ZenPays Documentation

Complete Documentation

SDK v0.6.0 · 215 documents
Edition 19 October 2026 · commit unknown
docs.zenpays.com

Contents

Getting Started
- Installation
- Quick Start
- Configuration
- Authentication

Concepts
- Payment flow

Guides
- Custom Checkout Integration
- Error Handling
- **Webhooks**
  - Webhooks Overview
  - Payment Webhooks
  - Payout Webhooks
  - Ramp Webhooks
  - KYC Webhooks
  - Other Webhooks
  - Signature Verification
- Testing
- TypeScript

SDK Reference
- Client
- Payments
- Customers
- Refunds
- Payouts
- Settlements
- Wallet
- Ledger
- Merchants
- Analytics
- **Ramp**
  - On-Ramp (Buy)
  - Off-Ramp
  - Ramp Intents
- Vendors
- Chargebacks
- **Billing**
  - Invoices

- Payment Links
- Subscriptions
- Authentication
- Security
- Tenant Hostnames

REST API

- REST API Overview
- REST API Authentication
- Error Handling

API objects

- The ApiKey object
- The BankAccount object
- The BatchPayout object
- The Chargeback object
- The CheckoutSession object
- The Commission object
- The Customer object
- The CustomerRiskProfile object
- The IPWhitelistEntry object
- The KycDocument object
- The LedgerBalance object
- The LedgerEntry object
- The MerchantLimits object
- The MerchantProfile object
- The PaymentIntent object
- The Payout object
- The PayoutIntent object
- The PayoutMethod object
- The RampOrder object
- The Refund object
- The Report object
- The ReportSchedule object
- The SecurityEvent object
- The Settlement object
- The StoredPaymentMethod object
- The Subscription object
- The SubscriptionPlan object
- The Transaction object
- The WalletAdjustment object
- The WalletBalance object
- The WalletTransaction object
- The WebhookConfig object
- The WebhookDeliveryLog object

Statuses and enums
- Chargeback status and reasons
- Compliance and risk
- Payment status and methods
- Payout status
- Refund status
- Report types and formats
- Security events
- Settlement status
- Subscription and billing
- Webhook events

Errors

Supported currencies

Payments
- Create Payment Intent
- Confirm Payment
- Get Payment Intent

Transactions
- List Transactions
- Get Transaction
- Export Transactions
- Get Customer Transactions

Customers
- Create Customer
- Get Customer
- List Customers
- Update Customer
- Get Customer History
- Get Customer Risk Profile
- Get Top Customers

Refunds
- Create Refund
- Get Refund
- List Refunds
- Cancel Refund
- Refund Statistics

Bulk Refunds
- Bulk Upload Refunds

Payout Intents
- Create Payout Intent
- Confirm Payout Intent
- Get Payout Intent
- List Payout Intents
- Cancel Payout Intent

Settlements
- Create Settlement
- Get Settlement
- List Settlements
- Cancel Settlement
- Settlement Statistics

Bank Accounts
- Add Bank Account
- List Bank Accounts
- Update Bank Account
- Delete Bank Account

Accounting
- Get Wallet Balances
- Get Currency Balance
- Get Wallet Transactions
- Get Wallet Summary

Ledger
- Get Ledger Overview
- Get Ledger Entries
- Get Ledger Balances
- Get Ledger Accounts
- Get Ledger Reserves
- Get Daily Ledger Summary
- Get Daily Ledger Summaries
- Get Ledger Reconciliation

Reports
- Generate Report
- List Reports
- Get Report
- Download Report
- Delete Report

Ramp
- Ramp Overview

Widget Integration (Recommended)
- Generate Widget URL

Ramp Intents
- Create Ramp Intent
- List Ramp Intents
- Get Ramp Intent
- Get KYC Status
- Update Ramp Intent Status

- Get Transaction Status

API Integration (Advanced)

On-Ramp (Buy)

- Get Buy Quotes
- Get Buy Rates
- Get Buy Prices
- Create Buy Checkout

Off-Ramp (Sell)

- Get Sell Quotes
- Get Sell Rates
- Create Sell Checkout
- Create Sell Deposit (Phase 1)
- Get Payout Preview (Phase 2a)
- Submit Sell Payout (Phase 2b)

Configuration

- Get Ramp Config
- Get Supported Assets
- Get Payment Methods
- Get Defaults
- Wallet Addresses

Vendors

- Create Vendor
- List Vendors
- Get Vendor
- Update Vendor
- Delete Vendor
- Vendor Stats
- Update Vendor Status
- Update Vendor KYC
- Vendor Commissions
- Vendor Referrals
- Create Vendor Referral
- Vendor Payout
- Vendor Analytics

Chargebacks

- Create Chargeback
- List Chargebacks
- Get Chargeback
- Chargeback Stats
- Chargeback Analytics
- Submit Evidence
- Add Response

Bulk Chargebacks

- Bulk Upload Chargebacks
- Bulk Upload Status

Billing

Products

- Create Product
- List Products
- Get Product
- Update Product
- Delete Product
- List Categories
- Manage Categories

Invoices

- Create Invoice
- List Invoices
- Get Invoice
- Update Invoice
- Send Invoice
- Cancel Invoice
- Invoice Analytics
- Public Invoice
- Pay Invoice
- Generate Payment Link

Payment Links

- Create Payment Link
- List Payment Links
- Get Payment Link
- Update Payment Link
- Deactivate Payment Link
- Payment Link Analytics
- Resolve Short Code
- Pay Payment Link
- Linked Invoices

Examples

- Basic Payment
- Checkout Flow
- On-Ramp (Buy) Flow
- Off-Ramp (Sell) Flow
- Payout Flow
- Refund Flow

Resources

- Postman Collection
- Using ZenPays docs with AI tools

PART

Getting Started

4 documents

GETTING STARTED

Installation

Get started with the ZenPays SDK in your project.

Requirements

JAVASCRIPT

Node.js 18.0 or later, plus npm, yarn, or pnpm

PYTHON

Python 3.8 or later, plus pip

Install the package

JAVASCRIPT

NPM

```
npm install @zenxdigitalholdings/zenpays
```

YARN

```
yarn add @zenxdigitalholdings/zenpays
```

PNPM

```
pnpm add @zenxdigitalholdings/zenpays
```

PYTHON

```
pip install zenpays
```

TypeScript Support

JAVASCRIPT

The SDK is written in TypeScript and includes complete type definitions out of the box. No additional `@types/*` package is needed.

```
import type { PaymentIntent } from '@zenxdigitalholdings/zenpays'
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_API_KEY!,
})

// Full type safety
const intent: PaymentIntent = await zenpays.payments.getPaymentIntent('pi_xxx')
```

PYTHON

The Python SDK includes full type hints and works seamlessly with mypy and other type checkers.

```python
from zenpays import ZenPays
from zenpays.types.payment import PaymentIntent
import os

zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

# Full type hints support
intent: PaymentIntent = zenpays.payments.get_payment_intent("pi_xxx")
```

Browser Support

JAVASCRIPT

The SDK works in both Node.js and browser environments. For browser usage, you can import directly:

```javascript
<script type="module">
  import { ZenPays } from 'https://unpkg.com/@zenxdigitalholdings/zenpays@latest/dist/index.mjs'

  const zenpays = new ZenPays({
    apiKey: 'your-publishable-key',
  })
</script>
```

> **Warning**
> Never expose your secret API key in client-side code. Use publishable keys for browser environments.

PYTHON

The Python SDK is designed for server-side use and is not compatible with browser environments.

> **Tip**
> Use the JavaScript SDK for browser-based integrations and the Python SDK for server-side operations.

Next Steps

- Quick Start - Create your first payment
- Configuration - Learn about SDK options
- Authentication - Manage API keys

GETTING STARTED

Quick Start

Learn how to create your first payment with the ZenPays SDK.

Initialize the Client

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({
  apiKey: 'your-api-key',
})
```

PYTHON

```
from zenpays import ZenPays

zenpays = ZenPays(api_key="your-api-key")
```

Create a Payment Intent

A payment intent represents a single payment flow. Create one to start accepting payments:

JAVASCRIPT

```
const paymentIntent = await zenpays.payments.createPaymentIntent({
  amount: 100,
  currency: 'INR',
  paymentMethod: 'upi',
  customerEmail: 'john@example.com',
  customerFirstName: 'John',
  customerLastName: 'Doe',
  customerCountry: 'IN',
  description: 'Premium subscription',
})

console.log('Payment Intent ID:', paymentIntent.intentId)
console.log('Payment Page URL:', paymentIntent.paymentPageUrl)
```

PYTHON

```
payment_intent = zenpays.payments.create_payment_intent({
    "amount": 100,
    "currency": "INR",
    "paymentMethod": "upi",
    "customerEmail": "john@example.com",
    "customerFirstName": "John",
    "customerLastName": "Doe",
    "customerCountry": "IN",
    "description": "Premium subscription",
})

print("Payment Intent ID:", payment_intent["intentId"])
print("Payment Page URL:", payment_intent["paymentPageUrl"])
```

Confirm the Payment

Once the customer provides payment details, confirm the payment:

JAVASCRIPT

```javascript
const result = await zenpays.payments.confirmPayment(paymentIntent.intentId, {
  customerDetails: {
    name: 'John Doe',
    email: 'john@example.com',
    address: { country: 'US' },
  },
  paymentMethodDetails: {
    type: 'upi',
    vpa: 'john@upi',
  },
})

if (result.status === 'succeeded') {
  console.log('Payment successful!')
}
else if (result.nextAction) {
  // Handle 3D Secure or other required actions
  console.log('Redirect to:', result.nextAction.redirectUrl)
}
```

PYTHON

```python
result = zenpays.payments.confirm_payment(
    payment_intent["intent_id"],
    {
        "customer_details": {
            "name": "John Doe",
            "email": "john@example.com",
            "address": {"country": "US"},
        },
        "payment_method_details": {
            "type": "upi",
            "vpa": "john@upi",
        },
    },
)

if result["status"] == "succeeded":
    print("Payment successful!")
elif "next_action" in result:
    # Handle 3D Secure or other required actions
    print("Redirect to:", result["next_action"]["redirect_url"])
```

Check Payment Status

JAVASCRIPT

```javascript
const intent = await zenpays.payments.getPaymentIntent(paymentIntent.intentId)

console.log('Status:', intent.status)
// 'pending' | 'processing' | 'succeeded' | 'failed' | 'cancelled'
```

PYTHON

```python
intent = zenpays.payments.get_payment_intent(payment_intent["intent_id"])

print("Status:", intent["status"])
# 'pending' | 'processing' | 'succeeded' | 'failed' | 'cancelled'
```

Handle Webhooks

Set up webhooks to receive real-time payment updates:

JAVASCRIPT

```javascript
// Register a webhook endpoint
await zenpays.merchants.createWebhook({
  url: 'https://your-site.com/webhooks/zenpays',
  events: [
    'payment.intent.succeeded',
    'payment.intent.failed',
    'refund.completed',
  ],
})
```

PYTHON

```python
# Register a webhook endpoint
zenpays.merchants.create_webhook({
    "url": "https://your-site.com/webhooks/zenpays",
    "events": [
        "payment.intent.succeeded",
        "payment.intent.failed",
        "refund.completed",
    ],
})
```

Error Handling

The SDK throws specific error types for different scenarios:

JAVASCRIPT

```javascript
import { PaymentError, ValidationError, ZenPaysError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.createPaymentIntent({
    amount: 1000,
    currency: 'USD',
  })
}
catch (error) {
  if (error instanceof ValidationError) {
    console.error('Validation failed:', error.message)
  }
  else if (error instanceof PaymentError) {
    console.error('Payment failed:', error.message)
  }
  else if (error instanceof ZenPaysError) {
    console.error('API error:', error.message, error.code)
  }
}
```

PYTHON

```python
from zenpays.errors import PaymentError, ValidationError, ZenPaysError

try:
    zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })
except ValidationError as e:
    print(f"Validation failed: {e.message}")
except PaymentError as e:
    print(f"Payment failed: {e.message}")
except ZenPaysError as e:
    print(f"API error: {e.message} [{e.code}]")
```

Complete Example

Here's a complete example bringing it all together:

JAVASCRIPT

```
import { ZenPays, ZenPaysError } from '@zenxdigitalholdings/zenpays'

async function processPayment() {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  try {
    // 1. Create payment intent
    const intent = await zenpays.payments.createPaymentIntent({
      amount: 2999,
      currency: 'INR',
      paymentMethod: 'upi',
      customerEmail: 'jane@example.com',
      customerFirstName: 'Jane',
      customerLastName: 'Smith',
      customerCountry: 'IN',
      description: 'Pro Plan - Monthly',
    })

    // 2. Confirm with payment details
    const result = await zenpays.payments.confirmPayment(intent.intentId, {
      customerDetails: {
        name: 'Jane Smith',
        email: 'jane@example.com',
        address: { country: 'US' },
      },
      paymentMethodDetails: {
        type: 'upi',
        vpa: 'jane@upi',
      },
    })

    // 3. Handle result
    if (result.status === 'succeeded') {
      console.log('Payment completed successfully!')
      console.log('Transaction ID:', result.externalTransactionId)
    }
    else if (result.nextAction?.type === 'redirect') {
      console.log('3D Secure required, redirect to:', result.nextAction.redirectUrl)
    }
  }
  catch (error) {
    if (error instanceof ZenPaysError) {
      console.error(`Error [${error.code}]: ${error.message}`)
    }
    throw error
  }
}

processPayment()
```

PYTHON

```python
import os
from zenpays import ZenPays
from zenpays.errors import ZenPaysError


def process_payment():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    try:
        # 1. Create payment intent
        intent = zenpays.payments.create_payment_intent({
            "amount": 2999,
            "currency": "INR",
            "paymentMethod": "upi",
            "customerEmail": "jane@example.com",
            "customerFirstName": "Jane",
            "customerLastName": "Smith",
            "customerCountry": "IN",
            "description": "Pro Plan - Monthly",
        })

        # 2. Confirm with payment details
        result = zenpays.payments.confirm_payment(
            intent["intent_id"],
            {
                "customer_details": {
                    "name": "Jane Smith",
                    "email": "jane@example.com",
                    "address": {"country": "US"},
                },
                "payment_method_details": {
                    "type": "upi",
                    "vpa": "jane@upi",
                },
            },
        )

        # 3. Handle result
        if result["status"] == "succeeded":
            print("Payment completed successfully!")
            print("Transaction ID:", result["external_transaction_id"])
        elif result.get("next_action", {}).get("type") == "redirect":
            print("3D Secure required, redirect to:", result["next_action"]["redirect_url"])

    except ZenPaysError as error:
        print(f"Error [{error.code}]: {error.message}")
        raise


if __name__ == "__main__":
    process_payment()
```

Next Steps

- Configuration - Customize SDK behavior
- Payments API - Explore all payment methods
- Error Handling - Handle errors gracefully

GETTING STARTED

Configuration

Learn how to configure the ZenPays SDK for your environment.

Configuration Options

JAVASCRIPT

```javascript
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({
  // Required: Your API key
  apiKey: 'your-api-key',
  // here

  // Optional: API base URL (defaults to production)
  baseUrl: 'https://api.zenpays.com',

  // Optional: API version (defaults to 'v1')
  apiVersion: 'v1',

  // Optional: Request timeout in milliseconds (defaults to 30000)
  timeout: 30000,

  // Optional: Custom fetch implementation
  fetch: customFetch,
})
```

PYTHON

```python
from zenpays import ZenPays

zenpays = ZenPays(
    # Required: Your API key
    api_key="your-api-key",

    # Optional: API base URL (auto-detected from API key)
    base_url="https://api.zenpays.com",

    # Optional: Request timeout in seconds (defaults to 30)
    timeout=30,

    # Optional: HMAC secret salt for request signing
    secret_salt="your-secret-salt",
)
```

Environment Configuration

Production

JAVASCRIPT

```javascript
const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_LIVE_API_KEY!,
})
```

PYTHON

```python
import os

zenpays = ZenPays(api_key=os.environ["ZENPAYS_LIVE_API_KEY"])
```

Sandbox/Testing

JAVASCRIPT

```javascript
const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_TEST_API_KEY!,
  baseUrl: 'https://api.zenpays.com',
})
```

PYTHON

```python
import os

zenpays = ZenPays(
    api_key=os.environ["ZENPAYS_TEST_API_KEY"],
    base_url="https://api.zenpays.com",
)
```

Configuration Options Reference

JavaScript

Option	Type	Default	Description
`apiKey`	`string`	*Required*	Your ZenPays API key
`baseUrl`	`string`	`'https://api.zenpays.com'`	API base URL
`apiVersion`	`string`	`'v1'`	API version to use
`timeout`	`number`	`30000`	Request timeout in milliseconds
`fetch`	`typeof fetch`	`globalThis.fetch`	Custom fetch implementation

Python

Option	Type	Default	Description
`api_key`	`str`	*Required*	Your ZenPays API key
`base_url`	`str`	Auto-detected	API base URL
`timeout`	`int`	`30`	Request timeout in seconds
`secret_salt`	`str`	`None`	HMAC secret for request signing

Environment Auto-Detection:

- `zp_live_*` → `https://api.zenpays.com`
- `zp_test_*` → `https://api.zenpays.com` (sandbox; same host, routed by key)
- anything else → `http://localhost:3000`

Using Environment Variables

We recommend storing your API keys in environment variables:

JAVASCRIPT

```
# .env
ZENPAYS_API_KEY=zp_live_xxxxx
ZENPAYS_TEST_API_KEY=zp_test_xxxxx
```

```
// Use test key in development
const apiKey = process.env.NODE_ENV === 'production'
  ? process.env.ZENPAYS_API_KEY
  : process.env.ZENPAYS_TEST_API_KEY

const zenpays = new ZenPays({ apiKey: apiKey! })
```

PYTHON

```
# .env
ZENPAYS_API_KEY=zp_live_xxxxx
ZENPAYS_TEST_API_KEY=zp_test_xxxxx
```

```
import os

# Use test key in development
api_key = (
    os.environ["ZENPAYS_API_KEY"]
    if os.getenv("ENV") == "production"
    else os.environ["ZENPAYS_TEST_API_KEY"]
)

zenpays = ZenPays(api_key=api_key)
```

Timeout Configuration

Adjust timeouts for different scenarios:

JAVASCRIPT

```
// Longer timeout for batch operations
const zenpays = new ZenPays({
  apiKey: 'your-api-key',
  timeout: 120000, // 2 minutes
})
```

PYTHON

```python
# Longer timeout for batch operations
zenpays = ZenPays(
    api_key="your-api-key",
    timeout=120,  # 2 minutes (in seconds)
)
```

Multiple Instances

Create multiple client instances for different environments or merchants:

JAVASCRIPT

```javascript
// Production client
const prodClient = new ZenPays({
  apiKey: process.env.ZENPAYS_LIVE_KEY!,
})

// Sandbox client for testing
const testClient = new ZenPays({
  apiKey: process.env.ZENPAYS_TEST_KEY!,
  baseUrl: 'https://api.zenpays.com',
})

// Use the appropriate client based on context
const client = isTestMode ? testClient : prodClient
```

PYTHON

```python
import os

# Production client
prod_client = ZenPays(api_key=os.environ["ZENPAYS_LIVE_KEY"])

# Sandbox client for testing
test_client = ZenPays(
    api_key=os.environ["ZENPAYS_TEST_KEY"],
    base_url="https://api.zenpays.com",
)

# Use the appropriate client based on context
client = test_client if is_test_mode else prod_client
```

Context Manager Support

JAVASCRIPT

The JavaScript SDK doesn't require explicit cleanup, but you can implement similar patterns if needed.

PYTHON

The Python SDK supports context managers for automatic resource cleanup:

```python
with ZenPays(api_key="your-api-key") as zenpays:
    # Perform operations
    intent = zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })
    # HTTP session automatically closed after the block
```

Type Safety

JAVASCRIPT

The SDK is fully typed. Get IntelliSense and type checking:

```javascript
import type { ZenPaysConfig } from '@zenxdigitalholdings/zenpays'
import { ZenPays } from '@zenxdigitalholdings/zenpays'

// Type-safe configuration
const config: ZenPaysConfig = {
  apiKey: 'your-api-key',
  timeout: 30000,
}

const zenpays = new ZenPays(config)
```

PYTHON

The SDK includes comprehensive type hints for IDE support:

```python
from zenpays import ZenPays
from zenpays.types.payment import PaymentIntent

zenpays = ZenPays(api_key="your-api-key")

# Type checkers understand the return types
intent: PaymentIntent = zenpays.payments.get_payment_intent("pi_xxx")
```

HMAC Request Signing

JAVASCRIPT

HMAC signing is not currently supported in the JavaScript SDK.

PYTHON

For enhanced security, enable HMAC request signing:

```python
zenpays = ZenPays(
    api_key="your-api-key",
    secret_salt="your-secret-salt",  # Get this from your dashboard
)

# All requests will now include an X-Signature header
```

This adds an HMAC-SHA256 signature to each request, providing an additional layer of security.

Next Steps

- Authentication - Learn about API key management

- Error Handling - Handle errors gracefully
- Testing - Test your integration

GETTING STARTED

Authentication

Learn how to manage API keys and authenticate with the ZenPays API.

API Key Types

ZenPays uses two types of API keys:

Key Type	Prefix	Usage
Live	`zp_live_`	Production payments
Test	`zp_test_`	Sandbox testing

Getting Your API Key

1. Log in to your ZenPays Dashboard
2. Navigate to **Settings** → **API Keys**
3. Copy your API key

> **Warning**
> Keep your API keys secure. Never expose them in client-side code or commit them to version control.

Using the SDK

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_API_KEY!,
})
```

PYTHON

```
import os
from zenpays import ZenPays

zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])
```

Environment-Based Authentication

JAVASCRIPT

```javascript
// Determine environment from API key prefix
function getEnvironment(apiKey: string): 'live' | 'test' {
  return apiKey.startsWith('zp_live_') ? 'live' : 'test'
}

const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_API_KEY!,
})

console.log('Environment:', getEnvironment(process.env.ZENPAYS_API_KEY!))
```

PYTHON

```python
import os

# Determine environment from API key prefix
def get_environment(api_key: str) -> str:
    if api_key.startswith("zp_live_"):
        return "live"
    elif api_key.startswith("zp_test_"):
        return "test"
    return "dev"

zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

print(f"Environment: {get_environment(os.environ['ZENPAYS_API_KEY'])}")
```

IP Whitelisting

For enhanced security, whitelist IPs allowed to use your API keys:

JAVASCRIPT

```javascript
// List whitelisted IPs
const ips = await zenpays.merchants.listWhitelistedIPs()

// Add a new IP
await zenpays.merchants.addIPToWhitelist(
  '203.0.113.50',
  'Production Server'
)

// Remove an IP
await zenpays.merchants.removeIPFromWhitelist('ip_entry_id')
```

PYTHON

```python
# List whitelisted IPs
ips = zenpays.merchants.list_whitelisted_ips()

# Add a new IP
zenpays.merchants.add_ip_to_whitelist(
    "203.0.113.50",
    "Production Server"
)

# Remove an IP
zenpays.merchants.remove_ip_from_whitelist("ip_entry_id")
```

Two-Factor Authentication

Enable 2FA for additional security:

JAVASCRIPT

```javascript
// Setup 2FA
const setup = await zenpays.security.setup2FA()
console.log('Scan this QR code:', setup.qrCodeUrl)
console.log('Backup codes:', setup.backupCodes)

// Verify 2FA code
await zenpays.security.verify2FA({ code: '123456' })

// Check 2FA status
const status = await zenpays.security.is2FAEnabled()
console.log('2FA enabled:', status.enabled)
```

PYTHON

```python
# Setup 2FA
setup = zenpays.security.setup_two_factor()
print(f"Scan this QR code: {setup['qr_code_url']}")
print(f"Backup codes: {setup['backup_codes']}")

# Verify 2FA code
zenpays.security.verify_two_factor({"code": "123456"})

# Check 2FA status
status = zenpays.security.is_two_factor_enabled()
print(f"2FA enabled: {status['enabled']}")
```

Best Practices

1. **Use environment variables** - Never hardcode API keys
2. **Rotate keys regularly** - Create new keys and revoke old ones
3. **Use minimal scopes** - Only grant necessary permissions
4. **Enable IP whitelisting** - Restrict access to known IPs
5. **Enable 2FA** - Add an extra layer of security
6. **Monitor usage** - Check API key usage in the dashboard

