT_STARTED | State: COMPLETED | 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": "PAYMENT_STARTED",
  "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": "2024-12-31T23:59:59Z",
  "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, Pago46 will retry sending the notification.

Common Errors

All error responses for order creation and updates use the unified Pago46 API error format and are documented in detail in the Common Errors section. This standard error structure applies to all pay-in operations. Please refer to that section for current formats, codes, and practical examples.

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 Table

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. Idempotency

Use the `merchant_order_id` field to identify unique orders in your system. If you need to retry order creation, use the same `merchant_order_id` to avoid duplicates.

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 a reasonable expiration time (`expiry` field). Orders typically expire 24-72 hours after creation.

4. State Monitoring

- Monitor orders in `PAYMENT_STARTED` state that don't transition to `COMPLETED` within a reasonable time.
- Implement alerts for orders that remain in intermediate states 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:

```
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": "2024-12-31T23:59:59Z"
    }

    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']}")

    # Update database based on status
    if order['status'] == 'CREATED':
        print("   ⏳ Order created and waiting for user to select a provider")
    elif order['status'] == 'PAYMENT_STARTED':
        print("   🔄 A provider is processing the payment")
    elif order['status'] == 'COMPLETED':
        print(f"   ✅ Payment completed on {order['paid']}")
        # Activate service, deliver product, etc.
    elif order['status'] == 'CANCELLED':
        print("   ❌ Payment cancelled")

    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("\n🚀 Starting 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 the Pago46 support team.

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}`

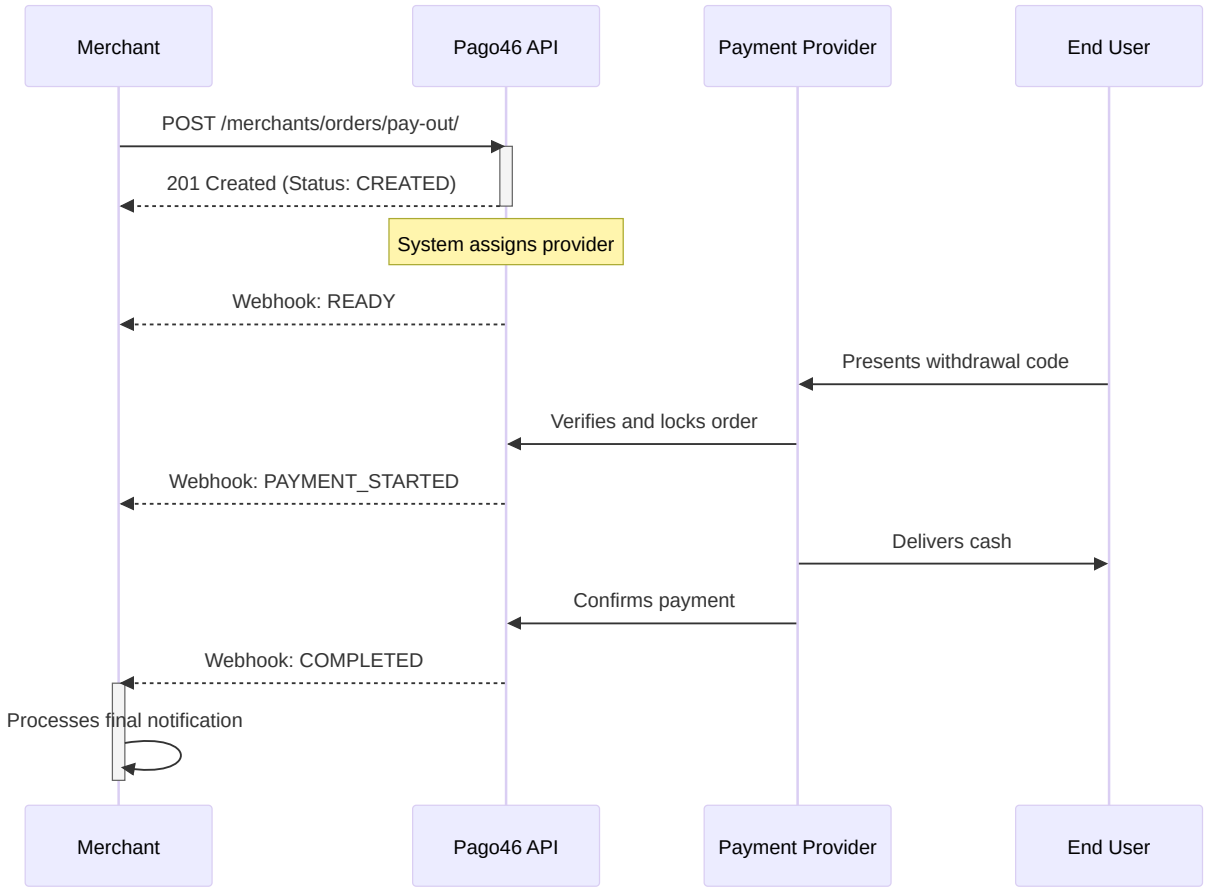
You can also open the base URL without UUID and manually enter the order id for iframe integration testing.

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 order status updates via configured webhooks.



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 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`
`country`	String	Country ISO code (e.g., `MX`, `CL`, `CO`)
`price`	Decimal	Withdrawal amount (format: `"1500.00"`)
`price_currency`	String	Currency ISO code (e.g., `MXN`, `CLP`, `COP`)
`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)

> **FOREIGN CURRENCY WITHDRAWALS**
>
> For international withdrawals or currency conversion, consult the **Foreign Currency Merchants** documentation in the sidebar menu.

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": "2024-12-31T23:59:59Z"
  }'
```