Handling Authentication Errors

JAVASCRIPT

```
import { AuthenticationError, AuthorizationError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.createPaymentIntent({
    amount: 1000,
    currency: 'USD',
  })
}
catch (error) {
  if (error instanceof AuthenticationError) {
    console.error('Invalid API key')
  }
  else if (error instanceof AuthorizationError) {
    console.error('API key lacks required permissions')
  }
}
```

PYTHON

```
from zenpays.errors import AuthenticationError, PermissionError

try:
    zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })
except AuthenticationError:
    print("Invalid API key")
except PermissionError:
    print("API key lacks required permissions")
```

Next Steps

- Quick Start - Create your first payment
- Security Guide (https://docs.zenpays.com/docs/guides/security) - Learn more about security best practices
- Webhooks - Verify webhook signatures

PART

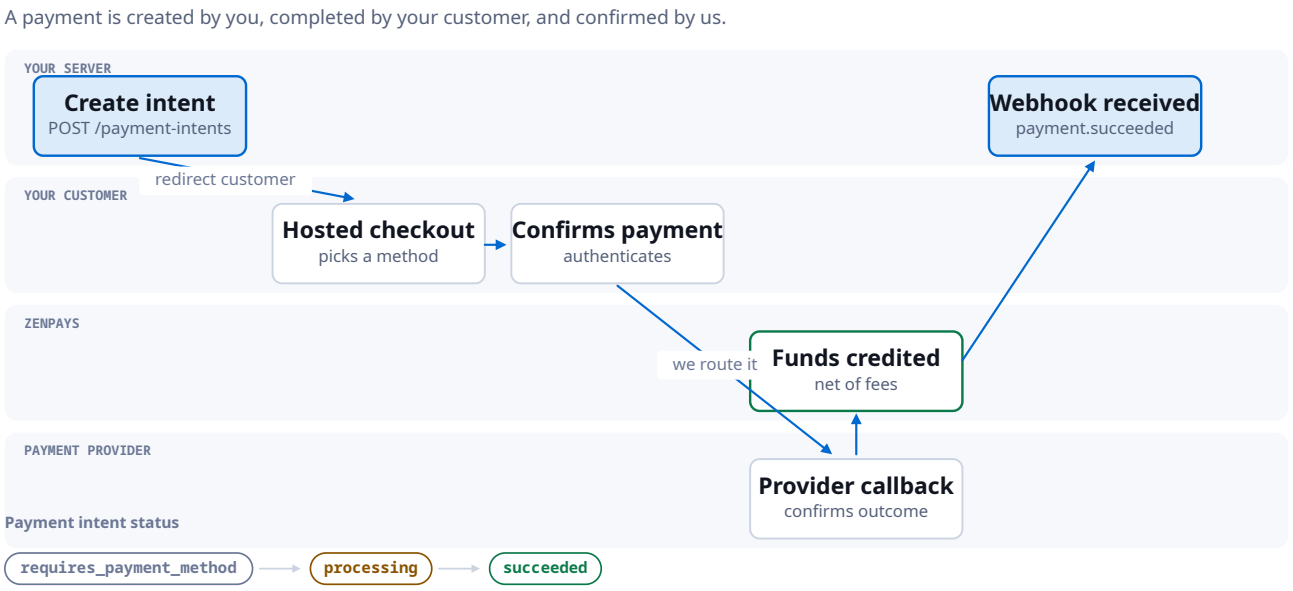
Concepts

1 documents

CONCEPTS

Payment flow

A ZenPays payment has three participants: your server, your customer, and us. You create an intent, your customer completes it, and we confirm the outcome and settle the funds.



How a payment moves from intent to settled funds

The steps

1. **You create a payment intent.** `POST /api/v1/payment-intents` with an amount and currency returns an intent in `requires_payment_method` and a hosted checkout URL. The intent expires after 20 minutes.
2. **Your customer completes it on the hosted checkout.** They choose a payment method from the options available for their currency and country.
3. **We route it.** When the checkout page loads payment options, we select a provider for that corridor. Which provider, and how it is chosen, is our concern — you never need to name one.
4. **The customer confirms.** The intent moves to `processing` under a lock, so a double submit cannot create two payments.
5. **The provider calls back.** We record the outcome on the transaction.
6. **Funds are credited.** Your wallet is credited net of fees *before* the intent is marked `succeeded` — so a `succeeded` intent always means the money reached your balance.
7. **You receive a webhook.** `payment.succeeded` (or `payment.failed`) is delivered to your endpoint. See Webhooks.

Status transitions

An intent starts at `requires_payment_method`, moves to `processing` on confirmation, and ends at `succeeded`.

> **There is no failed intent status**
>
> A payment that fails returns the intent to `requires_payment_method` so the customer can try a different method. The intent only leaves that state for `succeeded`, or for `canceled` when it expires or is cancelled outright.
>
> Judge a payment by its **transaction** status, or by the webhook you receive — not by waiting for an intent status that never arrives.

The transaction underneath the intent carries the detail: `initiated` → `processing` → `success` or `failed`, plus states for the cases in between such as `waiting_bank` and `waiting_customer`.

What to build against

Treat the webhook as the source of truth, not the redirect. A customer can close the tab before being returned to your site, and the payment still succeeds — the webhook is what tells you.

Poll `GET /api/v1/payment-intents/{id}` only as a fallback if you have not received a webhook.

PART

Guides

11 documents

GUIDES

Custom Checkout Integration

Build your own checkout page using ZenPays APIs. Instead of redirecting customers to the ZenPays hosted checkout, you can collect payment details on your own page and confirm the payment via API.

Flow Overview

```
1. Your Server          2. Your Checkout Page       3. Your Checkout Page       4. Customer Browser
   Create Intent   ──►     Fetch Intent Details  ──►    Confirm Payment     ──►    Redirect to TSP
   (server-side)           (client-side)               (client-side)               (payment page)
```

Step-by-step:

1. **Create Payment Intent** (server-side) — Your backend creates a payment intent via API with the amount, currency, and customer info.
2. **Fetch Payment Intent** (client-side) — Your checkout page fetches the intent details to get supported payment methods, countries, and channel limits.
3. **Confirm Payment** (client-side) — After the customer selects a payment method and fills in details, your page sends a confirm request.
4. **Redirect** — The confirm response returns a `redirectUrl`. Redirect the customer there to complete payment with the TSP (payment provider).
5. **Webhook** — ZenPays sends a webhook to your server when the payment succeeds or fails.

Base URL

```
https://api.zenpays.com/payment/api/v1
```

---

1. Create Payment Intent

Creates a payment intent on your server. This should be called from your **backend** (not the browser) since it requires your Merchant ID.

Request

```
POST /payment/api/v1/payment-intent
```

Headers:

Header	Required	Description
`X-Merchant-ID`	Yes	Your merchant ID
`Content-Type`	Yes	`application/json`
`X-Request-ID`	No	Unique request ID for idempotency

Body:

Field	Type	Required	Description
`amount`	number	Yes	Payment amount (minimum: 0.01)
`currency`	string	Yes	Currency code (e.g. `INR`, `USD`, `BRL`)
`customerEmail`	string	Yes*	Customer email (*required if no `customerId`)
`customerPhone`	string	No	Customer phone number
`customerFirstName`	string	No	Customer first name
`customerLastName`	string	No	Customer last name
`customerCountry`	string	No	ISO 3166-1 alpha-2 country code (e.g. `IN`)
`customerId`	string	No	Existing customer ID (skip email if provided)
`paymentMethod`	string	No	Preferred payment method
`description`	string	No	Payment description (max 500 chars)
`returnUrl`	string	No	URL to redirect customer after payment
`metadata`	object	No	Custom key-value data
`requestId`	string	No	Unique request ID (10-64 chars)

Example Request

```
{
  "amount": 500,
  "currency": "INR",
  "customerEmail": "customer@example.com",
  "customerPhone": "+911234567890",
  "customerFirstName": "John",
  "customerLastName": "Doe",
  "customerCountry": "IN",
  "description": "Order #12345",
  "returnUrl": "https://yoursite.com/payment-complete",
  "metadata": {
    "orderId": "order_12345"
  }
}
```

Example Response


```
{
  "success": true,
  "data": {
    "intentId": "pi_1775052514718_yn6v3nc47",
    "merchantId": "ZP_FIN_1774592804_0781001264",
    "amount": 500,
    "currency": "INR",
    "status": "requires_payment_method",
    "description": "Order #12345",
    "expiresAt": "2026-04-01T11:00:04.296Z",
    "processingFee": 0,
    "createdAt": "2026-04-01T10:50:04.299Z",
    "paymentPageUrl": "https://checkout.zenpays.com/pay/pi_1775052514718_yn6v3nc47?
token=pi_1775052514718_secret_abc123"
  },
  "message": "Payment intent created successfully",
  "error": null,
  "meta": {
    "request_id": "req_1775052514718_abc123",
    "timestamp": "2026-04-01T10:50:04.500Z",
    "processing_time_ms": 200,
    "api_version": "v1",
    "endpoint": "POST /payment-intent"
  }
}
```


Save the `intentId` — you'll need it for the next steps.

---

2. Fetch Payment Intent (Public)

Fetch the payment intent details on your checkout page. This is a **public endpoint** — no authentication required. Use this to render supported payment methods and validate amounts.

Request

```
GET /payment/api/v1/payment-intent/public/{intentId}
```

No headers required.

Example Response


```
{
  "success": true,
  "data": {
    "intentId": "pi_1775052514718_yn6v3nc47",
    "merchantId": "ZP_FIN_1774592804_0781001264",
    "merchantName": "Your Store",
    "merchantLogo": "https://cdn.example.com/logo.png",
    "amount": "500.00000000",
    "currency": "INR",
    "description": "Order #12345",
    "status": "requires_payment_method",
    "expiresAt": "2026-04-01T11:00:04.296Z",
    "isExpired": false,
    "timeRemaining": 598,
    "customerEmail": "customer@example.com",
    "customerPhone": "+911234567890",
    "supportedCountries": ["IN", "BR", "HK"],
    "supportedPaymentMethods": ["UPI", "UPIQR-H5", "BANK_TRANSFER"],
    "channelLimits": {
      "INR": {
        "UPI": { "min": 100, "max": 50000 },
        "UPIQR-H5": { "min": 500, "max": 50000 }
      }
    },
    "selectedTsp": "sulifu_pay",
    "processingFee": 0,
    "createdAt": "2026-04-01T10:50:04.299Z"
  },
  "message": "Payment intent details retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_1775052515000_xyz789",
    "timestamp": "2026-04-01T10:50:05.000Z",
    "processing_time_ms": 372,
    "api_version": "v1",
    "endpoint": "GET /payment-intent/public/pi_1775052514718_yn6v3nc47",
    "user_type": "merchant"
  }
}
```

Key Fields for Your Checkout UI

Field	How to Use
`supportedPaymentMethods`	Render only these payment methods as options. Values like `"UPI"`, `"UPIQR-H5"`, `"BANK_TRANSFER"`, `"CARD"`, `"PIX"`.
`channelLimits`	Validate the payment amount against min/max per method. If the amount is outside the range, disable or hide that method.
`isExpired` / `timeRemaining`	If `isExpired` is `true`, show an error — the intent can no longer be paid. `timeRemaining` is in seconds.
`status`	Must be `"requires_payment_method"` to proceed. Any other status means the intent is already processing or completed.
`amount` / `currency`	Display the payment amount to the customer.
`merchantName` / `merchantLogo`	Display merchant branding on your checkout page.

Edge Cases to Handle

- **Expired intent**: If `isExpired` is `true`, show "Payment link expired" and don't allow confirmation.
- **Already paid**: If `status` is `"succeeded"` or `"processing"`, show the appropriate message.
- **Amount below channel limit**: If `channelLimits.INR.UPI.min` is 100 and the amount is 50, don't show UPI as an option.
- **UPI methods only for INR**: `UPI` and `UPIQR-H5` are only valid when currency is `INR`. Don't show them for other currencies.
- **No supported methods**: If `supportedPaymentMethods` is empty or none pass the channel limits filter, show an error.

---

3. Confirm Payment Intent

After the customer selects a payment method and fills in details, confirm the payment. This is a **public endpoint**.

Request

```
POST /payment/api/v1/payment-intent/{intentId}/confirm
```

Headers:

Header	Required	Description
`Content-Type`	Yes	`application/json`

Request Body Structure

```
{
  "customerDetails": {
    "name": "Customer Name",
    "email": "customer@example.com",
    "phone": "+911234567890",
    "address": {
      "country": "IN"
    }
  },
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "UPI"
  },
  "deviceInfo": {
    "userAgent": "Mozilla/5.0 ...",
    "browserInfo": "Mozilla/5.0 ...",
    "screenWidth": 1512,
    "screenHeight": 982,
    "deviceFingerprint": "unique_fingerprint_hash",
    "metadata": {
      "selectedPaymentMethod": "fiat",
      "selectedMethod": "upi",
      "specificPaymentMethod": "UPI",
      "customerCurrency": "INR"
    }
  },
  "confirmSource": "MERCHANT_SITE"
}
```

Field Reference

customerDetails (required):

Field	Type	Required	Description
`name`	string	Yes	Full customer name
`email`	string	Yes	Customer email
`phone`	string	No	Customer phone
`address.country`	string	No	ISO country code

paymentMethodDetails (required):

Field	Type	Description
`paymentMethod`	string	Always `"FIAT"` for fiat payments
`paymentType`	string	The payment type (see table below)

confirmSource (required for custom checkout):

Always set to `"MERCHANT_SITE"`. The value `"ZENPAY_CHECKOUT"` is reserved for the ZenPays hosted checkout page.

Payment Method Examples

UPI Payment

```
{
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "UPI",
    "upiId": ""
  }
}
```

The `upiId` field can be empty — the TSP payment page will collect it.

UPIQR-H5 Payment (QR Code)

```
{
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "UPIQR-H5"
  }
}
```

No additional fields needed. The TSP generates the QR code on their payment page.

Bank Transfer

```
{
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "BANK_TRANSFER",
    "bankCode": "dp_hdfcbank_in"
  }
}
```

Card Payment

```
{
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "CARD",
    "cardNumber": "4111111111111111",
    "expiryMonth": "12",
    "expiryYear": "2027",
    "cvv": "123",
    "cardHolderName": "John Doe"
  }
}
```

PIX (Brazil)

```
{
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "PIX"
  }
}
```

Payment Type Reference

paymentType	Currency	Extra Fields
`UPI`	INR	`upiId` (optional)
`UPIQR-H5`	INR	None
`BANK_TRANSFER`	INR	`bankCode` (optional)
`CARD`	Any	`cardNumber`, `expiryMonth`, `expiryYear`, `cvv`, `cardHolderName`
`PIX`	BRL	None
`NETBANKING`	INR	`bankCode` (optional)
`WALLET`	INR	`walletProvider` (optional)

Confirm Response

Example (UPI)


```
{
  "success": true,
  "data": {
    "intentId": "pi_1775052514718_yn6v3nc47",
    "redirectUrl": "https://cashier.ppco.lol?orderNo=C2026040119435142093857",
    "tspProvider": "sulifu_pay",
    "externalTransactionId": "zpd6a82e78cedc1775052830530",
    "status": "created",
    "message": "Payment confirmed successfully. Redirecting to TSP.",
    "requiresCardDetails": false,
    "supportedCurrencies": ["INR"],
    "supportedCountries": ["IN", "BR", "HK"],
    "transactionId": "txn_1775052514718_1_mng4mhqh_p9axu5"
  },
  "message": "Payment confirmation successful",
  "error": null,
  "meta": {
    "request_id": "req_1775052830084_b0c0htr5e",
    "timestamp": "2026-04-01T14:13:51.654Z",
    "processing_time_ms": 1569,
    "api_version": "v1",
    "endpoint": "POST /payment-intent/pi_1775052514718_yn6v3nc47/confirm",
    "user_type": "merchant"
  }
}
```


What to Do with the Response

1. **Check** `success` — If `false`, show the error message to the customer.
2. **Redirect** — If `redirectUrl` is present, redirect the customer's browser to that URL. This is the TSP's payment page where the customer completes the payment (enters UPI PIN, scans QR code, etc.).
3. **Save** `transactionId` — Store this for tracking and reconciliation.

```
const response = await fetch(`/payment/api/v1/payment-intent/${intentId}/confirm`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(confirmPayload),
});

const result = await response.json();

if (result.success && result.data.redirectUrl) {
  // Redirect customer to TSP payment page
  window.location.href = result.data.redirectUrl;
} else {
  // Show error
  alert(result.data?.message || result.error?.message || 'Payment failed');
}
```

Edge Cases and Error Handling

Payment Intent Errors

Scenario	How to Detect	What to Show
Intent expired	`isExpired: true` from GET response	"This payment link has expired. Please request a new one."
Intent already paid	`status: "succeeded"`	"This payment has already been completed."
Intent processing	`status: "processing"`	"This payment is currently being processed."
Intent not found	404 from GET endpoint	"Payment not found."

Confirm Errors

Scenario	How to Detect	What to Show
Amount below minimum	Error response with min amount message	"Minimum amount for UPI is 100 INR."
Amount above maximum	Error response with max amount message	"Maximum amount for UPI is 50,000 INR."
Invalid payment method	400 error	"Selected payment method is not supported."
TSP unavailable	500 error with routing failure	"Payment service temporarily unavailable. Please try again."

Channel Limits Validation

Before showing payment methods to the customer, filter out methods where the amount is outside the channel limits:

```
const intentData = await fetchPaymentIntent(intentId);
const { amount, currency, supportedPaymentMethods, channelLimits } = intentData;

const currencyLimits = channelLimits?.[currency.toUpperCase()];

const availableMethods = supportedPaymentMethods.filter(method => {
  if (!currencyLimits) return true;
  const limit = currencyLimits[method];
  if (!limit) return true;
  return amount >= limit.min && amount <= limit.max;
});

// Render only `availableMethods` as options
```

UPI is INR-Only

`UPI` and `UPIQR-H5` payment methods are only valid when the currency is `INR`. Don't show them for other currencies:

```
const UPI_METHODS = ['UPI', 'UPIQR-H5'];
const isINR = currency.toUpperCase() === 'INR';

const filteredMethods = availableMethods.filter(method => {
  if (UPI_METHODS.includes(method) && !isINR) return false;
  return true;
});
```

Webhooks

After the customer completes payment on the TSP page, ZenPays sends a webhook to your configured endpoint with the payment result.

See the Webhooks Guide for setup instructions and payload format.

Full Integration Example

```
// 1. Server-side: Create payment intent
const createResponse = await fetch('https://api.zenpays.com/payment/api/v1/payment-intent', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-Merchant-ID': 'your_merchant_id',
  },
  body: JSON.stringify({
    amount: 500,
    currency: 'INR',
    customerEmail: 'customer@example.com',
    returnUrl: 'https://yoursite.com/payment-complete',
  }),
});
const { data: { intentId } } = await createResponse.json();

// 2. Client-side: Fetch intent details
const intentResponse = await fetch(
  `https://api.zenpays.com/payment/api/v1/payment-intent/public/${intentId}`
);
const { data: intent } = await intentResponse.json();

if (intent.isExpired) {
  showError('Payment link expired');
  return;
}

// 3. Client-side: Render payment methods filtered by channel limits
const methods = filterMethodsByLimits(
  intent.supportedPaymentMethods,
  intent.channelLimits,
  intent.currency,
  intent.amount
);

// 4. Client-side: After customer selects UPI and submits
const confirmResponse = await fetch(
  `https://api.zenpays.com/payment/api/v1/payment-intent/${intentId}/confirm`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      customerDetails: {
        name: 'John Doe',
        email: 'customer@example.com',
        phone: '+911234567890',
        address: { country: 'IN' },
      },
      paymentMethodDetails: {
        paymentMethod: 'FIAT',
        paymentType: 'UPI',
        upiId: '',
      },
      deviceInfo: {
        userAgent: navigator.userAgent,
        screenWidth: window.screen.width,
        screenHeight: window.screen.height,
      },
      confirmSource: 'MERCHANT_SITE',
    }),
  }
);
```

```
const result = await confirmResponse.json();

// 5. Redirect to TSP payment page
if (result.success && result.data.redirectUrl) {
  window.location.href = result.data.redirectUrl;
}
```

UPI Native Link Handling (INR Payments)

For INR payments, the `redirectUrl` opens a TSP payment page that shows a QR code. On mobile, customers can't scan a QR code on the same device they're paying from. ZenPays provides tools to **extract the `upi://` deep link** from the QR code so you can:

- **Mobile**: Open the customer's UPI app directly (GPay, PhonePe, Paytm) with one tap
- **Desktop**: Show your own clean QR code + "Copy UPI Link" button

Quick Start: Poll ZenPays API

After confirming payment, poll the ZenPays API to get the extracted UPI link:

```
// After confirm returns redirectUrl for INR payment:
if (result.data.redirectUrl && currency === 'INR') {
  const isMobile = /Mobi|Android|iPhone/i.test(navigator.userAgent);

  // Extract cashierOrderNo from redirectUrl for faster lookup
  let cashierOrderNo = '';
  try {
    cashierOrderNo = new URL(result.data.redirectUrl).searchParams.get('orderNo') || '';
  } catch {}

  // Poll every 4 seconds (ZenPays extracts the QR server-side)
  const interval = setInterval(async () => {
    const params = new URLSearchParams();
    if (cashierOrderNo) params.set('cashierOrderNo', cashierOrderNo);
    params.set('payPageUrl', result.data.redirectUrl);

    const res = await fetch(
      `https://api.zenpays.com/payment/api/v1/payment-intent/${intentId}/upi-link?${params}`
    );
    const data = await res.json();

    if (data.success && data.data?.upiDeepLink) {
      clearInterval(interval);
      const upiLink = data.data.upiDeepLink;

      if (isMobile && upiLink.startsWith('upi://')) {
        // Mobile: open UPI app with brief delay
        setTimeout(() => { window.location.href = upiLink; }, 600);
      } else {
        // Desktop: show your own QR + copy button (don't redirect)
        showDesktopUpiUI(upiLink);
      }
    }
  }, 4000);

  // Stop polling after 60 seconds
  setTimeout(() => clearInterval(interval), 60000);
}
```

Full Guide

For complete implementation with multiple approaches (server-side headless browser extraction, frontend screen capture), code examples in JavaScript, Python, and PHP, and best practices, see the **UPI Native Link Guide** (https://docs.zenpays.com/docs/guides/upi-native-link-guide).

GUIDES

Error Handling

Learn how to handle errors in the ZenPays SDK.

Error Types

The SDK provides specific error classes for different scenarios:

Error Class	HTTP Status	Description
`ZenPaysError`	Various	Base error class
`AuthenticationError`	401	Invalid API key
`AuthorizationError`	403	Insufficient permissions
`NotFoundError`	404	Resource not found
`ValidationError`	400	Invalid input data
`RateLimitError`	429	Rate limit exceeded
`NetworkError`	-	Network connectivity issues
`PaymentError`	402	Payment processing failed
`ConfigurationError`	-	SDK misconfiguration

Basic Error Handling

JAVASCRIPT

```
import {
  AuthenticationError,
  NotFoundError,
  ValidationError,
  ZenPaysError,
} from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.createPaymentIntent({
    amount: 1000,
    currency: 'USD',
  })
}
catch (error) {
  if (error instanceof AuthenticationError) {
    console.error('Invalid API key. Check your credentials.')
  }
  else if (error instanceof ValidationError) {
    console.error('Validation failed:', error.message)
    console.error('Fields:', error.fields)
  }
  else if (error instanceof NotFoundError) {
    console.error('Resource not found')
  }
  else if (error instanceof ZenPaysError) {
    console.error(`API Error [${error.code}]: ${error.message}`)
    console.error('Status:', error.status)
    console.error('Details:', error.details)
  }
  else {
    throw error // Re-throw unexpected errors
  }
}
```

PYTHON

```python
from zenpays import ZenPays
from zenpays.errors import (
    AuthenticationError,
    NotFoundError,
    ValidationError,
    ZenPaysError,
)

try:
    zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })
except AuthenticationError:
    print("Invalid API key. Check your credentials.")
except ValidationError as e:
    print(f"Validation failed: {e}")
    print(f"Fields: {e.fields}")
except NotFoundError:
    print("Resource not found")
except ZenPaysError as e:
    print(f"API Error [{e.code}]: {e}")
    print(f"Status: {e.status}")
    print(f"Details: {e.details}")
except Exception as e:
    raise  # Re-raise unexpected errors
```

Error Properties

All ZenPays errors include:

```typescript
interface ZenPaysError extends Error {
  message: string // Human-readable message
  code?: string // Error code (e.g., 'VALIDATION_ERROR')
  status?: number // HTTP status code
  details?: string // Additional details
}
```

Rate Limiting

Handle rate limits gracefully:

JAVASCRIPT

```
import { RateLimitError } from '@zenxdigitalholdings/zenpays'

async function makeRequestWithRetry(fn: () => Promise<any>, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      return await fn()
    }
    catch (error) {
      if (error instanceof RateLimitError && i < maxRetries - 1) {
        const delay = error.retryAfter ?? (2 ** i * 1000)
        console.log(`Rate limited. Retrying in ${delay}ms...`)
        await new Promise(resolve => setTimeout(resolve, delay))
      }
      else {
        throw error
      }
    }
  }
}

// Usage
const intent = await makeRequestWithRetry(() =>
  zenpays.payments.createPaymentIntent({ amount: 1000, currency: 'USD' })
)
```

PYTHON

```
import time
from zenpays.errors import RateLimitError


def make_request_with_retry(fn, max_retries=3):
    for i in range(max_retries):
        try:
            return fn()
        except RateLimitError as e:
            if i < max_retries - 1:
                delay = e.retry_after if e.retry_after else (2 ** i * 1000) / 1000
                print(f"Rate limited. Retrying in {delay}s...")
                time.sleep(delay)
            else:
                raise


# Usage
intent = make_request_with_retry(
    lambda: zenpays.payments.create_payment_intent({"amount": 1000, "currency": "USD"})
)
```

Network Errors

Handle network connectivity issues:

JAVASCRIPT

```
import { NetworkError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.createPaymentIntent({
    amount: 1000,
    currency: 'USD',
  })
}
catch (error) {
  if (error instanceof NetworkError) {
    if (error.message === 'Request timeout') {
      console.error('Request timed out. Try again later.')
    }
    else {
      console.error('Network error. Check your connection.')
    }
  }
}
```

PYTHON

```
from zenpays.errors import NetworkError

try:
    zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })
except NetworkError as e:
    if str(e) == "Request timeout":
        print("Request timed out. Try again later.")
    else:
        print("Network error. Check your connection.")
```

Payment Errors

Handle payment-specific errors:

JAVASCRIPT

```
import { PaymentError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.confirmPayment('pi_xxx', {
    customerDetails: { ... },
    paymentMethodDetails: { ... },
  })
} catch (error) {
  if (error instanceof PaymentError) {
    console.error('Payment failed:', error.message)
    console.error('Intent ID:', error.paymentIntentId)
    // Show user-friendly message
  }
}
```

PYTHON

```python
from zenpays.errors import PaymentError

try:
    zenpays.payments.confirm_payment("pi_xxx", {
        "customer_details": { ... },
        "payment_method_details": { ... },
    })
except PaymentError as e:
    print(f"Payment failed: {e}")
    print(f"Intent ID: {e.payment_intent_id}")
    # Show user-friendly message
```

Validation Errors

Handle validation errors with field-level details:

JAVASCRIPT

```javascript
import { ValidationError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.customers.create({
    email: 'invalid-email',
    name: '',
  })
}
catch (error) {
  if (error instanceof ValidationError) {
    console.error('Validation errors:')
    if (error.fields) {
      for (const [field, messages] of Object.entries(error.fields)) {
        console.error(`  ${field}: ${messages.join(', ')}`)
      }
    }
  }
}
```

PYTHON

```python
from zenpays.errors import ValidationError

try:
    zenpays.customers.create({
        "email": "invalid-email",
        "name": "",
    })
except ValidationError as e:
    print("Validation errors:")
    if e.fields:
        for field, messages in e.fields.items():
            print(f"  {field}: {', '.join(messages)}")
```

Channel Limit Errors

Handle payment method amount limit violations:

JAVASCRIPT

```
import { ChannelLimitError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.confirmPayment('pi_xxx', {
    customerDetails: { ... },
    paymentMethodDetails: { ... },
  })
} catch (error) {
  if (error instanceof ChannelLimitError) {
    const { amount, currency, paymentMethod, minimumAmount, maximumAmount } = error.limits ?? {}
    console.error(`Amount ${amount} ${currency} is out of range for ${paymentMethod}`)
    console.error(`Allowed range: ${minimumAmount} - ${maximumAmount} ${currency}`)
    // Prompt user to adjust the amount or select a different payment method
  }
}
```

PYTHON

```
from zenpays.errors import ChannelLimitError

try:
    zenpays.payments.confirm_payment("pi_xxx", {
        "customer_details": { ... },
        "payment_method_details": { ... },
    })
except ChannelLimitError as e:
    limits = e.limits or {}
    print(f"Amount {limits.get('amount')} {limits.get('currency')} is out of range for
{limits.get('paymentMethod')}")
    print(f"Allowed range: {limits.get('minimumAmount')} - {limits.get('maximumAmount')}
{limits.get('currency')}")
    # Prompt user to adjust the amount or select a different payment method
```

Best Practices

1. **Always catch errors** - Don't let errors crash your application
2. **Use specific error types** - Check for specific errors before generic ones
3. **Log error details** - Include code, status, and details for debugging
4. **Show user-friendly messages** - Don't expose technical details to users
5. **Implement retry logic** - Handle transient errors automatically
6. **Monitor errors** - Track error rates in production

Example: Complete Error Handler

JAVASCRIPT

```
import {
  AuthenticationError,
  AuthorizationError,
  ChannelLimitError,
  NetworkError,
  NotFoundError,
  PaymentError,
  RateLimitError,
  ValidationError,
  ZenPaysError,
} from '@zenxdigitalholdings/zenpays'

function handleZenPaysError(error: unknown): {
  message: string
  shouldRetry: boolean
  retryDelay?: number
} {
  if (error instanceof AuthenticationError) {
    return { message: 'Invalid API key', shouldRetry: false }
  }

  if (error instanceof AuthorizationError) {
    return { message: 'Access denied', shouldRetry: false }
  }

  if (error instanceof NotFoundError) {
    return { message: 'Resource not found', shouldRetry: false }
  }

  if (error instanceof ValidationError) {
    return { message: error.message, shouldRetry: false }
  }

  if (error instanceof ChannelLimitError) {
    return { message: error.message, shouldRetry: false }
  }

  if (error instanceof RateLimitError) {
    return {
      message: 'Too many requests',
      shouldRetry: true,
      retryDelay: error.retryAfter ?? 5000,
    }
  }

  if (error instanceof NetworkError) {
    return { message: 'Network error', shouldRetry: true, retryDelay: 1000 }
  }

  if (error instanceof PaymentError) {
    return { message: 'Payment failed', shouldRetry: false }
  }

  if (error instanceof ZenPaysError) {
    return { message: error.message, shouldRetry: false }
  }

  return { message: 'An unexpected error occurred', shouldRetry: false }
}
```

PYTHON

```python
from typing import TypedDict, Optional
from zenpays.errors import (
    AuthenticationError,
    AuthorizationError,
    ChannelLimitError,
    NetworkError,
    NotFoundError,
    PaymentError,
    RateLimitError,
    ValidationError,
    ZenPaysError,
)


class ErrorResult(TypedDict):
    message: str
    should_retry: bool
    retry_delay: Optional[int]


def handle_zenpays_error(error: Exception) -> ErrorResult:
    if isinstance(error, AuthenticationError):
        return {"message": "Invalid API key", "should_retry": False, "retry_delay": None}

    if isinstance(error, AuthorizationError):
        return {"message": "Access denied", "should_retry": False, "retry_delay": None}

    if isinstance(error, NotFoundError):
        return {"message": "Resource not found", "should_retry": False, "retry_delay": None}

    if isinstance(error, ValidationError):
        return {"message": str(error), "should_retry": False, "retry_delay": None}

    if isinstance(error, ChannelLimitError):
        return {"message": str(error), "should_retry": False, "retry_delay": None}

    if isinstance(error, RateLimitError):
        return {
            "message": "Too many requests",
            "should_retry": True,
            "retry_delay": error.retry_after if hasattr(error, "retry_after") and
error.retry_after else 5000,
        }

    if isinstance(error, NetworkError):
        return {"message": "Network error", "should_retry": True, "retry_delay": 1000}

    if isinstance(error, PaymentError):
        return {"message": "Payment failed", "should_retry": False, "retry_delay": None}

    if isinstance(error, ZenPaysError):
        return {"message": str(error), "should_retry": False, "retry_delay": None}

    return {"message": "An unexpected error occurred", "should_retry": False, "retry_delay": None}
```

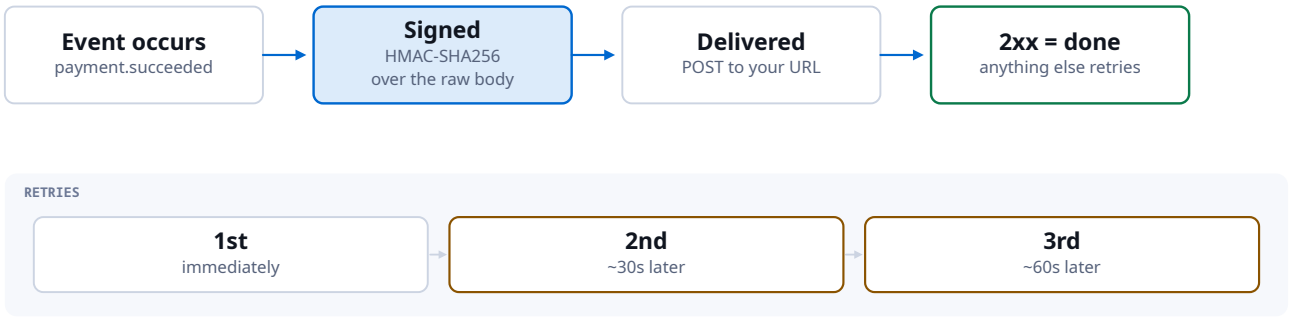
Webhooks

GUIDES / WEBHOOKS

Webhooks Overview

Receive real-time notifications about payment events.

Every webhook is signed over the exact bytes we send, and retried up to three times.

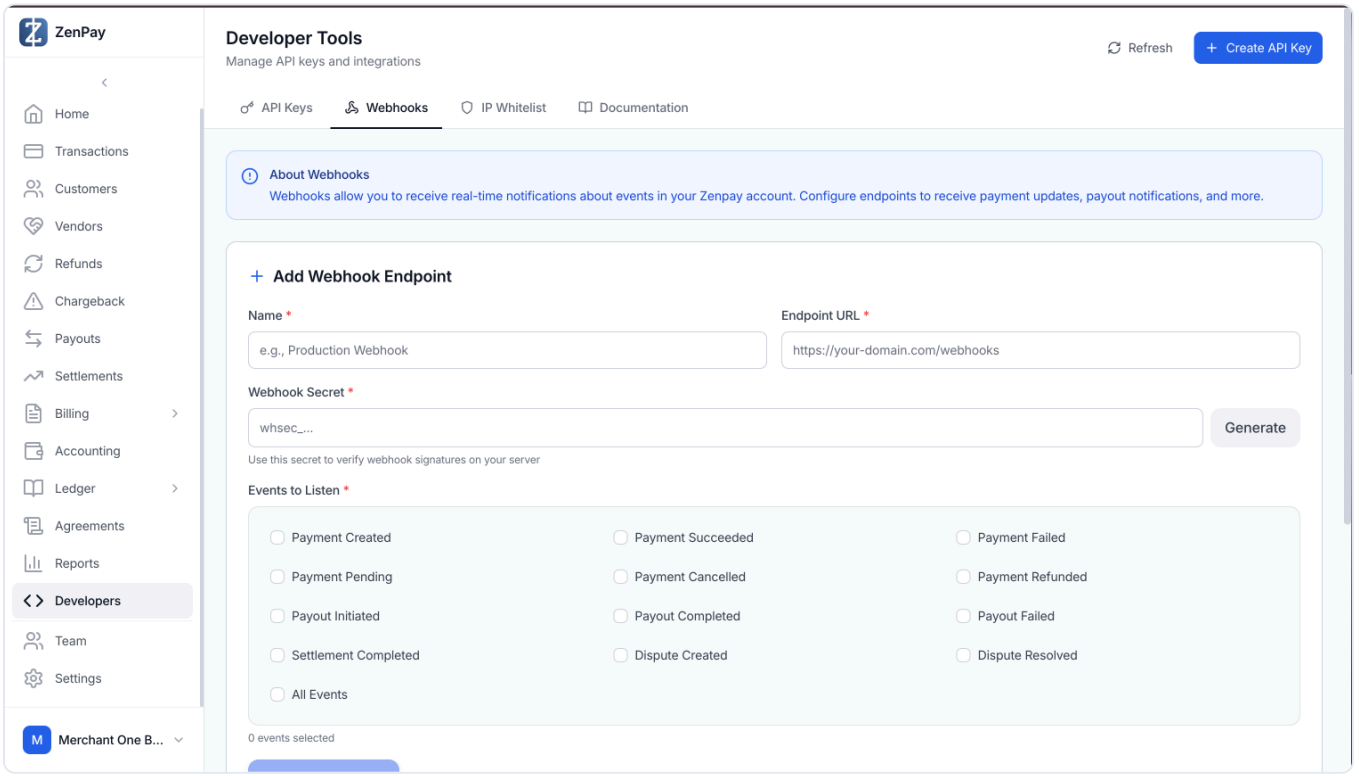


After three failed attempts delivery stops. Replay it from the dashboard once your endpoint is healthy.
Verify the signature against the raw request body, before any JSON parsing or re-serialising.

How webhooks are signed, delivered and retried

Creating a Webhook

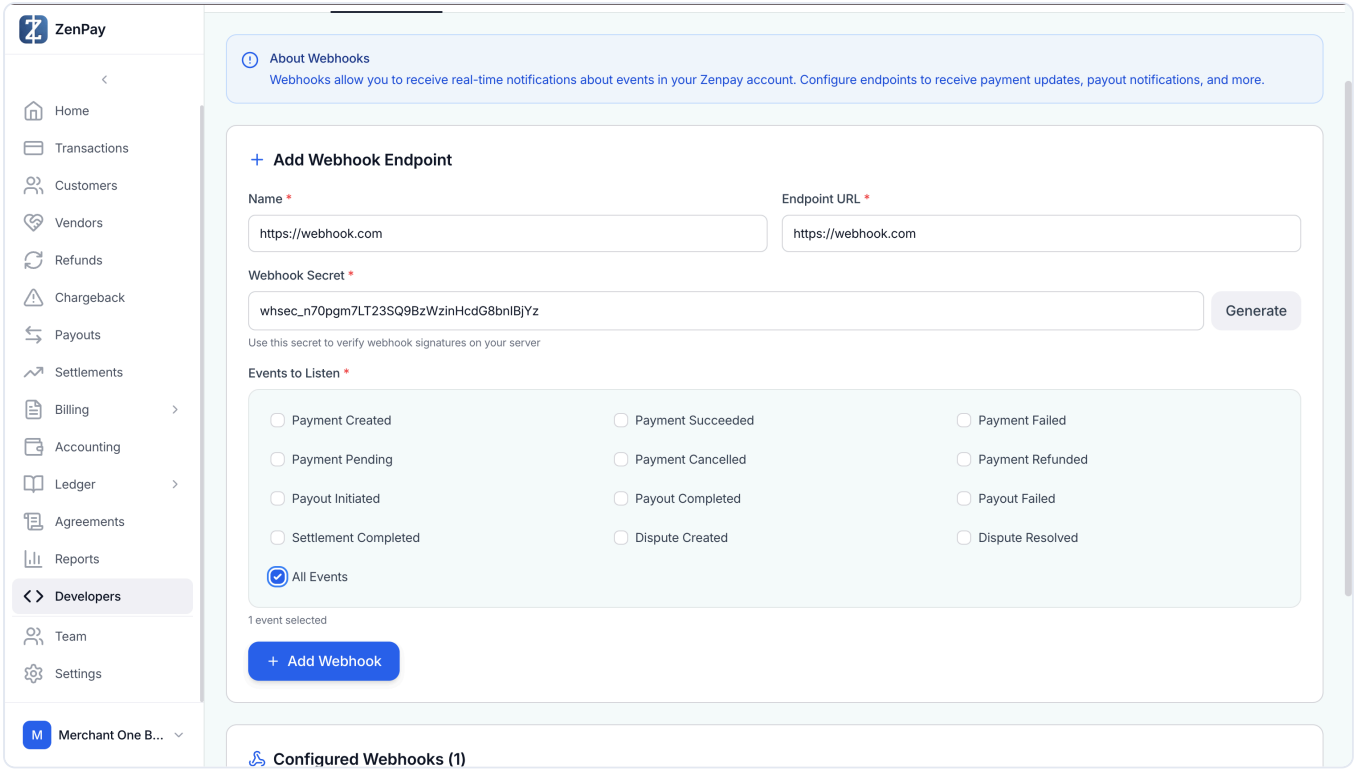
1. Log in to your **Merchant Dashboard** and navigate to **Developer Tools → Webhooks** tab.



2. Fill in the **Name** and **Endpoint URL** fields with your server's webhook URL (e.g. `https://your-domain.com/webhooks/zenpays`).
3. Click **Generate** to create a **Webhook Secret**. Copy and store this secret securely on your server — you'll need it to verify webhook signatures.

4. Select the **Events to Listen** to. You can pick individual events or select **All Events** to receive everything.

5. Click **+ Add Webhook** to save.



6. Your endpoint will appear under **Configured Webhooks**. You can add multiple webhook endpoints to receive events at different URLs.

Testing Webhooks

To test your webhook integration:

1. Create a webhook on the dashboard (see above).
2. Process a test payment transaction through the checkout flow.
3. Check your server logs to verify the webhook was received with the correct payload and signature.

Delivery attempts and status are visible in the dashboard under each configured webhook endpoint.

GUIDES / WEBHOOKS

Payment Webhooks

Payment webhook events track the lifecycle of a payment from initiation to completion or failure.

Events

Event	Description
`payment.success`	Payment completed successfully
`payment.failed`	Payment failed
`payment.pending`	Payment is pending
`payment.cancelled`	Payment was cancelled

Payload

```
{
  "event_type": "payment.success",
  "payment_data": {
    "merchant_id": "m_abc123",
    "payment_id": "pi_xyz789",
    "transaction_id": "txn_def456",
    "order_id": "pi_xyz789",
    "amount": 100.00,
    "currency": "USD",
    "status": "succeeded",
    "payment_method": "credit_card",
    "processed_at": "2026-02-28T10:30:00.000Z"
  }
}
```

For failed payments, an additional `failure_reason` field is included in `payment_data` along with an operational `code` (see Operational Errors) that maps to a friendly customer-facing message.

```
{
  "event_type": "payment.failed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "payment_id": "pi_xyz789",
    "transaction_id": "txn_def456",
    "amount": 100.00,
    "currency": "USD",
    "status": "failed",
    "code": "TSP_PAYMENT_FAILED",
    "failure_reason": "Card declined by issuer",
    "processed_at": "2026-02-28T10:30:00.000Z"
  }
}
```

Payment Callback Lifecycle

Treat payment events as a state machine:

- `payment.pending` means the payment is still in progress.

- `payment.success` is the final success signal.
- `payment.failed` is an explicit final failure signal.

Incomplete payments can remain pending

Some payment channels keep incomplete orders in `processing` / `pending` until a terminal success is received. In these cases, your server may receive repeated or unchanged pending updates and may not receive an automatic failure callback.

Your integration should:

- keep the order as pending while waiting for a terminal event,
- mark success only on `payment.success`,
- mark failed only when `payment.failed` is explicitly delivered.

Recommended merchant handling

If no terminal callback arrives yet, continue showing the payment as pending/in progress. Do not auto-fail based only on timeout unless your own business timeout policy requires it.

Real-time WebSocket events

If you embed the ZenPays checkout (or subscribe to its `/payment-status` namespace directly), you'll also see these socket events alongside the HTTP webhooks above. They drive the live status UI in the customer's browser.

Event	When fired	Notable fields
`payment-update`	Generic status transition	`status`, optionally `rampOrderStatus`, `kycStatus`
`payment-success`	Intent moved to `succeeded`	`transactionId`, `timestamp`
`payment-failed`	Intent moved to terminal failed state	`code`, `error`, `timestamp` — the `code` is one of the operational error codes (e.g., `TSP_PAYMENT_FAILED`, `PROVIDER_UNAVAILABLE`, `PAYMENT_STUCK`). Use it to pick a friendly message; never display the raw `error` string.
`payment-stuck`	Intent has been in `processing` past ~90s with no transitions	`code`, `message`, `status: 'processing'` — **informational only**. The intent is still alive. Use this to swap a generic spinner for a "this is taking longer than usual" message; do NOT treat it as a terminal failure.

Example: handling `payment-failed` with codes

```
socket.on('payment-failed', (data: { intentId: string; code?: string; error?: string }) => {
  switch (data.code) {
    case 'NO_TSP_AVAILABLE':
    case 'UNSUPPORTED_REGION':
      showMessage("This payment route isn't available right now.");
      break;
    case 'UNSUPPORTED_CURRENCY_PAIR':
      showMessage("This currency combination isn't supported.");
      break;
    case 'TSP_PAYMENT_FAILED':
      showMessage("Your payment couldn't be completed. Please try a different method.");
      break;
    case 'PAYMENT_STUCK':
      // Hard timeout — friendly but explicit
      showMessage("We tried but couldn't complete this payment. Please try again later.");
      break;
    default:
      showMessage("Something went wrong. Please try again or contact support.");
  }
});
```

Example: handling `payment-stuck`

```
socket.on('payment-stuck', (data: { intentId: string; code: string; message: string }) => {
  // Swap the generic spinner for a friendlier "still working" UI.
  // Do NOT mark the payment as failed — it's still in progress.
  showWaitingCard({
    title: "This is taking longer than usual",
    description: "We're still trying. Feel free to wait or check back in your dashboard.",
  });
});
```

GUIDES / WEBHOOKS

Payout Webhooks

Payout webhook events track single payouts, payout intents, and batch payouts.

Single Payout Events

Event	Description
`payout.initiated`	Payout accepted by ZenPay and moved into processing (funds are reserved but the final outcome is not yet known)
`payout.completed`	Payout processing finished successfully and funds have been fully debited from the reserved balance
`payout.failed`	Payout processing did not complete successfully and any reserved funds are released or returned according to ZenPay rules
`payout.cancelled`	Payout was cancelled before processing completed

Payload


```
{
  "event_type": "payout.completed",
  "payment_data": {
    "payoutId": "payout_1774778823937_17298d89",
    "transactionId": "po6efbc8502da51774842226226",
    "intentId": "poi_mnblh1o5_e8190e32",
    "amount": 100,
    "currency": "INR",
    "status": "completed",
    "beneficiary": "Rajesh Kumar",
    "payoutType": "BANK_TRANSFER",
    "failureReason": null,
    "metadata": {
      "order_no": "ORD-2026-001",
      "customer_ref": "CUST-789"
    }
  }
}
```


Payload Fields

Field	Type	Description
`payoutId`	string	Internal payout ID
`transactionId`	string	External payout transaction reference — matches `externalPayoutId` from the confirm response
`intentId`	string | null	Payout intent ID (if created via the two-step intent flow)
`amount`	number	Payout amount
`currency`	string	Payout currency

Field	Type	Description
`status`	string	`initiated`, `completed`, `failed`, or `cancelled`
`beneficiary`	string	Beneficiary name
`payoutType`	string	Payout method (e.g. `BANK_TRANSFER`, `UPI`)
`failureReason`	string | null	Reason for failure (only on `payout.failed`)
`metadata`	object | null	Custom key-value pairs passed when creating the payout intent — returned as-is in all webhook callbacks

Linking payouts end-to-end

When using the two-step payout intent flow:

1. **Create intent** → get `intentId`
2. **Confirm intent** → get `intentId` + `payoutId` + `externalPayoutId`
3. **Webhook callback** → get `intentId` + `payoutId` + `transactionId`

The `transactionId` in the webhook equals `externalPayoutId` from the confirm response. All three IDs are present in both the confirm response and webhook, so you can always link them.