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": "2024-12-31T23:59:59Z",
  "paid": null
}
```


SAVING THE ID

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

---

2. Query 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-a456-426614174(
  -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": "2024-12-31T23:59:59Z",
  "paid": null
```

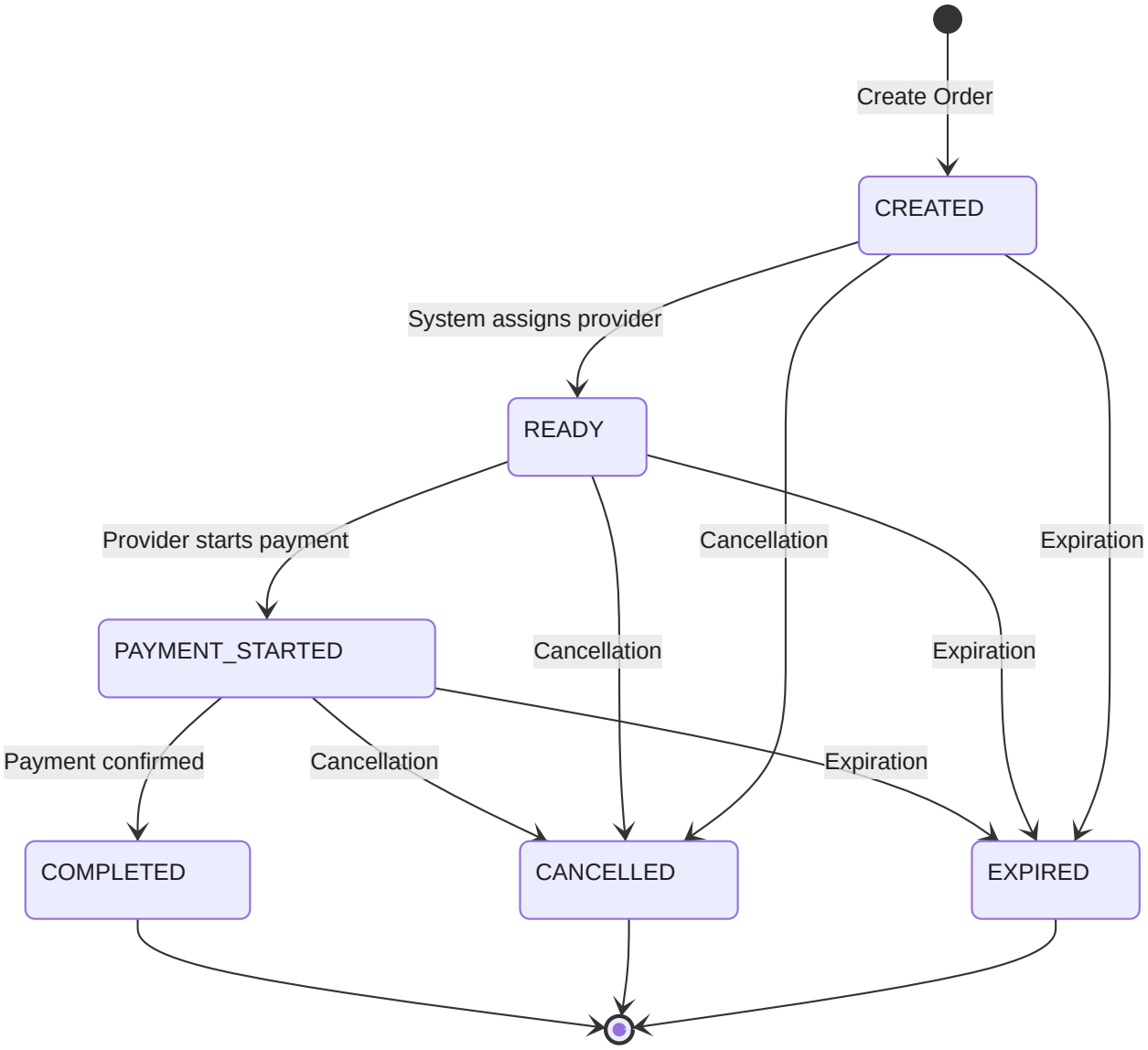
```
}
```

Order States

Each withdrawal order goes through different states during its lifecycle. It's important that your system properly handles each of these states.