Payout Intent Events

Event	Description
`payout_intent.created`	Payout intent created
`payout_intent.confirmed`	Payout intent confirmed and submitted for processing
`payout_intent.succeeded`	Payout intent completed successfully
`payout_intent.failed`	Payout intent failed
`payout_intent.cancelled`	Payout intent cancelled
`payout_intent.expired`	Payout intent expired without confirmation

Payload

Shares the same payload shape as single payout webhooks:

```
{
  "event_type": "payout_intent.succeeded",
  "payment_data": {
    "intentId": "poi_mnblh1o5_e8190e32",
    "payoutId": "payout_1774778823937_17298d89",
    "transactionId": "po6efbc8502da51774842226226",
    "amount": 500,
    "currency": "INR",
    "status": "succeeded",
    "beneficiary": "Rajesh Kumar",
    "payoutType": "BANK_TRANSFER",
    "failureReason": null,
    "metadata": {
      "order_no": "ORD-2026-001"
    }
  }
}
```


Batch Payout Events

Event	Description
`batch_payout.created`	Batch payout created
`batch_payout.processing`	Batch payout is being processed
`batch_payout.completed`	All payouts in the batch completed successfully
`batch_payout.partially_completed`	Some payouts in the batch completed, others failed
`batch_payout.failed`	Batch payout failed
`batch_payout.cancelled`	Batch payout was cancelled

Payload


```
{
  "event_type": "batch_payout.completed",
  "payment_data": {
    "batchId": "bp_abc123",
    "totalPayouts": 50,
    "successCount": 48,
    "failedCount": 2,
    "totalAmount": 25000,
    "currency": "INR",
    "status": "completed",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

Payload Fields

Field	Type	Description
`batchId`	string	Batch payout ID
`totalPayouts`	number	Total payouts in the batch
`successCount`	number	Number of successful payouts
`failedCount`	number	Number of failed payouts
`totalAmount`	number	Total amount across all payouts
`currency`	string	Batch currency
`status`	string	`created`, `processing`, `completed`, `partially_completed`, `failed`, `cancelled`
`completedAt`	string | null	ISO 8601 completion timestamp

GUIDES / WEBHOOKS

Ramp Webhooks

Ramp webhook events notify you when a ramp intent (buy or sell) reaches a terminal state.

Events

Event	Description
`ramp.completed`	Ramp intent completed (buy: crypto sent to wallet, sell: fiat payout submitted)
`ramp.failed`	Ramp intent failed (crypto send or payout submission failed)

Payloads

The payload varies depending on the ramp direction (buy vs sell) and outcome.

Buy Ramp Completed

Crypto has been sent to the customer's wallet:

```
{
  "event_type": "ramp.completed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "buy",
    "status": "completed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "chain": "ethereum",
    "destinationAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28",
    "fireblocksTxId": "fb_tx_9a8b7c6d5e4f",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

Sell Ramp Completed

Fiat payout has been submitted to the customer's bank account:

```
{
  "event_type": "ramp.completed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "sell",
    "status": "completed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "usdt_tron",
    "amount": 20,
    "payoutId": "po_xyz789",
    "payoutAmount": 19.50,
    "payoutCurrency": "USD",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

Ramp Failed

Buy or sell flow encountered an error:

```
{
  "event_type": "ramp.failed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "buy",
    "status": "failed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "reason": "Crypto send failed",
    "failedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

Payload Fields

Field	Type	Present	Description
`merchant_id`	string	Always	Your merchant ID
`intentId`	string	Always	Ramp intent ID (`ri_...`)
`type`	string	Always	`"buy"` or `"sell"`
`status`	string	Always	`"completed"` or `"failed"`
`fiatCurrency`	string	Always	ISO currency code (e.g. `USD` , `EUR`)
`cryptoCurrency`	string	Always	Crypto identifier (e.g. `eth_ethereum` , `usdt_tron`)
`amount`	number	Always	Original intent amount
`completedAt`	string	On success	ISO 8601 completion timestamp

Field	Type	Present	Description
`failedAt`	string	On failure	ISO 8601 failure timestamp
`reason`	string	On failure	Human-readable failure reason
`chain`	string	Buy success	Blockchain network (e.g. `ethereum`, `tron`)
`destinationAddress`	string	Buy success	Wallet address crypto was sent to
`cryptoTxId`	string	Buy success	Crypto transaction reference
`payoutId`	string	Sell success	Payout reference ID
`payoutAmount`	number	Sell success	Fiat amount paid out
`payoutCurrency`	string	Sell success	Fiat currency of payout

GUIDES / WEBHOOKS

KYC Webhooks

KYC webhook events notify you when a customer's ZenPays KYC verification status changes. These are fired for customers going through identity verification in the ramp checkout widget.

Events

Event	Description
`kyc.approved`	Customer's identity verification passed
`kyc.declined`	Customer's identity verification was rejected
`kyc.in_review`	Verification flagged for manual review
`kyc.expired`	Previous verification has expired

> **Note**
> Intermediate states like `pending`, `in_progress`, and `resubmitted` are **not** delivered as webhooks — they would be noise for most integrations. If you need that granularity, poll `GET /ramp-intents/:id/kyc-status` instead.

Payload

```
{
  "event_type": "kyc.approved",
  "user_id": "usr_abc123",
  "kyc_status": "approved",
  "aml_status": "clear",
  "decline_reason": null,
  "merchant_id": "mer_abc123",
  "affected_intent_ids": ["ri_1710345678000_a1b2c3d4e5f6g7h8"],
  "didit_session_id": "8a9b2c1d-e3f4-...",
  "occurred_at": "2026-03-16T14:30:00.000Z"
}
```

Payload Fields

Field	Type	Description
`event_type`	string	The KYC event: `kyc.approved`, `kyc.declined`, `kyc.in_review`, or `kyc.expired`
`user_id`	string	The `userId` you passed when creating the ramp intent
`kyc_status`	string	Verification state: `approved`, `declined`, `in_review`, or `expired`
`aml_status`	string | null	AML screening result: `clear`, `flagged`, `rejected`, `pending_review`, or `null` if not yet evaluated
`decline_reason`	string | null	Reason code when terminal: `kyc_rejected`, `kyc_expired`, or a provider-specific string. `null` for non-decline events.
`merchant_id`	string	Your merchant ID
`affected_intent_ids`	string[]	Ramp intents that this status change applied to. **May be empty** on late approvals — see "Multi-merchant fan-out & late approval" below.
`didit_session_id`	string | null	The underlying Didit verification session ID, for cross-referencing with our support team.
`occurred_at`	string	ISO 8601 timestamp of the status change

Multi-merchant fan-out & late approval

Two behaviors that surprise integrators if you don't model them:

1. **Multi-merchant fan-out.** A single customer can have ramp intents across multiple merchants. When their KYC status changes, every merchant that has any intent for that user receives the event independently. Each receives only the `affected_intent_ids` belonging to *their* merchant.
2. **Late approval after a prior decline.** If a user is declined today and approved tomorrow (e.g., after manual re-review), the original failed intents stay terminally `cancelled` — we do **not** auto-resume them. The merchant still receives `kyc.approved`, but with `affected_intent_ids: []`. Treat this as an instruction to update your **user verification cache** so future intents proceed without re-KYC, not as an event about a specific transaction.

Usage

KYC webhooks are useful for:

- **Updating user records** — mark a user as KYC-verified in your system when `kyc.approved` is received, so future ramp intents can pass `kycVerified: true`.

- **Handling rejections** — notify the customer or trigger support flows on `kyc.declined`. The `decline_reason` field tells you whether it was a hard rejection (`kyc_rejected`) or a session expiry (`kyc_expired`).
- **Monitoring review queues** — track `kyc.in_review` events for compliance reporting; pair with `aml_status` to know whether identity was approved but AML flagged for review.

Example

```
function handleWebhookEvent(event: any) {
  switch (event.event_type) {
    case 'kyc.approved':
      // Mark user as KYC verified in your database for future ramp intents
      await updateUserKycStatus(event.user_id, 'verified')

      if (event.affected_intent_ids.length > 0) {
        // The user had active intents that just unblocked — surface success
        // in any UI showing those intents.
        await notifyIntentsUnblocked(event.affected_intent_ids)
      } else {
        // Late approval after a prior decline — no active intent to resume.
        // Merchant may want to invite the customer to start a new transaction.
        await inviteRetry(event.user_id)
      }
      break

    case 'kyc.declined':
      // Hard rejection — notify the customer, offer support
      await notifyCustomer(event.user_id, 'KYC verification was not successful', {
        reason: event.decline_reason,
        affectedIntents: event.affected_intent_ids,
      })
      break

    case 'kyc.in_review':
      // Identity OK but flagged for review (often AML-driven)
      await flagForReview(event.user_id, {
        amlStatus: event.aml_status,
        affectedIntents: event.affected_intent_ids,
      })
      break

    case 'kyc.expired':
      // User will need to re-verify on the next ramp intent
      await updateUserKycStatus(event.user_id, 'expired')
      break
  }
}
```

Skip KYC on repeat transactions

Once you receive `kyc.approved` for a user, store that status in your system. For subsequent ramp intents, pass `kycVerified: true` so the customer doesn't have to verify again.

GUIDES / WEBHOOKS

Other Webhooks

Additional webhook events for refunds, settlements, chargebacks, and deposits.

Refund Events

Event	Description
`refund.initiated`	Refund initiated
`refund.completed`	Refund completed
`refund.failed`	Refund failed

Settlement Events

Event	Description
`settlement.initiated`	Settlement started
`settlement.completed`	Settlement completed
`settlement.failed`	Settlement failed

Chargeback Events

Event	Description
`chargeback.initiated`	Chargeback initiated
`chargeback.resolved`	Chargeback resolved

Deposit Events

Event	Description
`deposit.pending`	Crypto deposit pending
`deposit.confirmed`	Crypto deposit confirmed on-chain
`deposit.cleared`	Crypto deposit fully cleared

GUIDES / WEBHOOKS

Signature Verification

ZenPay signs every webhook delivery so you can verify it came from ZenPay and was not tampered with.

How Signatures Work

1. **Algorithm:** HMAC-SHA256 by default (configurable per endpoint to `sha256` or `sha512` in the dashboard).
2. **Signed content:** The raw JSON request body as a UTF-8 string.
3. **Signature format:** `{algorithm}={hex_digest}` — e.g. `sha256=a1b2c3d4e5f6...`
4. **Header:** `X-ZenPay-Signature` (default, configurable per endpoint).
5. **Secret:** Your webhook endpoint secret from the dashboard.

> **Use the raw body**
> Always verify against the **raw request body** (bytes/string), not a parsed-then-re-serialized JSON object. Re-serialization can change key ordering or whitespace, which invalidates the signature.

Webhook Headers

Every webhook delivery includes the following headers:

Header	Description	Example
`X-ZenPay-Signature`	`{algorithm}={hex_hmac_digest}` of the raw JSON body using your webhook secret	`sha256=a1b2c3d4...`
`X-Zenpay-Event`	Event type string	`payment.success`
`X-Zenpay-Event-Id`	Unique event ID for deduplication	`evt_abc123`
`X-Zenpay-Timestamp`	ISO 8601 delivery timestamp	`2026-02-28T10:30:45.123Z`
`X-Zenpay-Attempt`	Delivery attempt number (starts at 1)	`1`
`User-Agent`	Delivery user agent	`Zenpay-Webhook/1.0`

Verification Implementation

JAVASCRIPT / NODE.JS

Use the helpers shipped in the SDK — you do not need to implement HMAC yourself. Both are exported at the package root and take no API key, so they run inside a webhook route with no client configured:

```
import { constructWebhookEvent, verifyWebhookSignature } from '@zenxdigitalholdings/zenpays'

// Returns a boolean.
const ok = await verifyWebhookSignature(rawBody, signatureHeader, secret)

// Or: verify and parse in one step. Throws Error('Invalid webhook signature')
// on failure, so a successful return guarantees authenticity.
const event = await constructWebhookEvent<PaymentSuccessEvent>(rawBody, signatureHeader, secret)
```

Parameter	Type	Description
`rawBody`	`string`	The raw request body, exactly as received
`signatureHeader`	`string | undefined | null`	`X-ZenPay-Signature` value. Accepts `sha256=…` or bare hex
`secret`	`string`	Your webhook signing secret

Both are `async` (they use Web Crypto, with a `node:crypto` fallback below Node 18) and both return `false` / throw rather than error out on a missing body, header, or secret.

> **SHA-256 only**
> The bundled helpers implement HMAC-SHA256. If you have configured an endpoint to sign with `sha512`, verify it with the manual implementation below instead.

Full Express example:

```
import { constructWebhookEvent } from '@zenxdigitalholdings/zenpays'
import express from 'express'

const app = express()

app.post(
  '/webhooks/zenpays',
  // express.raw() keeps the body unparsed — express.json() would re-serialize
  // it and break the signature.
  express.raw({ type: 'application/json' }),
  async (req, res) => {
    const signature = req.headers['x-zenpay-signature'] as string
    const timestamp = req.headers['x-zenpay-timestamp'] as string
    const eventId = req.headers['x-zenpay-event-id'] as string

    // 1. Verify the signature and parse in one step.
    let event: unknown
    try {
      event = await constructWebhookEvent(
        req.body.toString(),
        signature,
        process.env.WEBHOOK_SECRET!,
      )
    }
    catch {
      return res.status(401).json({ error: 'Invalid signature' })
    }

    // 2. Check timestamp freshness (prevent replay attacks)
    const deliveryTime = new Date(timestamp)
    const now = new Date()
    if (Math.abs(now.getTime() - deliveryTime.getTime()) > 5 * 60 * 1000) {
      return res.status(401).json({ error: 'Webhook timestamp expired' })
    }

    // 3. Deduplicate using event ID
    // if (await isEventAlreadyProcessed(eventId)) {
    //   return res.status(200).send('OK')
    // }

    // 4. Process
    handleWebhookEvent(event)

    res.status(200).send('OK')
  }
)
```

Manual implementation (no SDK, or
sha512
endpoints)

```typescript
import crypto from 'node:crypto'

function verifyWebhookSignature(
  rawBody: Buffer | string,
  signature: string,
  secret: string
): boolean {
  const [algorithm, receivedHash] = signature.split('=')

  const expectedHash = crypto
    .createHmac(algorithm, secret)
    .update(rawBody)
    .digest('hex')

  // Timing-safe comparison to prevent timing attacks
  const received = Buffer.from(receivedHash, 'hex')
  const expected = Buffer.from(expectedHash, 'hex')

  if (received.length !== expected.length) return false
  return crypto.timingSafeEqual(received, expected)
}
```

PYTHON

```python
import hmac
import hashlib
from datetime import datetime, timezone, timedelta
from flask import Flask, request, jsonify

app = Flask(__name__)

def verify_webhook_signature(raw_body: bytes, signature: str, secret: str) -> bool:
    algorithm, received_hash = signature.split("=", 1)

    expected_hash = hmac.new(
        secret.encode("utf-8"),
        raw_body,
        getattr(hashlib, algorithm),
    ).hexdigest()

    return hmac.compare_digest(received_hash, expected_hash)


@app.route("/webhooks/zenpays", methods=["POST"])
def handle_webhook():
    signature = request.headers.get("X-ZenPay-Signature", "")
    timestamp = request.headers.get("X-Zenpay-Timestamp", "")
    event_id = request.headers.get("X-Zenpay-Event-Id", "")

    # 1. Verify signature
    if not verify_webhook_signature(request.data, signature, WEBHOOK_SECRET):
        return jsonify({"error": "Invalid signature"}), 401

    # 2. Check timestamp freshness
    delivery_time = datetime.fromisoformat(timestamp.replace("Z", "+00:00"))
    now = datetime.now(timezone.utc)
    if abs((now - delivery_time).total_seconds()) > 300:
        return jsonify({"error": "Webhook timestamp expired"}), 401

    # 3. Parse and process
    event = request.get_json(force=True)
    handle_webhook_event(event)

    return "OK", 200
```

Replay Attack Prevention

Validate `X-Zenpay-Timestamp` is within 5 minutes of the current server time. This prevents attackers from replaying captured webhook deliveries.

Retry Behavior

If your endpoint fails to respond with a `2xx` status code within 30 seconds, ZenPay will retry the delivery:

- **Max retries**: 3 attempts
- **Backoff**: Exponential backoff between retries
- Each retry includes an incremented `X-Zenpay-Attempt` header

Best Practices

1. **Respond quickly** — Return a `2xx` response within 30 seconds. Queue events for asynchronous background processing.
2. **Handle duplicates** — Events may be delivered more than once. Use `X-Zenpay-Event-Id` to deduplicate.
3. **Use HTTPS** — Always use secure endpoints in production.
4. **Verify signatures** — Never trust unverified payloads. Always validate `X-ZenPay-Signature` using the raw request body.
5. **Check timestamps** — Validate `X-Zenpay-Timestamp` is within 5 minutes of current time to prevent replay attacks.
6. **Log everything** — Keep records of received webhooks for debugging.
7. **Monitor failures** — Check the dashboard for failed deliveries and fix endpoint issues promptly.

GUIDES

Testing

Learn how to test your ZenPays integration.

Test Environment

Use test API keys for development:

JAVASCRIPT

```
const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_TEST_API_KEY!, // zp_test_xxx
  baseUrl: 'https://api.zenpays.com',
})
```

PYTHON

```
import os

zenpays = ZenPays(
    api_key=os.environ["ZENPAYS_TEST_API_KEY"],  # zp_test_xxx
    base_url="https://api.zenpays.com",
)
```

Test Payment Methods

Use these test values in the sandbox:

UPI

VPA	Scenario
`success@upi`	Successful payment
`failure@upi`	Payment failed
`timeout@upi`	Payment timeout

Net Banking

Bank Code	Scenario
`TEST_BANK_001`	Successful payment
`TEST_BANK_002`	Payment declined

Mocking the SDK

JAVASCRIPT

Using Vitest

```
import { describe, expect, it, vi } from 'vitest'
import { ZenPays } from '@zenxdigitalholdings/zenpays'

vi.mock('@zenxdigitalholdings/zenpays', () => ({
  ZenPays: vi.fn().mockImplementation(() => ({
    payments: {
      createPaymentIntent: vi.fn().mockResolvedValue({
        intentId: 'pi_test_xxx',
        status: 'pending',
      }),
    },
  })),
}))

describe('payment flow', () => {
  it('should create a payment intent', async () => {
    const zenpays = new ZenPays({ apiKey: 'test' })

    const intent = await zenpays.payments.createPaymentIntent({
      amount: 1000,
      currency: 'USD',
    })

    expect(intent.intentId).toBe('pi_test_xxx')
  })
})
```

Using Jest

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

jest.mock('@zenxdigitalholdings/zenpays')

const mockZenPays = ZenPays as jest.MockedClass<typeof ZenPays>

describe('payment flow', () => {
  beforeEach(() => {
    mockZenPays.mockImplementation(() => ({
      payments: {
        createPaymentIntent: jest.fn().mockResolvedValue({
          intentId: 'pi_test_xxx',
          status: 'pending',
        }),
      },
    } as any))
  })

  it('should create a payment intent', async () => {
    const zenpays = new ZenPays({ apiKey: 'test' })
    const intent = await zenpays.payments.createPaymentIntent({
      amount: 1000,
      currency: 'USD',
    })

    expect(intent.intentId).toBe('pi_test_xxx')
  })
})
```

PYTHON

Using unittest.mock

```
from unittest.mock import Mock, patch
import pytest
from zenpays import ZenPays


def test_create_payment_intent():
    with patch('zenpays.ZenPays') as MockZenPays:
        # Setup mock
        mock_instance = MockZenPays.return_value
        mock_instance.payments.create_payment_intent.return_value = {
            "intent_id": "pi_test_xxx",
            "status": "pending",
        }

        # Test code
        zenpays = ZenPays(api_key="test")
        intent = zenpays.payments.create_payment_intent({
            "amount": 1000,
            "currency": "USD",
        })

        assert intent["intent_id"] == "pi_test_xxx"
```

Using pytest-mock

```python
import pytest
from zenpays import ZenPays


def test_create_payment_intent(mocker):
    # Mock the ZenPays class
    mock_zenpays = mocker.Mock(spec=ZenPays)
    mock_zenpays.payments.create_payment_intent.return_value = {
        "intent_id": "pi_test_xxx",
        "status": "pending",
    }

    # Test code
    intent = mock_zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })

    assert intent["intent_id"] == "pi_test_xxx"
```

Integration Tests

Test against the sandbox API:

JAVASCRIPT

```javascript
import { ZenPays } from '@zenxdigitalholdings/zenpays'

describe('integration tests', () => {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_TEST_API_KEY!,
    baseUrl: 'https://api.zenpays.com',
  })

  it('should create and confirm a payment', async () => {
    // Create intent
    const intent = await zenpays.payments.createPaymentIntent({
      amount: 1000,
      currency: 'USD',
    })

    expect(intent.intentId).toBeDefined()

    // Confirm payment
    const result = await zenpays.payments.confirmPayment(intent.intentId, {
      customerDetails: {
        name: 'Test User',
        email: 'test@example.com',
        address: { country: 'US' },
      },
      paymentMethodDetails: {
        type: 'upi',
        vpa: 'test@upi',
      },
    })

    expect(result.status).toBe('succeeded')
  })
})
```

PYTHON

```python
import os
import pytest
from zenpays import ZenPays


@pytest.fixture
def zenpays():
    return ZenPays(
        api_key=os.environ["ZENPAYS_TEST_API_KEY"],
        base_url="https://api.zenpays.com",
    )


def test_create_and_confirm_payment(zenpays):
    # Create intent
    intent = zenpays.payments.create_payment_intent({
        "amount": 1000,
        "currency": "USD",
    })

    assert "intent_id" in intent

    # Confirm payment
    result = zenpays.payments.confirm_payment(intent["intent_id"], {
        "customer_details": {
            "name": "Test User",
            "email": "test@example.com",
            "address": {"country": "US"},
        },
        "payment_method_details": {
            "type": "upi",
            "vpa": "test@upi",
        },
    })

    assert result["status"] == "succeeded"
```

Custom Fetch for Testing

JAVASCRIPT

Provide a custom fetch implementation:

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const mockFetch = vi.fn().mockResolvedValue({
  ok: true,
  json: () => Promise.resolve({
    success: true,
    data: { intentId: 'pi_xxx', status: 'pending' },
  }),
})

const zenpays = new ZenPays({
  apiKey: 'test',
  fetch: mockFetch,
})

// Now all requests go through mockFetch
await zenpays.payments.createPaymentIntent({ amount: 1000, currency: 'USD' })

expect(mockFetch).toHaveBeenCalledWith(
  expect.stringContaining('/payment-intent'),
  expect.objectContaining({ method: 'POST' })
)
```

PYTHON

Provide a custom session for testing:

```
from unittest.mock import Mock
from zenpays import ZenPays
import requests


def test_with_custom_session():
    # Create a mock session
    mock_session = Mock(spec=requests.Session)
    mock_response = Mock()
    mock_response.json.return_value = {
        "success": True,
        "data": {"intent_id": "pi_xxx", "status": "pending"},
    }
    mock_response.raise_for_status.return_value = None
    mock_session.post.return_value = mock_response

    # Create ZenPays with custom session
    zenpays = ZenPays(api_key="test", session=mock_session)

    # Now all requests go through mock_session
    intent = zenpays.payments.create_payment_intent({"amount": 1000, "currency": "USD"})

    # Verify the call
    mock_session.post.assert_called_once()
    assert "payment-intent" in mock_session.post.call_args[0][0]
```

Testing Webhooks

Use a tool like ngrok for local webhook testing:

```
ngrok http 3000
```

Then register the ngrok URL as your webhook endpoint:

JAVASCRIPT

```
await zenpays.merchants.createWebhook({
  url: 'https://your-ngrok-url.ngrok.io/webhooks',
  events: ['payment.intent.succeeded'],
})
```

PYTHON

```
zenpays.merchants.create_webhook({
    "url": "https://your-ngrok-url.ngrok.io/webhooks",
    "events": ["payment.intent.succeeded"],
})
```

GUIDES

TypeScript

The ZenPays SDK is written in TypeScript and provides full type safety.

Type Imports

Import types alongside runtime exports:

```
import type { Customer, PaymentIntent, Transaction } from '@zenxdigitalholdings/zenpays'
import {

  ZenPays
} from '@zenxdigitalholdings/zenpays'
```

Or import types from the types subpath:

```
import type {
  CreatePaymentIntentRequest,
  Customer,
  PaymentIntent,
} from '@zenxdigitalholdings/zenpays/types'
```

Type-Safe API Calls

All API methods are fully typed:

```
const zenpays = new ZenPays({ apiKey: 'xxx' })

// Request type is inferred
const intent = await zenpays.payments.createPaymentIntent({
  amount: 1000,
  currency: 'USD',
  // TypeScript will error if you miss required fields
})

// Response type is PaymentIntentResponse
console.log(intent.intentId) // string
console.log(intent.status) // PaymentStatus
```

Generic Response Types

Paginated responses use generics:

```
import type { PaginatedResponse, Transaction } from '@zenxdigitalholdings/zenpays'

const result: PaginatedResponse<Transaction>
  = await zenpays.payments.listTransactions()

result.data // Transaction[]
result.total // number
result.page // number
```

Discriminated Unions

Status types use discriminated unions for type narrowing:

```
const result = await zenpays.payments.confirmPayment('pi_xxx', { ... })

if (result.nextAction?.type === 'redirect') {
  // TypeScript knows redirectUrl exists
  window.location.href = result.nextAction.redirectUrl!
} else if (result.nextAction?.type === 'three_ds') {
  // TypeScript knows threeDsUrl exists
  handle3DSecure(result.nextAction.threeDsUrl!)
}
```

Error Types

Error classes are also typed:

```
import {
  ZenPaysError,
  ValidationError,
  RateLimitError,
} from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.createPaymentIntent({ ... })
} catch (error) {
  if (error instanceof ValidationError) {
    // TypeScript knows about .fields
    console.log(error.fields)
  } else if (error instanceof RateLimitError) {
    // TypeScript knows about .retryAfter
    console.log(error.retryAfter)
  }
}
```

Utility Types

The SDK exports utility types:

```
import type {
  Currency,
  PaymentMethodType,
  PaymentStatus,
  RiskLevel,
} from '@zenxdigitalholdings/zenpays'

// Use in your own types
interface Order {
  id: string
  paymentStatus: PaymentStatus
  paymentMethod: PaymentMethodType
  currency: Currency
}
```

Configuration Types

```
import type { ZenPaysConfig } from '@zenxdigitalholdings/zenpays'

// Type-safe configuration
const config: ZenPaysConfig = {
  apiKey: process.env.ZENPAYS_API_KEY!,
  timeout: 30000,
}
```

Strict Mode

The SDK works best with strict TypeScript settings:

```
{
  "compilerOptions": {
    "strict": true,
    "strictNullChecks": true
  }
}
```

PART

SDK Reference

21 documents

SDK REFERENCE

Client

The `ZenPays` client is the main entry point for the SDK.

Constructor

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays(config: ZenPaysConfig)
```

PYTHON

```
from zenpays import ZenPays

zenpays = ZenPays(**config)
```

Configuration

JavaScript

Property	Type	Required	Default	Description
`apiKey`	`string`	Yes	-	Your ZenPays API key
`baseUrl`	`string`	No	`'https://api.zenpays.com'`	API base URL
`apiVersion`	`string`	No	`'v1'`	API version
`timeout`	`number`	No	`30000`	Request timeout in ms
`fetch`	`typeof fetch`	No	`globalThis.fetch`	Custom fetch function

Python

Property	Type	Required	Default	Description
`api_key`	`str`	Yes	-	Your ZenPays API key
`base_url`	`str`	No	Auto-detected	API base URL
`timeout`	`int`	No	`30`	Request timeout in seconds
`secret_salt`	`str`	No	`None`	HMAC secret for request signing

Factory Function

JAVASCRIPT

```javascript
import { createClient } from '@zenxdigitalholdings/zenpays'

const zenpays = createClient({
  apiKey: 'your-api-key',
})
```

PYTHON

```python
from zenpays import ZenPays

# Direct initialization (recommended)
zenpays = ZenPays(api_key="your-api-key")
```

Properties

version

Returns the SDK version.

JAVASCRIPT

```javascript
console.log(zenpays.version) // '0.5.0'
```

PYTHON

```python
from zenpays import __version__

print(__version__)  # '0.1.1'
```

API Modules

The client provides access to all API modules:

Property	Type	Description
`analytics`	`AnalyticsApi`	Reporting and aggregate metrics
`auth`	`AuthApi`	API key and session operations
`chargebacks`	`ChargebacksApi`	Disputes and evidence submission
`checkout`	`CheckoutApi`	Checkout sessions and on-ramp
`customers`	`CustomersApi`	Customer management
`invoices`	`InvoicesApi`	Invoice creation and delivery
`ledger`	`LedgerApi`	Double-entry ledger and reconciliation
`merchants`	`MerchantsApi`	Webhooks, bank accounts, IP whitelist
`offRamp`	`OffRampApi`	Crypto-to-fiat conversion
`paymentLinks`	`PaymentLinksApi`	Shareable payment links

Property	Type	Description
`payments`	`PaymentsApi`	Payment intents and transactions
`payoutIntents`	*(object)*	Payout-intent shorthands — see below
`payouts`	`PayoutsApi`	Payout operations
`rampIntents`	`RampIntentsApi`	On/off-ramp intent lifecycle
`refunds`	`RefundsApi`	Refund operations
`security`	`SecurityApi`	2FA and security settings
`settlements`	`SettlementsApi`	Settlement management
`subscriptions`	`SubscriptionsApi`	Plans, stored methods, recurring billing
`tenantHostnames`	`TenantHostnamesApi`	White-label hostname mapping
`vendors`	`VendorsApi`	Vendor accounts, commissions, referrals
`wallet`	`WalletApi`	Wallet balance and transactions

> **Python parity**
>
> The Python SDK currently implements `analytics`, `checkout`, `customers`, `merchants`, `payments`, `payouts`, `refunds`, `security`, `settlements`, and `wallet`. The remaining namespaces are JavaScript-only — call the REST endpoints directly from Python.

payoutIntents

`payoutIntents` is a convenience object, not an API class. Each property forwards to the correspondingly named method on `payouts`:

Property	Forwards to
`payoutIntents.create`	`payouts.createPayoutIntent`
`payoutIntents.confirm`	`payouts.confirmPayoutIntent`
`payoutIntents.get`	`payouts.getPayoutIntent`
`payoutIntents.list`	`payouts.listPayoutIntents`
`payoutIntents.cancel`	`payouts.cancelPayoutIntent`

Webhook Helpers

Signature verification is exported at the package root rather than hung off the client, so it can run in a webhook route with no API key present:

```
import { constructWebhookEvent, verifyWebhookSignature } from '@zenxdigitalholdings/zenpays'
```

See the signature verification guide.

Example

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({
  apiKey: process.env.ZENPAYS_API_KEY!,
})

// Use any API module
const balance = await zenpays.wallet.getBalance()
const customers = await zenpays.customers.list()
```

PYTHON

```
import os
from zenpays import ZenPays

zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

# Use any API module
balance = zenpays.wallet.get_balance()
customers = zenpays.customers.list()
```

Context Manager

JAVASCRIPT

JavaScript doesn't require explicit resource cleanup.

PYTHON

The Python SDK supports context managers for automatic cleanup:

```
with ZenPays(api_key="your-api-key") as zenpays:
    balance = zenpays.wallet.get_balance()
    # HTTP session automatically closed after the block
```

Error Handling

All API methods throw typed errors:

JAVASCRIPT

```
import {
  AuthenticationError,
  AuthorizationError,
  ConfigurationError,
  NetworkError,
  NotFoundError,
  PaymentError,
  RateLimitError,
  ValidationError,
  ZenPaysError,
} from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.getPaymentIntent('invalid-id')
}
catch (error) {
  if (error instanceof NotFoundError) {
    console.log('Payment intent not found')
  }
  else if (error instanceof AuthenticationError) {
    console.log('Invalid API key')
  }
}
```

PYTHON

```
from zenpays.errors import (
    AuthenticationError,
    PermissionError,
    NetworkError,
    ResourceNotFoundError,
    PaymentError,
    RateLimitError,
    ValidationError,
    ZenPaysError,
)

try:
    zenpays.payments.get_payment_intent("invalid-id")
except ResourceNotFoundError:
    print("Payment intent not found")
except AuthenticationError:
    print("Invalid API key")
```

See the Error Handling guide for more details.

SDK REFERENCE

Payments

Create and manage payment intents, confirm payments, and track transactions. Access via `zenpays.payments`.

Payment Intents

createPaymentIntent

JAVASCRIPT

```
const intent = await zenpays.payments.createPaymentIntent({
  amount: 500,
  currency: 'INR',
  customerEmail: 'customer@example.com',
  customerPhone: '+911234567890',
  customerCountry: 'IN',
  description: 'Order #12345',
  returnUrl: 'https://yoursite.com/complete',
})
```

PYTHON

```
intent = zenpays.payments.create_payment_intent({
    "amount": 500,
    "currency": "INR",
    "customerEmail": "customer@example.com",
    "description": "Order #12345",
})
```

---

getPaymentIntent

```
const intent = await zenpays.payments.getPaymentIntent('pi_xxx')
```

---

getPublicPaymentIntent

```
const intent = await zenpays.payments.getPublicPaymentIntent('pi_xxx', 'optional_token')
```

---

confirmPayment

JAVASCRIPT

```
const result = await zenpays.payments.confirmPayment('pi_xxx', {
  customerDetails: {
    name: 'John Doe',
    email: 'john@example.com',
    phone: '+911234567890',
    address: { country: 'IN' },
  },
  paymentMethodDetails: {
    paymentMethod: 'FIAT',
    paymentType: 'UPI',
  },
  confirmSource: 'MERCHANT_SITE',
})
```

PYTHON

```
result = zenpays.payments.confirm_payment("pi_xxx", {
    "customerDetails": {
        "name": "John Doe",
        "email": "john@example.com",
        "address": {"country": "IN"},
    },
    "paymentMethodDetails": {
        "paymentMethod": "FIAT",
        "paymentType": "UPI",
    },
    "confirmSource": "MERCHANT_SITE",
})
```

processPayment

```
const result = await zenpays.payments.processPayment('pi_xxx', confirmRequest)
```

cancelPaymentIntent

```
const intent = await zenpays.payments.cancelPaymentIntent('pi_xxx')
```

updateIntelligence

```
await zenpays.payments.updateIntelligence('pi_xxx', {
  userAgent: navigator.userAgent,
  screenWidth: window.screen.width,
  screenHeight: window.screen.height,
})
```

Transactions

getTransaction

```
const txn = await zenpays.payments.getTransaction('txn_xxx')
```

getTransactionsByIntent

```
const transactions = await zenpays.payments.getTransactionsByIntent('pi_xxx')
```

listTransactions

JAVASCRIPT

```
const { data, total } = await zenpays.payments.listTransactions({
  status: 'success',
  from: '2024-01-01',
  to: '2024-12-31',
  limit: 50,
})
```

PYTHON

```
result = zenpays.payments.list_transactions({
    "status": "success",
    "from": "2024-01-01",
    "to": "2024-12-31",
    "limit": 50,
})
```

getTransactionAnalytics

```
const analytics = await zenpays.payments.getTransactionAnalytics('30d')
```

exportTransactions

```
const downloadUrl = await zenpays.payments.exportTransactions(
  { status: 'success', from: '2024-01-01' },
  'csv' // or 'xlsx'
)
```

UPI Deep Link (INR Only)

getUpiDeepLink

Extract the `upi://` deep link from a TSP payment page QR code. Used for INR payments to enable direct UPI app opening on mobile.

```
const result = await zenpays.payments.getUpiDeepLink(
  'pi_xxx',
  'https://cashier.ppco.lol?orderNo=C123', // payPageUrl from confirm response
  'C123'                                    // cashierOrderNo (optional)
)

if (result.success && result.data.upiDeepLink) {
  window.location.href = result.data.upiDeepLink // Opens UPI app on mobile
}
```

SDK REFERENCE

Customers

Manage customers, track risk profiles, and view activity. Access via `zenpays.customers`.

Methods

create

JAVASCRIPT

```
const customer = await zenpays.customers.create({
  email: 'john@example.com',
  name: 'John Doe',
  phone: '+911234567890',
  country: 'IN',
})
```

PYTHON

```
customer = zenpays.customers.create({
    "email": "john@example.com",
    "name": "John Doe",
    "phone": "+911234567890",
    "country": "IN",
})
```

---

get

```
const customer = await zenpays.customers.get('cust_xxx')
```

---

update

```
const updated = await zenpays.customers.update('cust_xxx', { name: 'Jane Doe' })
```

---

list

JAVASCRIPT

```
const { data } = await zenpays.customers.list({ page: 1, limit: 20 })
```

PYTHON

```
result = zenpays.customers.list({"page": 1, "limit": 20})
```

---

search

```
const results = await zenpays.customers.search('john@example.com')
```

getRiskProfile

```
const profile = await zenpays.customers.getRiskProfile('cust_xxx')
```

getActivityHistory

```
const history = await zenpays.customers.getActivityHistory('cust_xxx', {
  type: 'payment',
  from: '2024-01-01',
  to: '2024-12-31',
  limit: 50,
})
```

getTopCustomers

```
const top = await zenpays.customers.getTopCustomers({
  startDate: '2024-01-01',
  endDate: '2024-12-31',
  limit: 10,
})
```

getRecentPayments

```
const payments = await zenpays.customers.getRecentPayments({ limit: 10 })
```

getAnalytics

```
const analytics = await zenpays.customers.getAnalytics({
  startDate: '2024-01-01',
  endDate: '2024-12-31',
})
// Returns: { totalCustomers, newCustomers, activeCustomers, churnRate }
```

block

```
const blocked = await zenpays.customers.block('cust_xxx', 'Suspected fraud')
```

unblock

```
const unblocked = await zenpays.customers.unblock('cust_xxx')
```

SDK REFERENCE

Refunds

Process refunds for INR transactions -- full, partial, and bulk. Access via `zenpays.refunds`.

Methods

getTemplate

Ask which beneficiary fields this transaction's payment provider actually requires, before you build a refund form. Different providers need different fields, so hard-coding them breaks the moment a transaction is routed elsewhere.

```
const template = await zenpays.refunds.getTemplate('txn_xxx')

if (template.requiredFields.length > 0) {
  // Render a form for exactly the fields this provider needs
}
else {
  // Nothing extra needed — refund directly
  await zenpays.refunds.create({ transactionId: 'txn_xxx' })
}

// Partial refund: pass the amount to preview fees and limits for that amount
const partial = await zenpays.refunds.getTemplate('txn_xxx', 500)
```

Parameter	Type	Required	Description
`transactionId`	`string`	Yes	Transaction to be refunded
`amount`	`number`	No	Refund amount. Omit for a full refund

Returns a `RefundPreview`:

Field	Type	Description
`tspProvider`	`string`	Provider slug, e.g. `sulifu_pay`, `stripe`
`tspDisplayName`	`string`	Human-readable provider name
`requiredFields`	`RefundFieldDefinition[]`	Fields that must be supplied to `create`
`optionalFields`	`RefundFieldDefinition[]`	Fields that may be supplied
`oneOfGroups`	`string[][]`	Groups where at least one complete group is required
`prefilled`	`object`	Values carried over from the original transaction

preview

Alias for `getTemplate`, taking the amount as an options object instead of a positional argument.

```
const preview = await zenpays.refunds.preview('txn_xxx', { amount: 500 })
```

create

Create a refund. Provide the customer's bank account details so they receive the funds via IMPS.

JAVASCRIPT

```
// Full refund
const refund = await zenpays.refunds.create({
  transactionId: 'txn_xxx',
  reason: 'Customer request',
  beneficiaryEmail: 'customer@example.com',
  beneficiaryAccount: '50100012345678',
  beneficiaryIfsc: 'HDFC0001234',
})

// Partial refund
const partialRefund = await zenpays.refunds.create({
  transactionId: 'txn_xxx',
  amount: 500,
  reason: 'Partial item return',
  beneficiaryEmail: 'customer@example.com',
  beneficiaryAccount: '91020034567890',
  beneficiaryIfsc: 'SBIN0001234',
  beneficiaryName: 'Priya Sharma',
})
```

PYTHON

```
# Full refund
refund = zenpays.refunds.create({
    "transactionId": "txn_xxx",
    "reason": "Customer request",
    "beneficiaryEmail": "customer@example.com",
    "beneficiaryAccount": "50100012345678",
    "beneficiaryIfsc": "HDFC0001234",
})

# Partial refund
partial_refund = zenpays.refunds.create({
    "transactionId": "txn_xxx",
    "amount": 500,
    "reason": "Partial item return",
    "beneficiaryEmail": "customer@example.com",
    "beneficiaryAccount": "91020034567890",
    "beneficiaryIfsc": "SBIN0001234",
    "beneficiaryName": "Priya Sharma",
})
```

Parameters:

Field	Type	Required	Description
`transactionId`	string	Yes	Transaction to refund
`reason`	string	Yes	Refund reason
`amount`	number	No	Partial refund amount (omit for full refund)

Field	Type	Required	Description
`beneficiaryEmail`	string	Yes	Customer email
`beneficiaryAccount`	string	Yes	Bank account number
`beneficiaryIfsc`	string	Yes	IFSC code (e.g. `HDFC0001234`)
`beneficiaryName`	string	No	Beneficiary name
`beneficiaryMobile`	string	No	Mobile number
`beneficiaryCity`	string	No	City
`customerId`	string	No	Customer ID (auto-resolved if omitted)

Try it out

Test refund creation interactively using the API Simulator.

---

bulkUpload

Create multiple refunds in a single call.

JAVASCRIPT

```
const result = await zenpays.refunds.bulkUpload({
  refunds: [
    {
      transactionId: 'txn_aaa',
      reason: 'Customer request',
      beneficiaryEmail: 'john@example.com',
      beneficiaryAccount: '50100012345678',
      beneficiaryIfsc: 'HDFC0001234',
    },
    {
      transactionId: 'txn_bbb',
      amount: 500,
      reason: 'Partial refund',
      beneficiaryEmail: 'jane@example.com',
      beneficiaryAccount: '91020034567890',
      beneficiaryIfsc: 'SBIN0001234',
    },
  ],
})
```

PYTHON

```
result = zenpays.refunds.bulk_upload({
    "refunds": [
        {
            "transactionId": "txn_aaa",
            "reason": "Customer request",
            "beneficiaryEmail": "john@example.com",
            "beneficiaryAccount": "50100012345678",
            "beneficiaryIfsc": "HDFC0001234",
        },
        {
            "transactionId": "txn_bbb",
            "amount": 500,
            "reason": "Partial refund",
            "beneficiaryEmail": "jane@example.com",
            "beneficiaryAccount": "91020034567890",
            "beneficiaryIfsc": "SBIN0001234",
        },
    ],
})
```

get

JAVASCRIPT

```
const refund = await zenpays.refunds.get('refund_xxx')
```

PYTHON

```
refund = zenpays.refunds.get("refund_xxx")
```

list

JAVASCRIPT

```
const { data } = await zenpays.refunds.list({
  status: 'completed',
  currency: 'INR',
  limit: 20,
})
```

PYTHON

```
result = zenpays.refunds.list({"status": "completed", "currency": "INR", "limit": 20})
```

cancel

Cancel a refund that is still in `pending_approval` status.

JAVASCRIPT

```
const cancelled = await zenpays.refunds.cancel('refund_xxx')
```

PYTHON

```
cancelled = zenpays.refunds.cancel("refund_xxx")
```

---

getStats

JAVASCRIPT

```
const stats = await zenpays.refunds.getStats('2026-01-01', '2026-12-31', 'INR')
```

PYTHON

```
stats = zenpays.refunds.get_stats("2026-01-01", "2026-12-31", "INR")
```

---

getAnalytics

JAVASCRIPT

```
const analytics = await zenpays.refunds.getAnalytics({
  startDate: '2026-01-01',
  endDate: '2026-12-31',
  granularity: 'day',
  currency: 'INR',
})
```

PYTHON

```
analytics = zenpays.refunds.get_analytics({
    "startDate": "2026-01-01",
    "endDate": "2026-12-31",
    "granularity": "day",
    "currency": "INR",
})
```

---

export

JAVASCRIPT

```
const downloadUrl = await zenpays.refunds.export(
  { status: 'completed', currency: 'INR' },
  'csv'
)
```

PYTHON

```python
download_url = zenpays.refunds.export({"status": "completed", "currency": "INR"}, "csv")
```

SDK REFERENCE

Payouts

Process payouts — single, batch, and intent-based flows. Access via `zenpays.payouts`.

Single Payouts

getMethods

```
const methods = await zenpays.payouts.getMethods()
```

---

preview

JAVASCRIPT

```
const preview = await zenpays.payouts.preview({
  amount: 5000,
  currency: 'INR',
  country: 'IN',
  payoutType: 'bank_transfer',
})
// Returns: { tspProvider, requiredFields, estimatedProcessingTime, fees, totalAmount }
```

PYTHON

```
preview = zenpays.payouts.preview({
    "amount": 5000,
    "currency": "INR",
    "country": "IN",
    "payoutType": "bank_transfer",
})
```

---

create

JAVASCRIPT

```
const payout = await zenpays.payouts.create({
  amount: 5000,
  currency: 'INR',
  beneficiary: {
    name: 'John Doe',
    accountNumber: '1234567890',
    ifscCode: 'HDFC0001234',
  },
})
```

PYTHON

```
payout = zenpays.payouts.create({
    "amount": 5000,
    "currency": "INR",
    "beneficiary": {
        "name": "John Doe",
        "accountNumber": "1234567890",
        "ifscCode": "HDFC0001234",
    },
})
```

get

```
const payout = await zenpays.payouts.get('po_xxx')
```

list

```
const { data } = await zenpays.payouts.list({ status: 'completed', limit: 20 })
```

listByCustomer

```
const { data } = await zenpays.payouts.listByCustomer('cust_xxx', { limit: 10 })
```

retry

```
const retried = await zenpays.payouts.retry('po_xxx', 'optional_tsp_provider')
```

cancel

```
const cancelled = await zenpays.payouts.cancel('po_xxx', 'No longer needed')
```

getStats

```
const stats = await zenpays.payouts.getStats('2024-01-01', '2024-12-31')
```

getAnalytics

```
const analytics = await zenpays.payouts.getAnalytics('30d')
```

getDashboardAnalytics

```
const dashboard = await zenpays.payouts.getDashboardAnalytics()
// Returns: PayoutAnalytics & { summary: PayoutStats }
```

export

```
const downloadUrl = await zenpays.payouts.export({ status: 'completed' })
```

Batch Payouts

createBatch

JAVASCRIPT

```
const batch = await zenpays.payouts.createBatch({
  name: 'April Payroll',
  payouts: [
    { amount: 5000, currency: 'INR', beneficiary: { name: 'John', accountNumber: '123' } },
    { amount: 3000, currency: 'INR', beneficiary: { name: 'Jane', accountNumber: '456' } },
  ],
})
```

PYTHON

```
batch = zenpays.payouts.create_batch({
    "name": "April Payroll",
    "payouts": [
        {"amount": 5000, "currency": "INR", "beneficiary": {"name": "John", "accountNumber":
"123"}},
        {"amount": 3000, "currency": "INR", "beneficiary": {"name": "Jane", "accountNumber":
"456"}},
    ],
})
```

getBatch

```
const batch = await zenpays.payouts.getBatch('batch_xxx')
```

listBatches

```
const { data } = await zenpays.payouts.listBatches({ status: 'pending', limit: 10 })
```

confirmBatch

```
const confirmed = await zenpays.payouts.confirmBatch('batch_xxx')
```

getBatchPayouts

```
const { data } = await zenpays.payouts.getBatchPayouts('batch_xxx', { limit: 50 })
```

cancelBatch

```
const result = await zenpays.payouts.cancelBatch('batch_xxx')
// Returns: { success, message }
```

Payout Intents

Intent-based payout flow: create intent → confirm with beneficiary fields → payout executes.

createPayoutIntent

JAVASCRIPT

```
const intent = await zenpays.payouts.createPayoutIntent({
  amount: 5000,
  currency: 'INR',
  country: 'IN',
  payoutType: 'bank_transfer',
  description: 'Vendor payment',
})
```

PYTHON

```
intent = zenpays.payouts.create_payout_intent({
    "amount": 5000,
    "currency": "INR",
    "country": "IN",
    "payoutType": "bank_transfer",
    "description": "Vendor payment",
})
```

confirmPayoutIntent

JAVASCRIPT

```
const confirmed = await zenpays.payouts.confirmPayoutIntent('poi_xxx', {
  beneficiaryName: 'John Doe',
  accountNumber: '1234567890',
  ifscCode: 'HDFC0001234',
})
```

PYTHON

```
confirmed = zenpays.payouts.confirm_payout_intent("poi_xxx", {
    "beneficiaryName": "John Doe",
    "accountNumber": "1234567890",
    "ifscCode": "HDFC0001234",
})
```

---

getPayoutIntent

```
const intent = await zenpays.payouts.getPayoutIntent('poi_xxx')
```

---

listPayoutIntents

```
const { data } = await zenpays.payouts.listPayoutIntents({ status: 'pending', limit: 20 })
```

---

cancelPayoutIntent

```
const cancelled = await zenpays.payouts.cancelPayoutIntent('poi_xxx', 'Changed plans')
```

SDK REFERENCE

Settlements

Manage merchant settlements and bank accounts. Access via `zenpays.settlements`.