State	Description	What does it mean for the merchant?
`CREATED`	Order successfully created	The order has been registered and is pending provider assignment
`READY`	Ready to process	A provider can begin processing the withdrawal
`PAYMENT_STARTED`	Payment in progress	A provider has locked the order and is processing the withdrawal
`COMPLETED`	Completed	The withdrawal completed successfully. The beneficiary received the cash
`CANCELLED`	Cancelled	The order was cancelled and will not be processed
`EXPIRED`	Expired	The order expired and will not be processed

State Transition Diagram



Webhooks (Notifications)

Every time an order's status changes, Pago46 will send an HTTP POST notification to the URL specified in your order's `notify_url` field.

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: PAGO46_SYSTEM
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Webhook Payload

State: READY | State: PAYMENT_STARTED | State: COMPLETED | 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": "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": "2024-12-31T23:59:59Z",
  "paid": null
}
```

Webhook Verification

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

Python Verification Example

```
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, Pago46 will retry sending the notification.

Common Errors

All error responses for withdrawal order creation and updates use the unified Pago46 API error format and are documented in detail in the Common Errors section. This standard error structure applies to all pay-out operations. Please refer to that section for the most current formats, error codes, and detailed examples.

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 of a required fields error response:

```
{
  "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 Table

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. Idempotency

Use the `merchant_order_id` field to identify unique orders in your system. If you need to retry order creation, use the same `merchant_order_id` to avoid duplicates.

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 a reasonable expiration time (`expiry` field). Orders typically expire 24-72 hours after creation.

4. State Monitoring

- Monitor orders in `PAYMENT_STARTED` state that don't transition to `COMPLETED` within a reasonable time.
- Implement alerts for orders that remain in intermediate states 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:

```
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": "2024-12-31T23:59:59Z"
    }

    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']}")

    # Update database based on status
    if order['status'] == 'READY':
        print("   ⏳ Order ready to be processed by a provider")
    elif order['status'] == 'PAYMENT_STARTED':
        print("   🔄 A provider is processing the withdrawal")
    elif 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.
```

```
    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("\n🚀 Starting webhook server...")
    app.run(port=5000)
```

Next Steps

- **Foreign Currency Withdrawals:** Consult the **Foreign Currency Merchants** documentation in the sidebar menu 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 the Pago46 support team.

Foreign Currency Withdrawals (Pay Out)

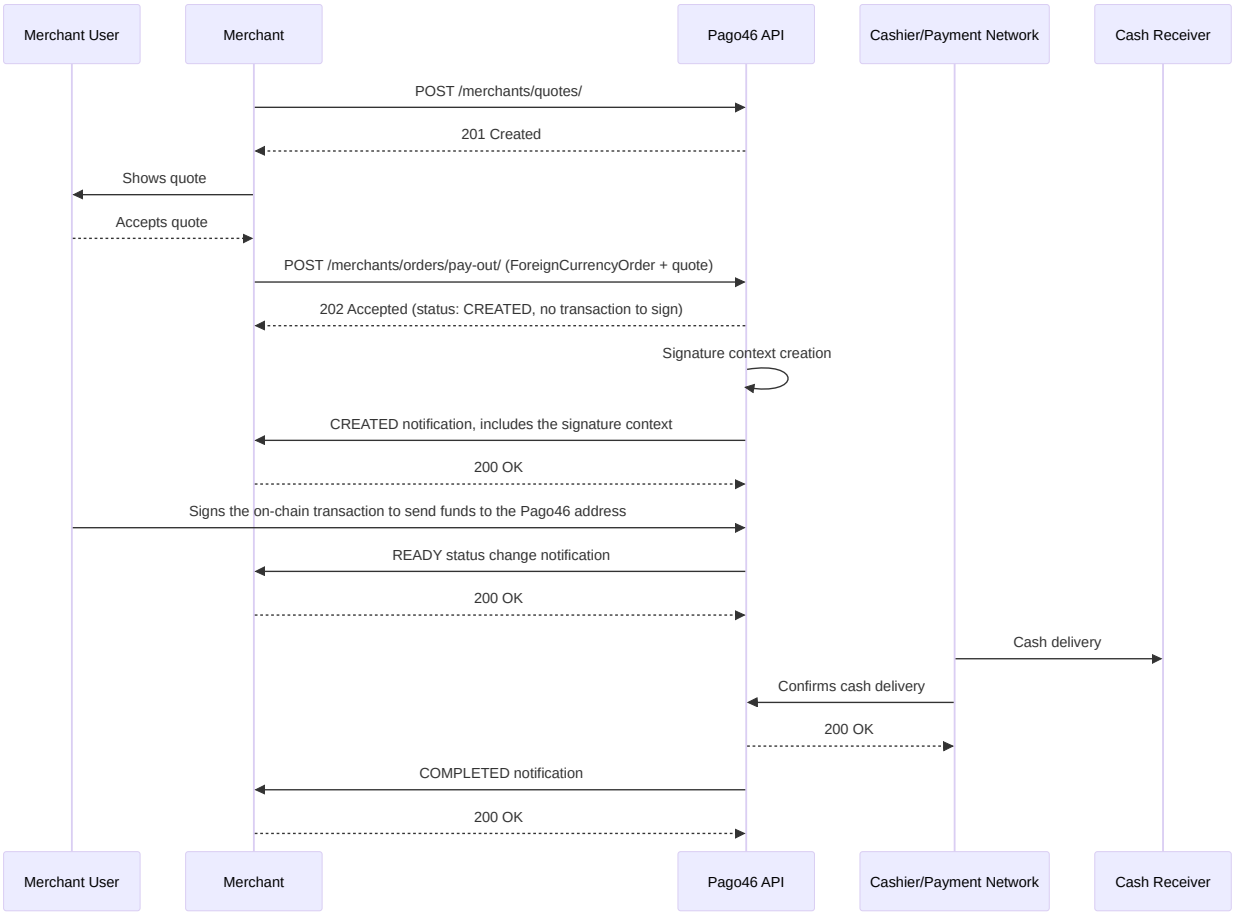
This module allows you to create withdrawals (pay-out) in local currency using source funds in foreign currency (for example, USDC → MXN).

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

> **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 is ready to 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 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"
  }'
```

Response (201 Created)

```
{
  "id": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "blockchain": "SOLANA",
  "send": "150.00",
  "send_currency": "USDC",
  "target_country": "MX",
  "target_currency": "MXN",
  "calculated_order_price": "2895.00",
  "calculated_order_price_currency": "MXN",
  "expires": "2030-01-01T11:05:00Z"
}
```

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

---

2) Create 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 per Merchant.
`notify_url`	URL	Yes	URL for status notifications.
`return_url`	URL	Yes	Return URL.
`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": "2030-01-01T12:05:00Z",
    "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV",
    "order_type": "ForeignCurrencyOrder"
  }'
```

> **IMPORTANT**
>
> **Do not assume a successful (202 Accepted) 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 will include the `transaction` field in the order.
- **Do not attempt to sign or show blockchain data to the user until you receive this webhook!**

Response (202 Accepted)


```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "direction": "PAY_OUT",
  "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": "2030-01-01T12:05:00Z",
  "paid": null,
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV",
  "order_type": "ForeignCurrencyOrder"
}
```

Notification webhook payload example


```
{
  "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": "2030-01-01T12:05:00Z",
  "paid": null,
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "transaction":
"AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADyTBSt8EsV++3pX0uVJVEfDjDp8mnxoq
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV"
}
```


---

3) Retrieve Order

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

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

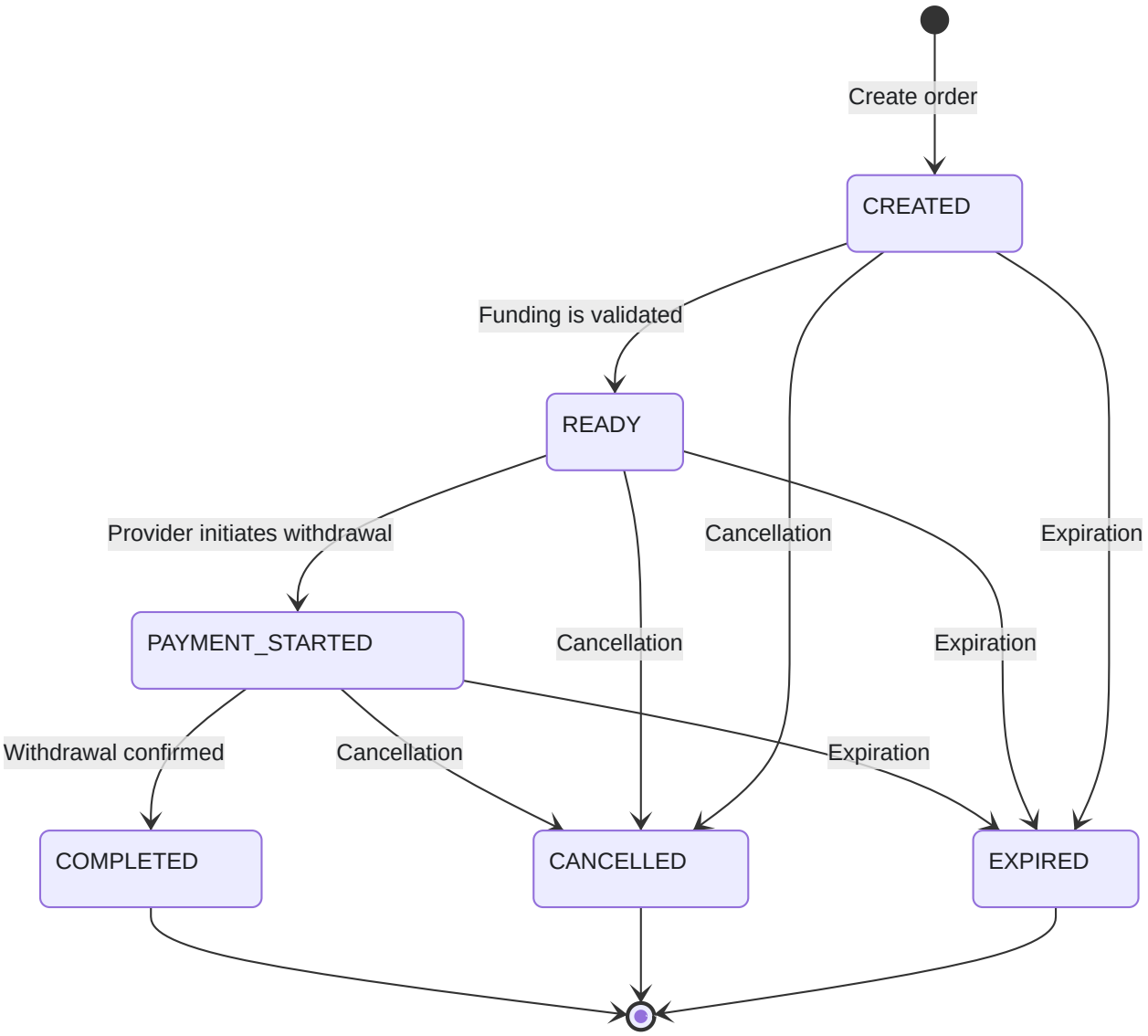
```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/01952d91-a0ff-7f57-8f4e-68d95be011
  -H "Merchant-Key: <YOUR_MERCHANT_KEY>" \
```