Settlements

list

JAVASCRIPT

```
const { data } = await zenpays.settlements.list({
  status: 'completed',
  from: '2024-01-01',
})
```

PYTHON

```
result = zenpays.settlements.list({"status": "completed", "from": "2024-01-01"})
```

---

get

```
const settlement = await zenpays.settlements.get('stl_xxx')
```

---

create

JAVASCRIPT

```
const settlement = await zenpays.settlements.create({
  amount: 10000,
  currency: 'INR',
  bankAccountId: 'ba_xxx',
})
```

PYTHON

```
settlement = zenpays.settlements.create({
    "amount": 10000,
    "currency": "INR",
    "bankAccountId": "ba_xxx",
})
```

---

cancel

```
const cancelled = await zenpays.settlements.cancel('stl_xxx')
```

---

getStats

```
const stats = await zenpays.settlements.getStats({
  startDate: '2024-01-01',
  endDate: '2024-12-31',
})
// Returns: { totalSettled, pendingAmount, settlementCount, currency }
```

Bank Accounts

listBankAccounts

```
const { bank_accounts } = await zenpays.settlements.listBankAccounts()
```

---

addBankAccount

```
const { bank_account } = await zenpays.settlements.addBankAccount({
  accountNumber: '1234567890',
  ifscCode: 'HDFC0001234',
  accountHolderName: 'John Doe',
  bankName: 'HDFC Bank',
})
```

---

updateBankAccount

```
const { bank_account } = await zenpays.settlements.updateBankAccount('ba_xxx', {
  accountHolderName: 'Jane Doe',
})
```

---

deleteBankAccount

```
await zenpays.settlements.deleteBankAccount('ba_xxx')
```

SDK REFERENCE

Wallet

Manage wallet balances, transactions, ledger entries, commissions, and fees. Access via `zenpays.wallet`.

Balance

getBalance

JAVASCRIPT

```
const balance = await zenpays.wallet.getBalance('INR')
// Or get all: const balances = await zenpays.wallet.getBalance()
```

PYTHON

```
balance = zenpays.wallet.get_balance("INR")
```

getAllBalances

```
const balances = await zenpays.wallet.getAllBalances()
```

Transactions

listTransactions

JAVASCRIPT

```
const { data } = await zenpays.wallet.listTransactions({
  type: 'credit',
  from: '2024-01-01',
  limit: 50,
})
```

PYTHON

```
result = zenpays.wallet.list_transactions({"type": "credit", "limit": 50})
```

getTransaction

```
const txn = await zenpays.wallet.getTransaction('wtxn_xxx')
```

Adjustments

listAdjustments

```
const adjustments = await zenpays.wallet.listAdjustments()
```

createAdjustment

```
const adjustment = await zenpays.wallet.createAdjustment({
  amount: 100,
  currency: 'INR',
  type: 'credit',
  reason: 'Manual adjustment',
})
```

Ledger

getLedgerEntries

```
const entries = await zenpays.wallet.getLedgerEntries({
  account: 'merchant_balance',
  from: '2024-01-01',
  to: '2024-12-31',
  limit: 100,
})
```

getLedgerBalances

```
const balances = await zenpays.wallet.getLedgerBalances()
```

getReserveBalances

```
const reserves = await zenpays.wallet.getReserveBalances()
// Returns: [{ currency, reserved, available }]
```

getDailyLedgerSummary

```
const summary = await zenpays.wallet.getDailyLedgerSummary('2024-06-15')
// Returns: { date, totalDebits, totalCredits, netChange, byAccount }
```

getLedgerAnalytics

```
const analytics = await zenpays.wallet.getLedgerAnalytics('30d')
// Returns: { period, totalVolume, trend }
```

Commissions

getCommissions

```
const commissions = await zenpays.wallet.getCommissions({
  from: '2024-01-01',
  to: '2024-12-31',
  type: 'payment',
})
```

---

getCommissionAnalytics

```
const analytics = await zenpays.wallet.getCommissionAnalytics('30d')
// Returns: { period, totalCommission, byType, trend }
```

Fees

calculateFees

```
const fees = await zenpays.wallet.calculateFees({
  amount: 1000,
  currency: 'INR',
  type: 'payment',
})
```

---

getFeeConfig

```
const config = await zenpays.wallet.getFeeConfig()
// Returns: { paymentFees, payoutFees, refundFees, currency }
```

SDK REFERENCE

Ledger

Double-entry bookkeeping ledger — track every financial movement with full audit trail.

Methods

getOverview

JAVASCRIPT

```
const overview = await zenpays.ledger.getOverview({ currency: 'INR' })
```

PYTHON

```
overview = zenpays.ledger.get_overview({"currency": "INR"})
```

listEntries

JAVASCRIPT

```
const { data, total, hasMore } = await zenpays.ledger.listEntries({
  account: 'merchant_balance',
  from: '2024-01-01',
  to: '2024-12-31',
  page: 1,
  limit: 50,
})
```

PYTHON

```
result = zenpays.ledger.list_entries({
    "account": "merchant_balance",
    "from": "2024-01-01",
    "to": "2024-12-31",
    "page": 1,
    "limit": 50,
})
```

getBalances

```
const balances = await zenpays.ledger.getBalances({ currency: 'INR' })
// Returns: { available, pending, reserve, total } per account
```

getAccounts

```
const accounts = await zenpays.ledger.getAccounts({ type: 'asset', status: 'active' })
```

getReserves

```
const reserves = await zenpays.ledger.getReserves({ type: 'rolling', currency: 'INR' })
```

getDailySummary

```
const summaries = await zenpays.ledger.getDailySummary({
  startDate: '2024-01-01',
  endDate: '2024-01-31',
  currency: 'INR',
})
```

getSummaries

```
const summaries = await zenpays.ledger.getSummaries({ period: 'monthly', currency: 'INR' })
```

getReconciliation

```
const reconciliation = await zenpays.ledger.getReconciliation({
  startDate: '2024-01-01',
  endDate: '2024-01-31',
})
```

SDK REFERENCE

Merchants

Manage merchant profile, settings, API keys, webhooks, bank accounts, and IP whitelist. Access via `zenpays.merchants`.

Profile & Settings

getProfile

```
const profile = await zenpays.merchants.getProfile()
```

---

updateSettings

```
const settings = await zenpays.merchants.updateSettings({
  timezone: 'Asia/Kolkata',
  currency: 'INR',
  notifications: { emailNotifications: true },
})
```

---

getLimits

```
const limits = await zenpays.merchants.getLimits()
```

---

requestLimitIncrease

```
const result = await zenpays.merchants.requestLimitIncrease({
  type: 'daily_payin',
  requestedLimit: 50000000,
  reason: 'Scaling business volume',
})
// Returns: { requestId, status }
```

API Keys

listApiKeys

```
const keys = await zenpays.merchants.listApiKeys()
```

---

createApiKey

JAVASCRIPT

```
const { apiKey, apiSecret } = await zenpays.merchants.createApiKey({
  description: 'Production key',
  environment: 'live',
})
```

PYTHON

```
result = zenpays.merchants.create_api_key({
    "description": "Production key",
    "environment": "live",
})
```

---

revokeApiKey

```
await zenpays.merchants.revokeApiKey('key_xxx')
```

Webhooks

listWebhooks

```
const webhooks = await zenpays.merchants.listWebhooks()
```

---

createWebhook

JAVASCRIPT

```
const webhook = await zenpays.merchants.createWebhook({
  url: 'https://yoursite.com/webhooks/zenpays',
  events: ['payment.success', 'payment.failed', 'refund.created'],
})
```

PYTHON

```
webhook = zenpays.merchants.create_webhook({
    "url": "https://yoursite.com/webhooks/zenpays",
    "events": ["payment.success", "payment.failed", "refund.created"],
})
```

---

updateWebhook

```
const updated = await zenpays.merchants.updateWebhook('wh_xxx', {
  events: ['payment.success'],
})
```

---

deleteWebhook

```
await zenpays.merchants.deleteWebhook('wh_xxx')
```

---

testWebhook

```
const result = await zenpays.merchants.testWebhook('wh_xxx', 'payment.success')
// Returns: { success, statusCode?, error? }
```

---

getWebhookLogs

```
const logs = await zenpays.merchants.getWebhookLogs('wh_xxx', {
  from: '2024-01-01',
  to: '2024-12-31',
  limit: 50,
})
```

Bank Accounts

listBankAccounts

```
const accounts = await zenpays.merchants.listBankAccounts()
```

---

addBankAccount

```
const account = await zenpays.merchants.addBankAccount({
  accountNumber: '1234567890',
  ifscCode: 'HDFC0001234',
  accountHolderName: 'Business Pvt Ltd',
  bankName: 'HDFC Bank',
})
```

---

deleteBankAccount

```
await zenpays.merchants.deleteBankAccount('ba_xxx')
```

---

setDefaultBankAccount

```
const account = await zenpays.merchants.setDefaultBankAccount('ba_xxx')
```

IP Whitelist

listWhitelistedIPs

```
const ips = await zenpays.merchants.listWhitelistedIPs()
```

---

addIPToWhitelist

```
const entry = await zenpays.merchants.addIPToWhitelist('203.0.113.50', 'Production server')
```

---

removeIPFromWhitelist

```
await zenpays.merchants.removeIPFromWhitelist('ip_xxx')
```

SDK REFERENCE

Analytics

Access dashboard metrics, revenue trends, reports, and business insights. Access via `zenpays.analytics`.

Dashboard

getDashboardOverview

JAVASCRIPT

```
const overview = await zenpays.analytics.getDashboardOverview('30d')
```

PYTHON

```
overview = zenpays.analytics.get_dashboard_overview("30d")
```

---

getBusinessInsights

```
const insights = await zenpays.analytics.getBusinessInsights()
```

Revenue

getRevenueTrends

```
const trends = await zenpays.analytics.getRevenueTrends('30d', 'day') // granularity: 'hour' | 'day'
| 'week' | 'month'
```

---

getRevenueMetrics

```
const metrics = await zenpays.analytics.getRevenueMetrics('30d')
```

Payments

getPaymentMethodBreakdown

```
const breakdown = await zenpays.analytics.getPaymentMethodBreakdown('30d')
```

---

getPaymentStatusDistribution

```
const distribution = await zenpays.analytics.getPaymentStatusDistribution('30d')
```

getRecentPayments

```
const payments = await zenpays.analytics.getRecentPayments(10)
```

getFailedPaymentsAnalysis

```
const analysis = await zenpays.analytics.getFailedPaymentsAnalysis('30d')
```

Customers

getTopCustomers

```
const customers = await zenpays.analytics.getTopCustomers(10, '30d')
```

getRecentCustomerActivity

```
const activity = await zenpays.analytics.getRecentCustomerActivity(10)
// Returns: [{ customerId, name, lastActivity, type }]
```

Reports

generateReport

JAVASCRIPT

```
const report = await zenpays.analytics.generateReport({
  type: 'transactions',
  format: 'pdf',
  startDate: '2024-01-01',
  endDate: '2024-12-31',
  currency: 'INR',
})
```

PYTHON

```python
report = zenpays.analytics.generate_report({
    "type": "transactions",
    "format": "pdf",
    "startDate": "2024-01-01",
    "endDate": "2024-12-31",
    "currency": "INR",
})
```

listReports

```javascript
const { data } = await zenpays.analytics.listReports({ type: 'transactions', limit: 10 })
```

getReport

```javascript
const report = await zenpays.analytics.getReport('rpt_xxx')
```

downloadReport

```javascript
const downloadUrl = await zenpays.analytics.downloadReport('rpt_xxx')
```

Report Schedules

createReportSchedule

```javascript
const schedule = await zenpays.analytics.createReportSchedule({
  type: 'transactions',
  format: 'csv',
  frequency: 'weekly',
  recipients: ['finance@company.com'],
})
```

listReportSchedules

```javascript
const schedules = await zenpays.analytics.listReportSchedules()
```

updateReportSchedule

```
const updated = await zenpays.analytics.updateReportSchedule('sched_xxx', {
  frequency: 'monthly',
})
```

---

deleteReportSchedule

```
await zenpays.analytics.deleteReportSchedule('sched_xxx')
```

Ramp

SDK REFERENCE / RAMP

On-Ramp (Buy)

Buy crypto with fiat — checkout sessions, provider selection, quotes, rates, and configuration. Access via `zenpays.checkout`.

Widget Integration

generateWidgetUrl

Generate a signed widget URL for embedding on-ramp in your site.

JAVASCRIPT

```
const { url } = await zenpays.checkout.generateWidgetUrl({
  mode: 'buy',
  defaultFiat: 'usd',
  defaultCrypto: 'btc_bitcoin',
  defaultAmount: 100,
  networkWallets: 'ethereum:0x123...',
  successRedirectUrl: 'https://yoursite.com/success',
  failureRedirectUrl: 'https://yoursite.com/cancel',
  email: 'user@example.com',
})
// Embed url in iframe or redirect user
```

PYTHON

```
result = zenpays.checkout.generate_widget_url({
    "mode": "buy",
    "defaultFiat": "usd",
    "defaultCrypto": "btc_bitcoin",
    "defaultAmount": 100,
    "networkWallets": "ethereum:0x123...",
    "successRedirectUrl": "https://yoursite.com/success",
    "failureRedirectUrl": "https://yoursite.com/cancel",
})
```

---

selectProvider

Select the optimal provider for an on-ramp transaction.

JAVASCRIPT

```
const result = await zenpays.checkout.selectProvider({
  sessionId: 'cs_xxx',
  amount: 100,
  sourceCurrency: 'USD',
  destinationCurrency: 'USDT',
  country: 'US',
})
```

PYTHON

```
result = zenpays.checkout.select_provider({
    "sessionId": "cs_xxx",
    "amount": 100,
    "sourceCurrency": "USD",
    "destinationCurrency": "USDT",
    "country": "US",
})
```

getProviderWidget

Get the provider widget URL for embedding.

```
const widget = await zenpays.checkout.getProviderWidget('transak', 'cs_xxx')
```

getTransactionStatus

```
const status = await zenpays.checkout.getTransactionStatus('txn_xxx')
```

Checkout Sessions

createSession

JAVASCRIPT

```
const session = await zenpays.checkout.createSession({
  amount: 100,
  currency: 'USD',
  type: 'on_ramp',
  destinationCurrency: 'USDT',
  customerEmail: 'user@example.com',
})
```

PYTHON

```
session = zenpays.checkout.create_session({
    "amount": 100,
    "currency": "USD",
    "type": "on_ramp",
    "destinationCurrency": "USDT",
})
```

getSession

```
const session = await zenpays.checkout.getSession('cs_xxx')
```

cancelSession

```
const session = await zenpays.checkout.cancelSession('cs_xxx')
```

Ramp Orders

getRampOrder

```
const order = await zenpays.checkout.getRampOrder('ro_xxx')
```

getRampOrderStatus

```
const status = await zenpays.checkout.getRampOrderStatus('ro_xxx')
// Returns: { orderId, status, updatedAt }
```

updateRampOrder

```
const updated = await zenpays.checkout.updateRampOrder('ro_xxx', {
  walletAddress: '0x123...',
  network: 'ethereum',
})
```

Buy Quotes & Rates

getBuyQuotes

Get buy quotes for fiat-to-crypto conversion.

JAVASCRIPT

```
const quotes = await zenpays.checkout.getBuyQuotes({
  source: 'USD',
  destination: 'USDT',
  amount: 100,
  paymentMethod: 'card',
  country: 'US',
})
```

PYTHON

```
quotes = zenpays.checkout.get_buy_quotes({
    "source": "USD",
    "destination": "USDT",
    "amount": 100,
    "paymentMethod": "card",
    "country": "US",
})
```

getBuyRates

Get exchange rates for multiple crypto destinations.

```
const rates = await zenpays.checkout.getBuyRates({
  source: 'USD',
  destinations: 'USDT,BTC,ETH',
  amount: 100,
})
```

getBuyPrices

Get current crypto prices in fiat.

```
const prices = await zenpays.checkout.getBuyPrices({
  assets: 'BTC,ETH,USDT',
  fiat: 'USD',
  country: 'US',
})
```

createBuyCheckout

Create a buy checkout (fiat to crypto).

JAVASCRIPT

```
const checkout = await zenpays.checkout.createBuyCheckout({
  intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
  cryptoCurrency: 'USDT',
  chain: 'tron',
  fiatCurrency: 'USD',
  fiatAmount: 100,
  cryptoAmount: 98.5,
  rate: 1.0,
  totalFee: 1.5,
})
// Returns: { intentId, paymentIntentId, checkoutUrl, fiatAmount, fiatCurrency, cryptoAmount,
cryptoCurrency, chain, rate, expiresAt }
```

PYTHON

```
checkout = zenpays.checkout.create_buy_checkout({
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "cryptoCurrency": "USDT",
    "chain": "tron",
    "fiatCurrency": "USD",
    "fiatAmount": 100,
    "cryptoAmount": 98.5,
    "rate": 1.0,
    "totalFee": 1.5,
})
```

Configuration

getConfig

Get ramp provider configuration.

```
const config = await zenpays.checkout.getConfig()
```

getSupportedAssets

Get supported crypto and fiat currencies.

```
const assets = await zenpays.checkout.getSupportedAssets({ type: 'buy', country: 'US' })
```

getPaymentMethods

Get payment methods for a source currency.

```
const methods = await zenpays.checkout.getPaymentMethods('USD', {
  destination: 'USDT',
  country: 'US',
})
```

getDefaults

Get default currencies and amounts per country.

```
const defaults = await zenpays.checkout.getDefaults({ type: 'buy', country: 'US' })
```

Geo & Currency

detectGeo

```
const geo = await zenpays.checkout.detectGeo()
// Returns: { country, currency, timezone, ... }
```

---

getBanks

```
const banks = await zenpays.checkout.getBanks('INR')
```

---

searchBanks

```
const results = await zenpays.checkout.searchBanks('INR', 'HDFC')
```

---

getCurrencies

```
const currencies = await zenpays.checkout.getCurrencies()
```

SDK REFERENCE / RAMP

Off-Ramp

Sell crypto for fiat (crypto-to-fiat conversion). Access via `zenpays.offRamp`.

Methods

getQuotes

Get sell quotes for a crypto-to-fiat currency pair.

JAVASCRIPT

```
const quotes = await zenpays.offRamp.getQuotes({
  source: 'usdt_tron',
  destination: 'NGN',
  amount: 100,
  paymentMethod: 'bank_transfer',
  country: 'NG',
})
```

PYTHON

```
quotes = zenpays.off_ramp.get_quotes({
    "source": "usdt_tron",
    "destination": "NGN",
    "amount": 100,
    "paymentMethod": "bank_transfer",
    "country": "NG",
})
```

Response

Returns an array of quotes from available providers:

```
[
  {
    "onramp": "zenpay",
    "paymentMethod": "bank_transfer",
    "inputAmount": 100,
    "outputAmount": 157500,
    "rate": 1575,
    "networkFee": 0,
    "transactionFee": 1.5,
    "totalFee": 1.5,
    "recommended": true,
    "minAmount": 5,
    "maxAmount": 10000
  }
]
```

---

getRates

Get exchange rates for one or more fiat destinations.

```
const rates = await zenpays.offRamp.getRates({
  source: 'usdt_tron',
  destinations: 'NGN,GHS,KES',
  amount: 100,
})
```

---

createCheckout

Initiate a generic off-ramp checkout session (legacy).

JAVASCRIPT

```
const checkout = await zenpays.offRamp.createCheckout({
  onramp: 'zenpay',
  source: 'usdt_tron',
  destination: 'NGN',
  amount: 100,
  type: 'sell',
  paymentMethod: 'bank_transfer',
  walletAddress: '0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28',
})
```

PYTHON

```
checkout = zenpays.off_ramp.create_checkout({
    "onramp": "zenpay",
    "source": "usdt_tron",
    "destination": "NGN",
    "amount": 100,
    "type": "sell",
    "paymentMethod": "bank_transfer",
    "walletAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28",
})
```

---

createSellCheckout

Phase 1 of the two-phase sell flow — generate a crypto deposit address. Requires a ramp intent.

JAVASCRIPT

```
const sellCheckout = await zenpays.offRamp.createSellCheckout({
  intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
  cryptoCurrency: 'USDT',
  chain: 'tron',
  fiatCurrency: 'USD',
  cryptoAmount: 20,
  fiatAmount: 19.78,
  rate: 0.999,
  totalFee: 0.62,
})
```

PYTHON

```
sell_checkout = zenpays.off_ramp.create_sell_checkout({
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "cryptoCurrency": "USDT",
    "chain": "tron",
    "fiatCurrency": "USD",
    "cryptoAmount": 20,
    "fiatAmount": 19.78,
    "rate": 0.999,
    "totalFee": 0.62,
})
```

Parameters

Field	Type	Required	Description
`intentId`	string	Yes	Ramp intent ID
`cryptoCurrency`	string	Yes	Crypto to sell (e.g. `USDT`)
`chain`	string	Yes	Blockchain network (e.g. `tron`, `ethereum`)
`fiatCurrency`	string	Yes	Target fiat currency
`cryptoAmount`	number	Yes	Amount of crypto to sell
`fiatAmount`	number	Yes	Expected fiat proceeds
`rate`	number	Yes	Locked exchange rate
`totalFee`	number	Yes	Total fees
`customerEmail`	string	No	Customer email (for KYC, optional if `kycVerified`)
`customerFirstName`	string	No	Customer first name
`customerLastName`	string	No	Customer last name
`quoteId`	string	No	Quote ID from quotes endpoint

Response

```
{
  "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
  "depositAddress": "TLfMkVBKQSy8i1XN2wDCQxkPaL1Dqz7v98",
  "memo": null,
  "currency": "USDT",
  "chain": "tron",
  "expectedCryptoAmount": 20,
  "fiatAmount": 19.78,
  "fiatCurrency": "USD",
  "rate": 0.999,
  "cryptoDepositId": "d4e5f6a7-b8c9-1234-5678-abcdef012345",
  "expiresAt": "2026-03-15T10:00:00.000Z",
  "provider": "fireblocks"
}
```

getSellPayoutPreview

Phase 2a — discover required beneficiary fields and fees for the target currency/country.

JAVASCRIPT

```
const preview = await zenpays.offRamp.getSellPayoutPreview({
  intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
  fiatCurrency: 'USD',
  country: 'US',
  payoutType: 'bank_transfer',
})
```

PYTHON

```
preview = zenpays.off_ramp.get_sell_payout_preview({
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "fiatCurrency": "USD",
    "country": "US",
    "payoutType": "bank_transfer",
})
```

Parameters

Field	Type	Required	Description
`intentId`	string	Yes	Ramp intent ID
`fiatCurrency`	string	Yes	Target fiat currency
`country`	string	No	ISO country code
`payoutType`	string	No	Payout method (default: `bank_transfer`)

Response

```
{
  "selectedTsp": "zenpay_routing",
  "tspDisplayName": "PayCross",
  "requiredFields": [
    { "fieldName": "receiverFirstName", "label": "First Name", "type": "string", "required": true },
    { "fieldName": "receiverLastName", "label": "Last Name", "type": "string", "required": true },
    { "fieldName": "receiverAccountNumber", "label": "Account Number", "type": "string", "required":
true }
  ],
  "optionalFields": [],
  "fiatAmount": 19.78,
  "fiatCurrency": "USD",
  "fees": { "platform": 0.22, "network": 0, "total": 0.22 }
}
```

createSellPayout

Phase 2b — submit beneficiary details to trigger the fiat payout.

JAVASCRIPT

```
const payout = await zenpays.offRamp.createSellPayout({
  intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
  cryptoDepositId: 'd4e5f6a7-b8c9-1234-5678-abcdef012345',
  fiatCurrency: 'USD',
  country: 'US',
  selectedTsp: 'zenpay_routing',
  beneficiaryDetails: {
    receiverFirstName: 'John',
    receiverLastName: 'Doe',
    receiverAccountNumber: '123456789',
    receiverBankName: 'Bank of America',
    receiverCountry: 'US',
  },
})
```

PYTHON

```
payout = zenpays.off_ramp.create_sell_payout({
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "cryptoDepositId": "d4e5f6a7-b8c9-1234-5678-abcdef012345",
    "fiatCurrency": "USD",
    "country": "US",
    "selectedTsp": "zenpay_routing",
    "beneficiaryDetails": {
        "receiverFirstName": "John",
        "receiverLastName": "Doe",
        "receiverAccountNumber": "123456789",
        "receiverBankName": "Bank of America",
        "receiverCountry": "US",
    },
})
```

Parameters

Field	Type	Required	Description
`intentId`	string	Yes	Ramp intent ID
`cryptoDepositId`	string	Yes	Deposit ID from sell checkout response
`fiatCurrency`	string	Yes	Target fiat currency
`payoutType`	string	No	Payout method (default: `bank_transfer`)
`country`	string	No	ISO country code
`selectedTsp`	string	No	Provider from payout preview response
`beneficiaryDetails`	object	Yes	Dynamic fields from payout preview `requiredFields`

Response

```
{
  "payoutId": "po_abc123def456",
  "status": "processing",
  "amount": 19.78,
  "currency": "USD",
  "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8"
}
```

---

getTransaction

```
const txn = await zenpays.offRamp.getTransaction('txn_xxx')
```

---

createWidgetUrl

Generate a hosted widget URL for off-ramp.

JAVASCRIPT

```
const { url } = await zenpays.offRamp.createWidgetUrl({
  mode: 'sell',
  defaultFiat: 'usd',
  defaultCrypto: 'usdt_tron',
  defaultAmount: 100,
  networkWallets: 'tron:TLfMkVBKQSy...',
  successRedirectUrl: 'https://yoursite.com/success',
  failureRedirectUrl: 'https://yoursite.com/failure',
})
// Embed url in iframe or redirect user
```

PYTHON

```
result = zenpays.off_ramp.create_widget_url({
    "mode": "sell",
    "defaultFiat": "usd",
    "defaultCrypto": "usdt_tron",
    "defaultAmount": 100,
    "networkWallets": "tron:TLfMkVBKQSy...",
    "successRedirectUrl": "https://yoursite.com/success",
    "failureRedirectUrl": "https://yoursite.com/failure",
})
```

Parameters

Field	Type	Required	Description
`mode`	string	No	`buy`, `sell`, or `buy,sell`
`defaultFiat`	string	No	Default fiat currency
`defaultCrypto`	string	No	Default crypto
`defaultAmount`	number	No	Default amount
`networkWallets`	string	No	Format: `network:address,network:address`
`successRedirectUrl`	string	No	Redirect on success
`failureRedirectUrl`	string	No	Redirect on failure
`email`	string	No	Pre-fill customer email
`uuid`	string	No	User identifier

SDK REFERENCE / RAMP

Ramp Intents

Create and manage ramp intents — merchant-initiated buy/sell requests with embeddable widget URLs. Access via `zenpays.rampIntents`.

Methods

create

Create a new ramp intent.

Parameters

Field	Type	Required	Description
`userId`	string	Yes	Unique user identifier from your system — used as the KYC record key
`kycVerified`	boolean	Yes	When `true`, ZenPays KYC verification is skipped. When `false`, the widget may prompt the user for identity verification.
`type`	string	Yes	`buy` (fiat → crypto) or `sell` (crypto → fiat)
`wallets`	array	Yes	Array of `{ network, address }` objects (min 1)
`fiatCurrency`	string	Yes	Fiat currency ISO code (e.g. `USD`, `EUR`, `INR`)
`cryptoCurrency`	string	No	Pre-selected crypto (e.g. `btc_bitcoin`, `usdt_tron`)
`amount`	number	No	Pre-filled amount in the widget
`successUrl`	string	No	URL to redirect on success
`cancelUrl`	string	No	URL to redirect on cancel
`customerEmail`	string	No	Customer email for pre-filling
`metadata`	object	No	Custom key-value pairs (returned in webhooks)

JAVASCRIPT

```
const intent = await zenpays.rampIntents.create({
  userId: 'usr_abc123',
  kycVerified: false,
  type: 'buy',
  wallets: [
    { network: 'ethereum', address: '0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28' },
  ],
  fiatCurrency: 'USD',
  cryptoCurrency: 'eth_ethereum',
  amount: 100,
  successUrl: 'https://yoursite.com/success',
  cancelUrl: 'https://yoursite.com/cancel',
})
```

PYTHON

```
intent = zenpays.ramp_intents.create({
    "userId": "usr_abc123",
    "kycVerified": False,
    "type": "buy",
    "wallets": [
        {"network": "ethereum", "address": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28"}
    ],
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "successUrl": "https://yoursite.com/success",
    "cancelUrl": "https://yoursite.com/cancel",
})
```

Response

```
{
  "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
  "redirectUrl":
"https://checkout.zenpays.com/payments/ramping/widget/ri_1710345678000_a1b2c3d4e5f6g7h8",
  "expiresAt": "2026-03-15T10:00:00.000Z"
}
```

Field	Type	Description
`intentId`	string	Unique intent identifier (`ri_{timestamp}_{hex}`)
`redirectUrl`	string	URL to redirect the customer to the ramp widget
`expiresAt`	string	ISO 8601 expiry timestamp (1 hour from creation)

list

JAVASCRIPT

```
const { data, meta } = await zenpays.rampIntents.list({
  type: 'buy',
  status: 'active',
  page: 1,
  limit: 20,
})
```

PYTHON

```
result = zenpays.ramp_intents.list({"type": "buy", "status": "active", "limit": 20})
```

Filters

Field	Type	Description
`page`	number	Page number (default: 1)
`limit`	number	Items per page (default: 20, max: 100)
`status`	string	Filter by status: `created`, `active`, `completed`, `expired`, `cancelled`
`type`	string	Filter by direction: `buy` or `sell`
`search`	string	Search by intent ID or customer email

---

get

Retrieve intent details including widget URL and KYC status.

```
const result = await zenpays.rampIntents.get('ri_xxx')
```

Response


```
{
  "intent": {
    "intentId": "ri_xxx",
    "type": "buy",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "status": "active",
    "kycStatus": "approved",
    "cancelReason": null,
    "wallets": [{ "network": "ethereum", "address": "0x742d..." }],
    "userId": "usr_abc123",
    "kycVerified": true,
    "successUrl": "https://yoursite.com/success",
    "cancelUrl": "https://yoursite.com/cancel",
    "expiresAt": "2026-03-15T10:00:00.000Z"
  },
  "widgetUrl": "https://widget.onramper.com/...signed_url...",
  "kycRequired": false,
  "kycVerificationUrl": null
}
```


Field	Type	Description
`intent`	object	Full intent details
`intent.status`	string	Lifecycle state: `created`, `active`, `completed`, `expired`, or `cancelled`
`intent.kycStatus`	string	KYC sub-state: `pending`, `in_review`, `approved`, `declined`, or `expired`
`intent.cancelReason`	string | null	Set when `status='cancelled'`. `kyc_rejected` when KYC was declined; `kyc_expired` for expiry; otherwise a free-form string.
`widgetUrl`	string | null	Signed widget URL for iframe embedding. `null` if KYC required or expired.
`kycRequired`	boolean	Whether ZenPays KYC verification is needed
`kycVerificationUrl`	string | null	Verification URL (call `initiateKyc` if null and KYC is required)

Distinguishing decline reasons

When a customer's KYC is rejected, `intent.status` flips to `'cancelled'` and `intent.cancelReason` becomes `'kyc_rejected'`. You can also drive your UI off `intent.kycStatus === 'declined'` directly — both are set together.

updateStatus

```
const updated = await zenpays.rampIntents.updateStatus('ri_xxx', 'cancelled')
```

---

initiateKyc

Start ZenPays KYC verification for a ramp intent. Call this when `kycRequired` is `true` and `kycVerificationUrl` is `null`.

JAVASCRIPT

```
const kyc = await zenpays.rampIntents.initiateKyc('ri_xxx', {
  email: 'user@example.com',
  firstName: 'John',
  lastName: 'Doe',
})

if (kyc.kycVerificationUrl) {
  // Redirect user to verification
  window.location.href = kyc.kycVerificationUrl
}
```

PYTHON

```
kyc = zenpays.ramp_intents.initiate_kyc("ri_xxx", {
    "email": "user@example.com",
    "firstName": "John",
    "lastName": "Doe",
})

if kyc["kycVerificationUrl"]:
    print(f"Redirect to: {kyc['kycVerificationUrl']}")
```

Response

```
{
  "kycRequired": true,
  "kycVerificationUrl": "https://verify.zenpays.com/session/abc123",
  "sessionId": "abc123"
}
```

---

getKycStatus

Poll the current KYC verification status for a ramp intent. This is the same endpoint the checkout widget polls every 5 seconds while a customer is verifying — call it the same way from your own UI if you embed the widget yourself.

JAVASCRIPT

```
const status = await zenpays.rampIntents.getKycStatus('ri_xxx')

switch (status.kycStatus) {
  case 'approved':
    // KYC passed — proceed with the checkout
    break
  case 'in_review':
    // Identity OK but flagged (often AML) — keep waiting, surface "under review"
    break
  case 'declined':
  case 'expired':
    // Terminal failure — `status.intentStatus` is now 'cancelled' and
    // `status.cancelReason` will tell you why ('kyc_rejected' or 'kyc_expired')
    break
  case 'pending':
  default:
    // Verification not yet completed — keep polling (or call initiateKyc)
    break
}
```

PYTHON

```
status = zenpays.ramp_intents.get_kyc_status("ri_xxx")

if status["kycStatus"] == "approved":
    print("KYC verified — proceed")
elif status["kycStatus"] == "in_review":
    print("Identity OK, flagged for review — keep waiting")
elif status["kycStatus"] in ("declined", "expired"):
    print(f"Terminal: {status['cancelReason']}")
else:
    print("Still pending — keep polling")
```

Response


```
{
  "verified": false,
  "kycStatus": "in_review",
  "intentStatus": "active",
  "cancelReason": null
}
```

Field	Type	Description
`verified`	boolean	Convenience flag — `true` when `kycStatus === 'approved'`
`kycStatus`	string	`pending`, `in_review`, `approved`, `declined`, or `expired`
`intentStatus`	string	Current ramp intent lifecycle state (`created`, `active`, `completed`, `expired`, `cancelled`)
`cancelReason`	string | null	Set when the intent moved to `cancelled` because of KYC. `kyc_rejected` for declines, `kyc_expired` for expiry.

Understanding KYC states

`kycStatus`	What it means	What to do
`pending`	Customer hasn't completed the Didit flow yet	Keep polling. Optionally show a "verification in progress" UI.
`in_review`	Identity passed but AML or face-match was flagged for manual review	Keep polling. Surface a friendly "under review" message — this can take a few minutes.
`approved`	KYC and AML both cleared	Proceed with the checkout / surface `widgetUrl`.
`declined`	Hard rejection (identity mismatch, document issue, AML rejected)	Stop polling. `intentStatus` is now `cancelled` with `cancelReason='kyc_rejected'`. Show a terminal "verification could not be completed" message.
`expired`	Verification session expired before the customer finished	Stop polling. Same handling as `declined`. Optionally offer to start a new intent.

Polling cadence

The checkout widget polls every 5 seconds with a hard ceiling of ~5 minutes (60 attempts) before showing a "still confirming your verification" message. If you implement your own polling, mirror that pattern so you don't leave customers on a spinner indefinitely.

SDK REFERENCE

Vendors

Manage vendors/partners, track commissions, referrals, and process payouts. Access via `zenpays.vendors`.

Vendor Management

create

JAVASCRIPT

```
const vendor = await zenpays.vendors.create({
  name: 'Acme Corp',
  email: 'vendor@acme.com',
  phone: '+911234567890',
  country: 'IN',
  commissionRate: 5, // percentage
})
```

PYTHON

```
vendor = zenpays.vendors.create({
    "name": "Acme Corp",
    "email": "vendor@acme.com",
    "phone": "+911234567890",
    "country": "IN",
    "commissionRate": 5,
})
```

---

list

JAVASCRIPT

```
const { data } = await zenpays.vendors.list({
  search: 'acme',
  status: 'active',
  page: 1,
  limit: 20,
})
```

PYTHON

```
result = zenpays.vendors.list({"status": "active", "limit": 20})
```

---

getStats

```
const stats = await zenpays.vendors.getStats()
```

---

get

```
const vendor = await zenpays.vendors.get('vnd_xxx')
```

---

update

```
const updated = await zenpays.vendors.update('vnd_xxx', { commissionRate: 7.5 })
```

---

delete

Soft-delete a vendor.

```
await zenpays.vendors.delete('vnd_xxx')
```

---

updateStatus

```
const updated = await zenpays.vendors.updateStatus('vnd_xxx', 'suspended')
// status: 'active' | 'inactive' | 'suspended'
```

KYC

updateKyc

```
const updated = await zenpays.vendors.updateKyc('vnd_xxx', {
  verificationLevel: 'advanced',
  documents: ['doc_xxx'],
})
```

Commissions & Referrals

listCommissions

```
const commissions = await zenpays.vendors.listCommissions('vnd_xxx')
```

---

listReferrals

```
const referrals = await zenpays.vendors.listReferrals('vnd_xxx')
```

---

createReferral

JAVASCRIPT

```
const referral = await zenpays.vendors.createReferral('vnd_xxx', {
  referredMerchantId: 'mer_xxx',
  referralCode: 'ACME2024',
})
```

PYTHON

```
referral = zenpays.vendors.create_referral("vnd_xxx", {
    "referredMerchantId": "mer_xxx",
    "referralCode": "ACME2024",
})
```

Payouts & Analytics

processPayout

```
const payout = await zenpays.vendors.processPayout('vnd_xxx', {
  amount: 5000,
  currency: 'INR',
})
```

---

getAnalytics

```
const analytics = await zenpays.vendors.getAnalytics('vnd_xxx')
```

SDK REFERENCE

Chargebacks

Manage chargebacks, submit evidence, and track dispute resolution.

Methods

create

JAVASCRIPT

```
const chargeback = await zenpays.chargebacks.create({
  transactionId: 'txn_xxx',
  amount: 500,
  reason: 'product_not_received',
  description: 'Customer did not receive the product',
})
```

PYTHON

```
chargeback = zenpays.chargebacks.create({
    "transactionId": "txn_xxx",
    "amount": 500,
    "reason": "product_not_received",
    "description": "Customer did not receive the product",
})
```

get

JAVASCRIPT

```
const chargeback = await zenpays.chargebacks.get('chg_xxx')
```

PYTHON

```
chargeback = zenpays.chargebacks.get("chg_xxx")
```

list

JAVASCRIPT

```
const { data } = await zenpays.chargebacks.list({
  status: 'open',
  page: 1,
  limit: 20,
})
```

PYTHON

```
result = zenpays.chargebacks.list({"status": "open", "page": 1, "limit": 20})
data = result["data"]
```

updateStatus

```
const updated = await zenpays.chargebacks.updateStatus('chg_xxx', 'under_review', 'Reviewing
evidence')
```

submitEvidence

```
const result = await zenpays.chargebacks.submitEvidence('chg_xxx', {
  evidenceType: 'delivery_proof',
  description: 'Delivery confirmation from courier',
  documentUrl: 'https://example.com/proof.pdf',
})
```

addMerchantResponse

```
const result = await zenpays.chargebacks.addMerchantResponse('chg_xxx', {
  response: 'The product was delivered on time. See tracking proof.',
  documentUrls: ['https://example.com/tracking.pdf'],
})
```

resolve

```
const resolved = await zenpays.chargebacks.resolve('chg_xxx', {
  resolution: 'accepted',
  notes: 'Refund issued to customer',
})
```

cancel

```
const cancelled = await zenpays.chargebacks.cancel('chg_xxx')
```

getStats

```
const stats = await zenpays.chargebacks.getStats('2024-01-01', '2024-12-31', 'INR')
```

getAnalytics

```
const analytics = await zenpays.chargebacks.getAnalytics('2024-01-01', '2024-12-31', 'month', 'INR')
```

export

```
const downloadUrl = await zenpays.chargebacks.export({ status: 'open' })
```

bulkUpload

```
const result = await zenpays.chargebacks.bulkUpload(csvFile, 'https://yoursite.com/webhook')
```

Billing

SDK REFERENCE / BILLING

Invoices

Create, manage, and send invoices to customers. Access via `zenpays.invoices`.

Methods

create

JAVASCRIPT

```
const invoice = await zenpays.invoices.create({
  customerId: 'cust_xxx',
  items: [
    { description: 'Premium Plan', amount: 999, quantity: 1 },
  ],
  currency: 'INR',
  dueDate: '2024-12-31',
})
```

PYTHON

```
invoice = zenpays.invoices.create({
    "customerId": "cust_xxx",
    "items": [
        {"description": "Premium Plan", "amount": 999, "quantity": 1},
    ],
    "currency": "INR",
    "dueDate": "2024-12-31",
})
```

---

list

JAVASCRIPT

```
const { data } = await zenpays.invoices.list({
  status: 'pending',
  page: 1,
  limit: 20,
})
```

PYTHON

```
result = zenpays.invoices.list({"status": "pending", "limit": 20})
```

---

getAnalytics

```
const analytics = await zenpays.invoices.getAnalytics()
```

---

get

```
const invoice = await zenpays.invoices.get('inv_xxx')
```

---

update

```
const updated = await zenpays.invoices.update('inv_xxx', {
  dueDate: '2025-01-31',
})
```

---

send

Send an invoice to the customer via email.

```
const result = await zenpays.invoices.send('inv_xxx')
```

---

cancel

```
const cancelled = await zenpays.invoices.cancel('inv_xxx')
```

---

generateLink

Generate a shareable payment link for an invoice.

```
const { url } = await zenpays.invoices.generateLink('inv_xxx')
// Returns: { url: 'https://pay.zenpays.com/invoice/inv_xxx' }
```

SDK REFERENCE / BILLING

Payment Links

Create and manage shareable payment links. Access via `zenpays.paymentLinks`.

Methods

create

JAVASCRIPT

```
const link = await zenpays.paymentLinks.create({
  name: 'Product Purchase',
  amount: 500,
  currency: 'INR',
  description: 'Payment for premium plan',
  expiresAt: '2024-12-31',
})
```

PYTHON

```
link = zenpays.payment_links.create({
    "name": "Product Purchase",
    "amount": 500,
    "currency": "INR",
    "description": "Payment for premium plan",
})
```

---

list

JAVASCRIPT

```
const { data } = await zenpays.paymentLinks.list({
  search: 'premium',
  status: 'active',
  page: 1,
  limit: 20,
})
```

PYTHON

```
result = zenpays.payment_links.list({"status": "active", "limit": 20})
```

---

getAnalytics

```
const analytics = await zenpays.paymentLinks.getAnalytics()
```

---

get

```
const link = await zenpays.paymentLinks.get('pl_xxx')
```

---

update

```
const updated = await zenpays.paymentLinks.update('pl_xxx', {
  name: 'Updated Name',
  amount: 1000,
})
```

---

deactivate

```
const deactivated = await zenpays.paymentLinks.deactivate('pl_xxx')
```

---

listInvoices

```
const invoices = await zenpays.paymentLinks.listInvoices('pl_xxx')
```

SDK REFERENCE / BILLING

Subscriptions

Recurring billing — plans, stored payment methods, and the subscription lifecycle. Access via `zenpays.subscriptions`.

> **JavaScript only**
> The Python SDK does not yet expose a `subscriptions` namespace. Call the REST endpoints directly from Python.

Plans

A plan defines *what* is charged and *how often*. Subscriptions attach a customer to a plan.

createPlan

```
const plan = await zenpays.subscriptions.createPlan({
  name: 'Pro Monthly',
  amount: 2999,
  currency: 'USD',
  interval: 'month',
  description: 'Pro tier, billed monthly',
  intervalCount: 1, // 3 + 'month' would be quarterly
  trialDays: 14,
})
```

Parameter	Type	Required	Description
`name`	`string`	Yes	Plan name shown to customers
`amount`	`number`	Yes	Amount charged each interval, in the currency's minor unit
`currency`	`string`	Yes	ISO 4217 currency code
`interval`	`'day' | 'week' | 'month' | 'year'`	Yes	Billing interval
`description`	`string`	No	Longer plan description
`intervalCount`	`number`	No	Intervals between charges. Defaults to `1`
`trialDays`	`number`	No	Free trial length in days
`metadata`	`Record<string, unknown>`	No	Arbitrary key/value data

Returns a `SubscriptionPlan`.

listPlans

```
const plans = await zenpays.subscriptions.listPlans()
```

Returns `SubscriptionPlan[]` for the authenticated merchant.

archivePlan

Stops new subscriptions on the plan. Existing subscriptions continue to bill.

```
const plan = await zenpays.subscriptions.archivePlan('plan_xxx')
```

getPublicPlan

Unauthenticated plan details, safe to render on a hosted subscribe page.

```
const plan = await zenpays.subscriptions.getPublicPlan('plan_xxx')
```

Payment methods

To charge a subscription automatically you need a stored payment method. Collect one on the client with a setup token, then store the resulting provider token.

createSetupToken

```
const setupToken = await zenpays.subscriptions.createSetupToken({
  customerId: 'cust_xxx',
  customerEmail: 'customer@example.com',
  currency: 'USD',
})
```

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer the method will belong to
`customerEmail`	`string`	Yes	Customer email
`currency`	`string`	No	Currency the method will be charged in
`tspProvider`	`string`	No	Force a specific payment provider

storePaymentMethod

```
const method = await zenpays.subscriptions.storePaymentMethod({
  customerId: 'cust_xxx',
  customerEmail: 'customer@example.com',
  tspToken: 'tok_from_provider',
  tspProvider: 'stripe',
  type: 'card',
  last4: '4242',
  brand: 'visa',
  expiryMonth: 12,
  expiryYear: 2028,
})
```

`tspToken` is the provider token representing the instrument the customer entered. Raw card details never reach ZenPays.

listPaymentMethods

```
const methods = await zenpays.subscriptions.listPaymentMethods('cust_xxx')
```

revokePaymentMethod

```
await zenpays.subscriptions.revokePaymentMethod('spm_xxx')
```

Subscription lifecycle

create

```
const subscription = await zenpays.subscriptions.create({
  customerId: 'cust_xxx',
  customerEmail: 'customer@example.com',
  planId: 'plan_xxx',
  storedPaymentMethodId: 'spm_xxx',
})
```

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer to subscribe
`customerEmail`	`string`	Yes	Customer email
`planId`	`string`	Yes	Plan to subscribe them to
`storedPaymentMethodId`	`string`	No	Method to charge. Omit to require setup first
`tspProvider`	`string`	No	Force a specific payment provider
`metadata`	`Record<string, unknown>`	No	Arbitrary key/value data

list

```
const all = await zenpays.subscriptions.list()
const active = await zenpays.subscriptions.list('active')
```

Optionally filtered by `SubscriptionStatus`.

get

```
const subscription = await zenpays.subscriptions.get('sub_xxx')
```

cancel

Cancels at the end of the current billing period by default.

```
// At period end — the customer keeps access until currentPeriodEnd
await zenpays.subscriptions.cancel('sub_xxx')

// Immediately
await zenpays.subscriptions.cancel('sub_xxx', { immediately: true })
```

pause

```
const subscription = await zenpays.subscriptions.pause('sub_xxx')
```

resume

```
const subscription = await zenpays.subscriptions.resume('sub_xxx')
```

Types

SubscriptionStatus

`'trialing'` · `'active'` · `'past_due'` · `'paused'` · `'cancelled'`

SubscriptionPlan

Field	Type	Description
`planId`	`string`	Plan identifier
`merchantId`	`string`	Owning merchant
`name`	`string`	Plan name
`description`	`string?`	Plan description
`amount`	`number`	Amount per interval
`currency`	`string`	ISO 4217 currency code
`interval`	`BillingInterval`	`'day' | 'week' | 'month' | 'year'`
`intervalCount`	`number`	Intervals between charges
`trialDays`	`number`	Free trial length in days
`status`	`'active' | 'archived'`	Plan status
`metadata`	`Record<string, unknown>?`	Arbitrary data

Subscription

Field	Type	Description
`subscriptionId`	`string`	Subscription identifier
`merchantId`	`string`	Owning merchant
`customerId`	`string`	Subscribed customer

Field	Type	Description
`customerEmail`	`string`	Customer email
`planId`	`string`	Plan being billed
`status`	`SubscriptionStatus`	Current status
`collectionMethod`	`'auto_charge' | 'send_invoice'`	How each period is collected
`storedPaymentMethodId`	`string?`	Method charged automatically
`tspProvider`	`string?`	Payment provider
`currentPeriodStart`	`string`	ISO 8601 timestamp
`currentPeriodEnd`	`string`	ISO 8601 timestamp
`nextBillingAt`	`string`	ISO 8601 timestamp of the next charge
`trialEnd`	`string?`	ISO 8601 timestamp the trial ends
`cancelAtPeriodEnd`	`boolean`	Whether cancellation is pending
`cancelledAt`	`string?`	ISO 8601 timestamp of cancellation
`pausedAt`	`string?`	ISO 8601 timestamp of pause

StoredPaymentMethod

Field	Type	Description
`storedPaymentMethodId`	`string`	Method identifier
`merchantId`	`string`	Owning merchant
`customerId`	`string`	Owning customer
`customerEmail`	`string`	Customer email
`tspProvider`	`string`	Payment provider
`type`	`string`	Instrument type, e.g. `card`
`brand`	`string?`	Card brand
`expiryMonth`	`number?`	Card expiry month
`expiryYear`	`number?`	Card expiry year
`isActive`	`boolean`	Whether the method can still be charged
`metadata`	`Record<string, unknown>?`	Arbitrary data

SDK REFERENCE

Authentication

Merchant dashboard authentication — login, 2FA, password management, and session handling.