```
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Order Statuses (Foreign Pay-Out)

Status	Description	What does it mean for the merchant?
`CREATED`	Order successfully created	The order is registered and pending provider assignment
`READY`	Ready for processing	Funding received; withdrawal can continue in the payment network
`PAYMENT_STARTED`	Payment in progress	A provider is processing the withdrawal
`COMPLETED`	Completed	The withdrawal completed; the beneficiary received the cash
`CANCELLED`	Cancelled	The order was cancelled and will not be processed
`EXPIRED`	Expired	The order expired and will not be processed

State Transition Diagram



Integration Best Practices

- Use 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.

If you need support for new asset pairs (for example, additional source currencies or destination countries), contact your Pago46 representative.

Checkout

Checkout is Pago46's web application designed to be embedded as an iframe in your user experience. Users can complete pay-in and pay-out flows with an interface optimized for mobile and desktop.

Why Checkout matters

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.

Checkout also includes internal capabilities for technical monitoring, error and performance tracking, and operational support. It may load third-party services to improve the user experience, including geospatial components with real-time connections.

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 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 view.
- Test full foreground/background cycles to validate continuity.

User permissions that may be required

Depending on the flow and operational context, Checkout may require:

- approximate or precise location (geospatial services and nearby points),
- stable internet/network access for real-time updates,
- browser or WebView permissions related to geolocation.

Recommendation: explain why each permission is needed before system prompts.

Recommended test flow

1. Create a sandbox order.
2. Take `redirect_url` and load it in iframe or WebView.
3. Complete the flow with the preloaded order URL.
4. Repeat from Checkout home and manually enter UUID.

5. Validate behavior on desktop, Android, and iOS.

Demos

Android

iOS

Desktop

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: User presents code → Validate and lock order → Receive cash → Confirm transaction.

→ **View Pay-In documentation**

Pay-Out (Disbursements)

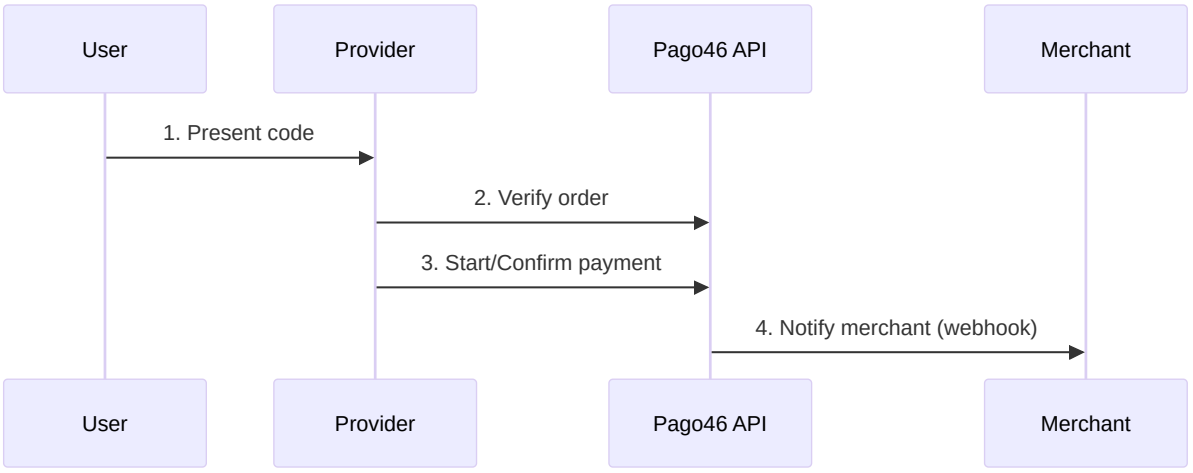
Disburse cash to end users at your physical points.

Flow: User presents code → Validate and lock order → Deliver cash → Confirm 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

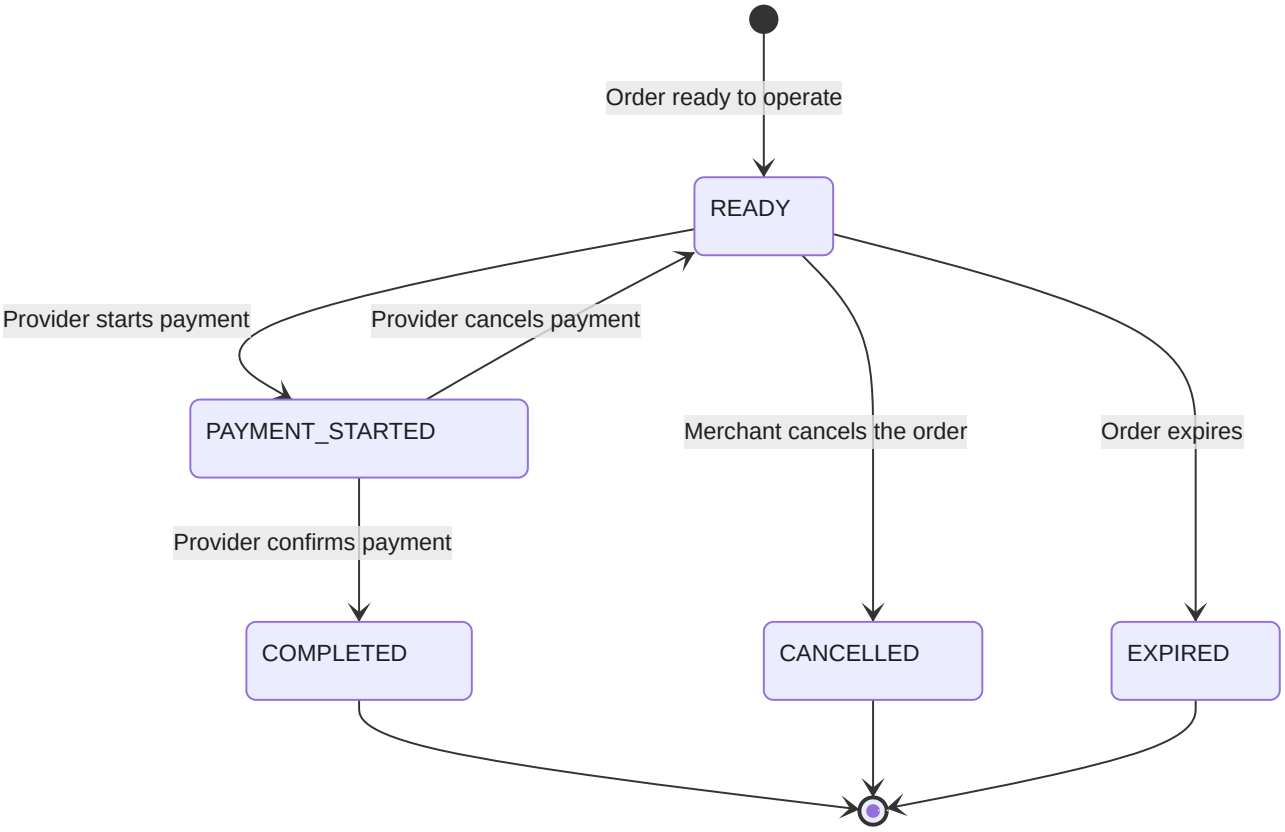
Environment	URL	Usage
Sandbox	`api.dev.pago46.io`	Development and testing
Production	`api.prd.pago46.io`	Real operations

→ 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.



Coverage

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

Payments (Pay In)

This module allows Payment Providers to process **deposit or payment** requests initiated by Pago46 users. Like withdrawals, this flow uses a locking mechanism to prevent the same order from being charged twice by different providers simultaneously.

> **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:

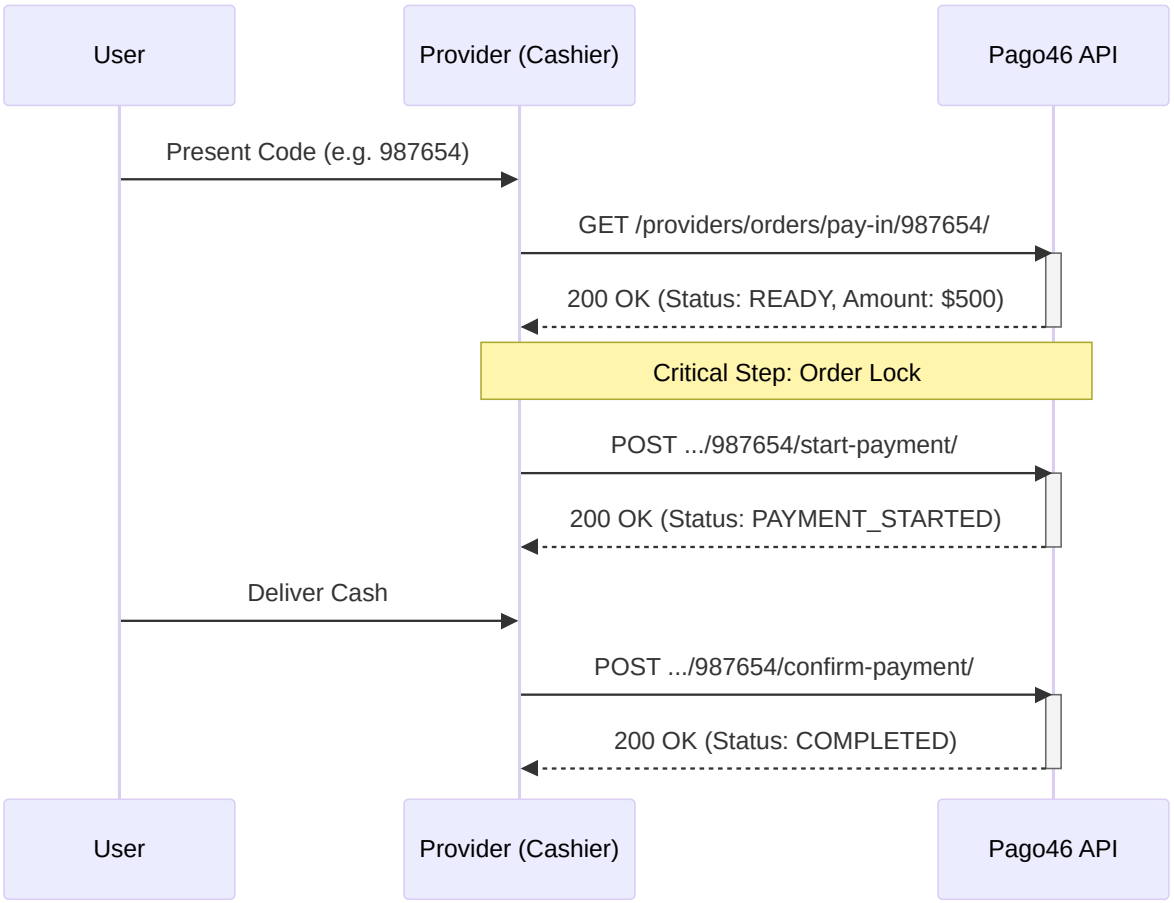
- `01`
- `oxxo_network`
- `seven_eleven_mx`
- `farmacia_abc`
- `network_123`

> **CONTACT**
>
> To register your payment networks, contact the Pago46 integration team. 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 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 **Response (200 OK) - Order Ready**

```
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>"
```

⚠ 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 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 | **Response (200 OK)** | **Error (403 Forbidden) - Order Already In Progress**

```
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"
  }'
```

Field	Type	Description
`network_id`	string	ID of the payment network used by the provider. **Required**.
`price`	string	Exact amount being collected. Must match the order's `price`. **Required**.
`price_currency`	string	Currency code (e.g., `MXN`, `USD`). Must match the order's `price_currency`. **Required**.

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.

COLLECTION MOMENT

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

3. Confirm 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 | **Response (200 OK)**

```
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"
  }'
```

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 so it becomes available again (or is canceled definitively, according to the order's expiration rules).

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 requests initiated by Pago46 users. The flow is designed to guarantee transaction atomicity and prevent duplicate fund delivery through a locking mechanism.

> **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:

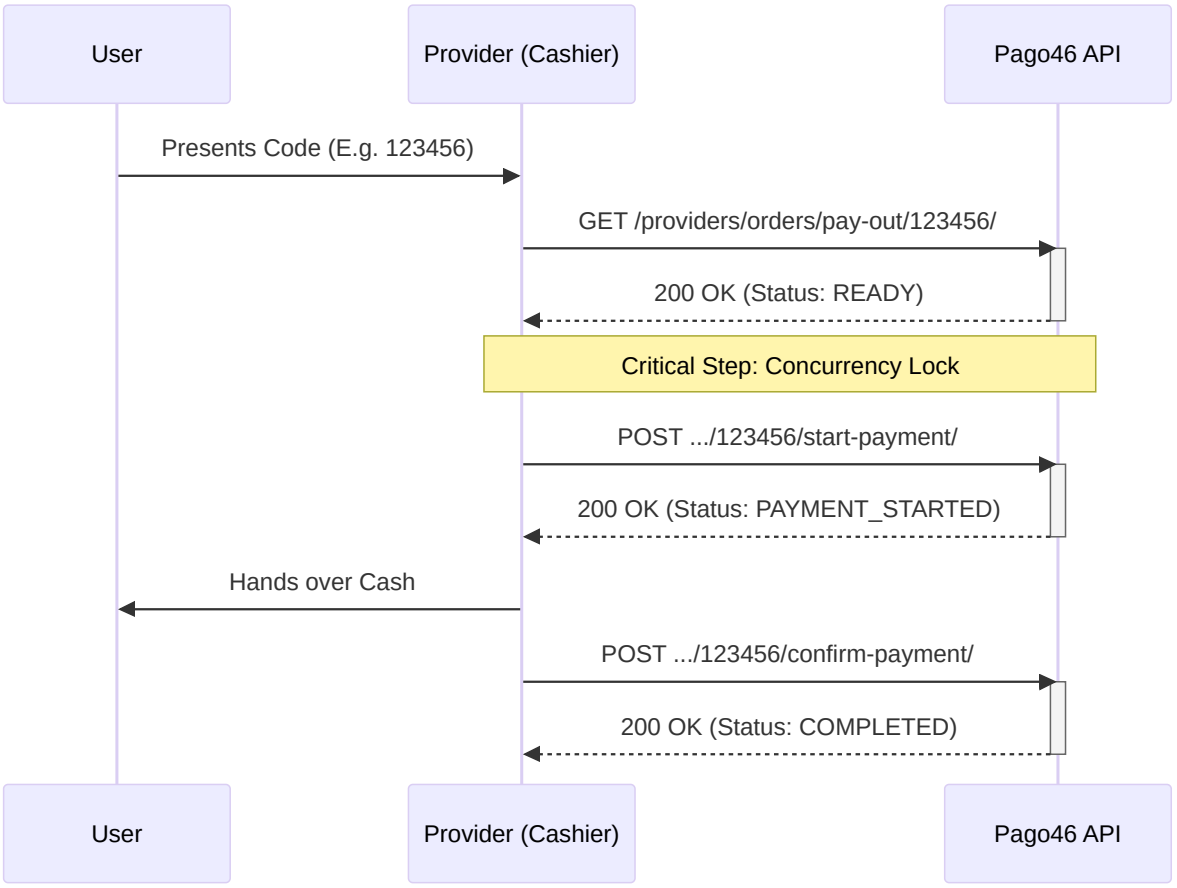
- `01`
- `oxxo_network`
- `seven_eleven_mx`
- `farmacia_abc`
- `network_123`

> **CONTACT**
>
> To register your payment networks, contact the Pago46 integration team. The IDs you provide will be the ones you must use in payment 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 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 provided by the end-user.

Request | **Response (200 OK) - Order Ready** | **Response (200 OK) - Order Already In Progress**

```
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>"
```

⚠ 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 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 simultaneously, they will receive an error indicating the order is already being processed.

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

Request | **Response (200 OK)** | **Error (409 Conflict)**

```
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"
  }'
```

Field	Type	Description
`network_id`	string	ID of the payment network used by the provider. **Required**.
`price`	string	Exact amount to be delivered. Must match the order's `price`. **Required**.
`price_currency`	string	Currency code (e.g., `MXN`, `USD`). Must match the order's `price_currency`. **Required**.

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 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 | Response (200 OK)

```
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"
  }'
```

Upon receiving this response, we will notify the end-user that their transaction has concluded successfully.

Cancellation (Optional)

If for any reason (insufficient funds in the register, operational error) you cannot complete the payment after having executed `start-payment`, you **must** release the order so it is not left locked indefinitely.

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

This will return the order to a state where (depending on business logic) it could be cancelled permanently or retried.

```
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 annulled.	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 and one shared Slack channel to improve internal 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.

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 facilitates readability in payment interfaces and mobile devices.

- **Use in Buttons:** Bold, **16px**, centered, letter-spacing: 0px.
- **Fallback:** In case **Lato** cannot be loaded, use sans-serif.

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

Base Logotypes

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

Version	Color	SVG	PNG
Isotype	Green	 **Download:** svg	 **Download:** sm · md · lg
	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 | Español

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

Green / Blue | Green / White | Blue / Green | Blue / White | White / Blue | White / 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

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 | **React (Tailwind)**

```
<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>
```

Restrictions and Best Practices

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 review that the button implementation complies with these guidelines to ensure the best conversion rate for your business.