Methods

login

JAVASCRIPT

```
const result = await zenpays.auth.login({
  email: 'merchant@example.com',
  password: 'your-password',
})

if (result.requires2FA) {
  // Prompt for 2FA code
  const verified = await zenpays.auth.verify2FALogin({
    tempToken: result.tempToken,
    code: '123456',
  })
}
```

PYTHON

```
result = zenpays.auth.login({
    "email": "merchant@example.com",
    "password": "your-password",
})

if result.get("requires2FA"):
    verified = zenpays.auth.verify_2fa_login({
        "tempToken": result["tempToken"],
        "code": "123456",
    })
```

verify2FALogin

```
const result = await zenpays.auth.verify2FALogin({
  tempToken: 'temp_xxx',
  code: '123456',
})
```

refreshToken

```
const tokens = await zenpays.auth.refreshToken({
  refreshToken: 'refresh_xxx',
})
```

logout

```
await zenpays.auth.logout()
```

changePassword

```
await zenpays.auth.changePassword({
  currentPassword: 'old-password',
  newPassword: 'new-password',
})
```

forgotPassword

```
await zenpays.auth.forgotPassword({
  email: 'merchant@example.com',
})
```

verifyResetToken

```
const result = await zenpays.auth.verifyResetToken({
  token: 'reset_xxx',
})
```

resetPassword

```
await zenpays.auth.resetPassword({
  token: 'reset_xxx',
  newPassword: 'new-password',
})
```

SDK REFERENCE

Security

Manage two-factor authentication, security events, and KYC documents. Access via `zenpays.security`.

Two-Factor Authentication

setup2FA

```
const setup = await zenpays.security.setup2FA()
// Returns: { secret, qrCodeUrl, backupCodes }
```

---

enable2FA

Enable 2FA after scanning the QR code.

```
const result = await zenpays.security.enable2FA('123456') // TOTP code
// Returns: { enabled: boolean }
```

---

verify2FA

```
const result = await zenpays.security.verify2FA({ code: '123456', type: 'totp' })
// Returns: { verified: boolean }
```

---

disable2FA

```
await zenpays.security.disable2FA('123456') // current TOTP code to confirm
```

---

is2FAEnabled

```
const status = await zenpays.security.is2FAEnabled()
// Returns: { enabled: boolean, setupAt?: string }
```

---

regenerateBackupCodes

```
const result = await zenpays.security.regenerateBackupCodes()
// Returns: { backupCodes: string[] }
```

Security Events

listSecurityEvents

JAVASCRIPT

```
const { data } = await zenpays.security.listSecurityEvents({
  severity: 'high',
  from: '2024-01-01',
  limit: 50,
})
```

PYTHON

```
result = zenpays.security.list_security_events({
    "severity": "high",
    "from": "2024-01-01",
    "limit": 50,
})
```

---

getSecurityEvent

```
const event = await zenpays.security.getSecurityEvent('sec_xxx')
```

KYC Documents

getKycStatus

```
const status = await zenpays.security.getKycStatus()
```

---

uploadKycDocument

JAVASCRIPT

```
const doc = await zenpays.security.uploadKycDocument({
  type: 'identity_proof',
  file: fileBlob,
  description: 'Passport front page',
})
```

PYTHON

```
doc = zenpays.security.upload_kyc_document({
    "type": "identity_proof",
    "file": file_data,
    "description": "Passport front page",
})
```

---

getKycDocument

```
const doc = await zenpays.security.getKycDocument('kyc_xxx')
```

---

downloadKycDocument

```
const downloadUrl = await zenpays.security.downloadKycDocument('kyc_xxx')
```

---

deleteKycDocument

```
await zenpays.security.deleteKycDocument('kyc_xxx')
```

SDK REFERENCE

Tenant Hostnames

White-label hostname management — map your own domain to your ZenPays merchant account and brand the hosted pages served on it. Access via `zenpays.tenantHostnames`.

> **JavaScript only**
> The Python SDK does not yet expose a `tenantHostnames` namespace. Call the REST endpoints directly from Python.

Methods

create

Map a hostname to the authenticated merchant.

```
const hostname = await zenpays.tenantHostnames.create({
  hostname: 'pay.yourbrand.com',
  slug: 'yourbrand',
})
```

Point the hostname at ZenPays with a CNAME before creating the mapping, or the hosted pages will not resolve.

list

```
const hostnames = await zenpays.tenantHostnames.list()
```

resolve

Look up which merchant a hostname belongs to. Useful when your own edge needs to route a request before it reaches ZenPays.

```
const result = await zenpays.tenantHostnames.resolve('pay.yourbrand.com')
// { found: true, merchantId: 'mrc_xxx', slug: 'yourbrand' }
```

Field	Type	Description
`found`	`boolean`	Whether the hostname is mapped
`merchantId`	`string?`	Owning merchant, when found
`slug`	`string?`	Tenant slug, when found

get

```
const hostname = await zenpays.tenantHostnames.get('th_xxx')
```

updateStatus

Enable or disable a hostname without deleting the mapping.

```
await zenpays.tenantHostnames.updateStatus('th_xxx', 'inactive')
```

`status` is `'active'` or `'inactive'`.

updateBranding

Set the logo, colours, and theme used on pages served from this hostname.

```
await zenpays.tenantHostnames.updateBranding('th_xxx', {
  logoUrl: 'https://cdn.yourbrand.com/logo.svg',
  primaryColor: '#0f172a',
  theme: 'light',
})
```

delete

```
await zenpays.tenantHostnames.delete('th_xxx')
```

Removes the mapping permanently. Pages served on the hostname stop resolving to your merchant immediately — use `updateStatus` if you only want to disable it temporarily.

PART

REST API

170 documents

REST API

REST API Overview

The ZenPays REST API provides direct HTTP access to all payment processing functionality. Use this API when you need more control than the SDK provides, or when working in a language without an official SDK.

Base URL

Environment	Base URL	Description
Sandbox	`https://api.zenpays.com`	Testing and development

API Versioning

All API endpoints are versioned. The current version is `v1`.

```
https://api.zenpays.com/api/v1/{endpoint}
```

Request Format

All requests must:

- Use HTTPS
- Include authentication headers (see Authentication)
- Send request bodies as JSON with `Content-Type: application/json`

Response Format

All responses are JSON with the following structure:

Success Response

```
{
  "success": true,
  "data": {
    // Response data
  }
}
```

Error Response

```
{
  "success": false,
  "error": {
    "code": "INVALID_REQUEST",
    "message": "The amount field is required",
    "details": {
      "field": "amount",
      "reason": "required"
    }
  }
}
```

HTTP Methods

Method	Usage
`GET`	Retrieve resources
`POST`	Create resources
`PUT`	Update resources (full replacement)
`PATCH`	Update resources (partial)
`DELETE`	Delete resources

Pagination

List endpoints support pagination with the following query parameters:

Parameter	Type	Default	Description
`page`	integer	1	Page number
`limit`	integer	20	Items per page (max 100)

Paginated responses include metadata:

```
{
  "success": true,
  "data": [...],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

Rate Limits

API requests are rate-limited to prevent abuse:

Environment	Limit
Production	1,000 requests/minute
Sandbox	100 requests/minute

Rate limit headers are included in all responses:

```
X-RateLimit-Limit: 1000
X-RateLimit-Remaining: 999
X-RateLimit-Reset: 1640000000
```

SDK vs REST API

Use Case	Recommended
Quick integration	SDK
Custom language support	REST API
Webhooks handling	REST API
Server-to-server calls	Both
Browser/client-side	SDK (handles signatures)

Quick Start

CURL

```
# Create a payment intent
curl -X POST https://api.zenpays.com/api/v1/payment-intents \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: $(date -u +%Y-%m-%dT%H:%M:%S.000Z)" \
  -H "X-Signature: your_hmac_signature" \
  -H "X-Secret-Salt: $SECRET_SALT" \
  -H "Content-Type: application/json" \
  -d '{
    "amount": 1000,
    "currency": "USD",
    "description": "Order #123"
  }'
```

JAVASCRIPT

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

const apiKey = process.env.ZENPAYS_API_KEY;
const secretSalt = process.env.ZENPAYS_SECRET_SALT;
const timestamp = new Date().toISOString();
const body = {
  amount: 1000,
  currency: 'USD',
  description: 'Order #123'
};

// Generate HMAC signature
const data = timestamp + JSON.stringify(body);
const signature = crypto
  .createHmac('sha256', secretSalt)
  .update(data)
  .digest('hex');

const response = await fetch('https://api.zenpays.com/api/v1/payment-intents', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(body),
});

const result = await response.json();
console.log(result);
```

PYTHON

```python
import hmac
import hashlib
import json
import os
from datetime import datetime
import requests

api_key = os.environ["ZENPAYS_API_KEY"]
secret_salt = os.environ["ZENPAYS_SECRET_SALT"]
timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")

body = {
    "amount": 1000,
    "currency": "USD",
    "description": "Order #123"
}

# Generate HMAC signature
data = timestamp + json.dumps(body, separators=(",", ":"))
signature = hmac.new(
    secret_salt.encode(),
    data.encode(),
    hashlib.sha256
).hexdigest()

response = requests.post(
    "https://api.zenpays.com/api/v1/payment-intents",
    headers={
        "Authorization": f"Bearer {api_key}",
        "X-Timestamp": timestamp,
        "X-Signature": signature,
        "X-Secret-Salt": secret_salt,
        "Content-Type": "application/json",
    },
    json=body,
)

print(response.json())
```

Next Steps

- Authentication - Learn how to authenticate requests
- Error Handling - Understand error responses
- SDK Reference - Use the SDK for easier integration

REST API

REST API Authentication

All REST API requests must include authentication headers. ZenPays uses **API Key + HMAC Signature** authentication for secure request verification.

Required Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}` - Your API key
`X-Signature`	Yes	HMAC-SHA256 signature of the request
`X-Timestamp`	Yes	ISO 8601 timestamp (UTC)
`X-Secret-Salt`	Yes	Your secret salt for HMAC signature validation
`Content-Type`	Yes*	`application/json` for POST/PUT/PATCH requests
`X-Merchant-ID`	No	Optional merchant ID for multi-merchant setups

API Keys

ZenPays uses prefixed API keys to identify the environment:

Prefix	Environment	Usage
`zp_live_`	Production	Real payments
`zp_test_`	Sandbox	Testing and development

Getting Your API Key

1. Log in to your ZenPays Dashboard
2. Navigate to **Settings → API Keys**
3. Copy your API key and Secret Salt

> **Warning**
> Keep your API keys and Secret Salt secure. Never expose them in client-side code or commit them to version control.

Signature Generation

Every request must include an HMAC-SHA256 signature to verify its integrity.

Signature Formula

```
signature = HMAC-SHA256(timestamp + body, secret_salt)
```

Where:

- `timestamp` - The value of the `X-Timestamp` header
- `body` - JSON stringified request body (empty string for GET requests)
- `secret_salt` - Your secret salt from the dashboard

Implementation Examples

JAVASCRIPT

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

function generateSignature(timestamp, body, secretSalt) {
  const data = timestamp + JSON.stringify(body);
  return crypto
    .createHmac('sha256', secretSalt)
    .update(data)
    .digest('hex');
}

// Example usage
const timestamp = new Date().toISOString();
const body = { amount: 1000, currency: 'USD' };
const signature = generateSignature(
  timestamp,
  body,
  process.env.ZENPAYS_SECRET_SALT
);

console.log('Signature:', signature);
// Output: a1b2c3d4e5f6...
```

PYTHON

```
import hmac
import hashlib
import json
import os
from datetime import datetime

def generate_signature(timestamp: str, body: dict, secret_salt: str) -> str:
    data = timestamp + json.dumps(body, separators=(",", ":"))
    return hmac.new(
        secret_salt.encode(),
        data.encode(),
        hashlib.sha256
    ).hexdigest()

# Example usage
timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")
body = {"amount": 1000, "currency": "USD"}
signature = generate_signature(
    timestamp,
    body,
    os.environ["ZENPAYS_SECRET_SALT"]
)

print(f"Signature: {signature}")
# Output: a1b2c3d4e5f6...
```

GO

```go
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "encoding/json"
    "time"
)

func generateSignature(timestamp string, body interface{}, secretSalt string) string {
    bodyBytes, _ := json.Marshal(body)
    data := timestamp + string(bodyBytes)

    h := hmac.New(sha256.New, []byte(secretSalt))
    h.Write([]byte(data))
    return hex.EncodeToString(h.Sum(nil))
}

// Example usage
func main() {
    timestamp := time.Now().UTC().Format("2006-01-02T15:04:05.000Z")
    body := map[string]interface{}{
        "amount":   1000,
        "currency": "USD",
    }
    signature := generateSignature(timestamp, body, "your_secret_salt")
    // Use signature in X-Signature header
}
```

PHP

```php
<?php

function generateSignature(string $timestamp, array $body, string $secretSalt): string {
    $data = $timestamp . json_encode($body, JSON_UNESCAPED_SLASHES);
    return hash_hmac('sha256', $data, $secretSalt);
}

// Example usage
$timestamp = gmdate('Y-m-d\TH:i:s.000\Z');
$body = ['amount' => 1000, 'currency' => 'USD'];
$signature = generateSignature(
    $timestamp,
    $body,
    getenv('ZENPAYS_SECRET_SALT')
);

echo "Signature: $signature\n";
```

Complete Request Example

CURL

```
#!/bin/bash

API_KEY="zp_test_xxxxx"
SECRET_SALT="your_secret_salt"
TIMESTAMP=$(date -u +"%Y-%m-%dT%H:%M:%S.000Z")

BODY='{"amount":1000,"currency":"USD","description":"Order #123"}'

# Generate signature (requires openssl)
SIGNATURE=$(echo -n "${TIMESTAMP}${BODY}" | openssl dgst -sha256 -hmac "$SECRET_SALT" | cut -d' '
-f2)

curl -X POST https://api.zenpays.com/api/v1/payment-intents \
  -H "Authorization: Bearer $API_KEY" \
  -H "X-Timestamp: $TIMESTAMP" \
  -H "X-Signature: $SIGNATURE" \
  -H "X-Secret-Salt: $SECRET_SALT" \
  -H "Content-Type: application/json" \
  -d "$BODY"
```

JAVASCRIPT

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

async function createPaymentIntent() {
  const apiKey = process.env.ZENPAYS_API_KEY;
  const secretSalt = process.env.ZENPAYS_SECRET_SALT;
  const timestamp = new Date().toISOString();

  const body = {
    amount: 1000,
    currency: 'USD',
    description: 'Order #123',
  };

  // Generate signature
  const data = timestamp + JSON.stringify(body);
  const signature = crypto
    .createHmac('sha256', secretSalt)
    .update(data)
    .digest('hex');

  const response = await fetch('https://api.zenpays.com/api/v1/payment-intents', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${apiKey}`,
      'X-Timestamp': timestamp,
      'X-Signature': signature,
      'X-Secret-Salt': secretSalt,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(body),
  });

  return response.json();
}
```

PYTHON

```python
import hmac
import hashlib
import json
import os
from datetime import datetime
import requests

def create_payment_intent():
    api_key = os.environ["ZENPAYS_API_KEY"]
    secret_salt = os.environ["ZENPAYS_SECRET_SALT"]
    timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")

    body = {
        "amount": 1000,
        "currency": "USD",
        "description": "Order #123",
    }

    # Generate signature
    data = timestamp + json.dumps(body, separators=(",", ":"))
    signature = hmac.new(
        secret_salt.encode(),
        data.encode(),
        hashlib.sha256
    ).hexdigest()

    response = requests.post(
        "https://api.zenpays.com/api/v1/payment-intents",
        headers={
            "Authorization": f"Bearer {api_key}",
            "X-Timestamp": timestamp,
            "X-Signature": signature,
            "X-Secret-Salt": secret_salt,
            "Content-Type": "application/json",
        },
        json=body,
    )

    return response.json()
```

Timestamp Validation

The `X-Timestamp` header must be within **±5 minutes** of the server time. Requests with timestamps outside this window are rejected to prevent replay attacks.

```
{
  "success": false,
  "error": {
    "code": "TIMESTAMP_EXPIRED",
    "message": "Request timestamp is outside the allowed window"
  }
}
```

Tip

Always use UTC timestamps and ensure your server clock is synchronized with NTP.

GET Request Authentication

For GET requests without a body, use an empty object `{}` when calculating the signature:

JAVASCRIPT

```javascript
// For GET requests
const timestamp = new Date().toISOString();
const data = timestamp + '{}';
const signature = crypto
  .createHmac('sha256', secretSalt)
  .update(data)
  .digest('hex');

const response = await fetch('https://api.zenpays.com/api/v1/payment-intents/pi_xxx', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
  },
});
```

PYTHON

```python
# For GET requests
timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")
data = timestamp + "{}"
signature = hmac.new(
    secret_salt.encode(),
    data.encode(),
    hashlib.sha256
).hexdigest()

response = requests.get(
    "https://api.zenpays.com/api/v1/payment-intents/pi_xxx",
    headers={
        "Authorization": f"Bearer {api_key}",
        "X-Timestamp": timestamp,
        "X-Signature": signature,
        "X-Secret-Salt": secret_salt,
    },
)
```

IP Whitelisting

For enhanced security, you can restrict API access to specific IP addresses:

1. Go to **Dashboard → Settings → API Security**
2. Add your server IP addresses to the whitelist
3. Enable IP restriction for your API keys

Requests from non-whitelisted IPs will receive:

```
{
  "success": false,
  "error": {
    "code": "IP_NOT_WHITELISTED",
    "message": "Request from unauthorized IP address"
  }
}
```

Authentication Errors

Error Code	HTTP Status	Description
`UNAUTHORIZED`	401	Missing or invalid API key
`INVALID_SIGNATURE`	401	Signature verification failed
`TIMESTAMP_EXPIRED`	401	Timestamp outside allowed window
`API_KEY_REVOKED`	401	API key has been revoked
`IP_NOT_WHITELISTED`	403	Request from unauthorized IP
`INSUFFICIENT_PERMISSIONS`	403	API key lacks required scope

Best Practices

1. **Use environment variables** - Never hardcode API keys or secrets
2. **Rotate keys regularly** - Create new keys and revoke old ones periodically
3. **Use minimal scopes** - Only grant necessary permissions to each key
4. **Enable IP whitelisting** - Restrict access to known server IPs
5. **Monitor API usage** - Check the dashboard for unusual activity
6. **Use HTTPS only** - Never send credentials over unencrypted connections

Next Steps

- Error Handling - Learn about error responses
- SDK Authentication - Simpler auth with the SDK
- Webhooks - Verify webhook signatures

REST API

Error Handling

The ZenPays REST API uses standard HTTP status codes and returns detailed error information in JSON format.

Error Response Format

All error responses follow this structure:

```
{
  "success": false,
  "error": {
    "code": "ERROR_CODE",
    "message": "Human-readable error message",
    "details": {
      // Additional error context (optional)
    }
  }
}
```

HTTP Status Codes

Status	Description
`200`	Success
`201`	Resource created
`400`	Bad Request - Invalid parameters
`401`	Unauthorized - Invalid or missing authentication
`403`	Forbidden - Insufficient permissions
`404`	Not Found - Resource doesn't exist
`409`	Conflict - Resource state conflict
`422`	Unprocessable Entity - Validation error
`429`	Too Many Requests - Rate limit exceeded
`500`	Internal Server Error
`502`	Bad Gateway - TSP provider error
`503`	Service Unavailable

Error Codes

Authentication Errors (401)

Code	Description
`UNAUTHORIZED`	Missing or invalid API key
`INVALID_SIGNATURE`	HMAC signature verification failed
`TIMESTAMP_EXPIRED`	Request timestamp outside allowed window
`API_KEY_REVOKED`	API key has been revoked
`API_KEY_EXPIRED`	API key has expired

Authorization Errors (403)

Code	Description
`FORBIDDEN`	Access denied
`IP_NOT_WHITELISTED`	Request from unauthorized IP
`INSUFFICIENT_PERMISSIONS`	API key lacks required scope
`MERCHANT_SUSPENDED`	Merchant account is suspended

Validation Errors (400/422)

Code	Description
`INVALID_REQUEST`	Request format is invalid
`VALIDATION_ERROR`	One or more fields failed validation
`INVALID_AMOUNT`	Amount is invalid or out of range
`INVALID_CURRENCY`	Currency code not supported
`INVALID_PAYMENT_METHOD`	Payment method not available
`MISSING_REQUIRED_FIELD`	Required field is missing

Resource Errors (404/409)

Code	Description
`RESOURCE_NOT_FOUND`	Requested resource doesn't exist
`PAYMENT_INTENT_NOT_FOUND`	Payment intent ID is invalid
`TRANSACTION_NOT_FOUND`	Transaction ID is invalid
`CUSTOMER_NOT_FOUND`	Customer ID is invalid
`REFUND_NOT_FOUND`	Refund ID is invalid
`DUPLICATE_REQUEST`	Request with same idempotency key exists
`STATE_CONFLICT`	Resource is in conflicting state

Payment Errors

Code	Description
`PAYMENT_FAILED`	Payment processing failed
`PAYMENT_DECLINED`	Payment was declined by provider
`PAYMENT_EXPIRED`	Payment intent has expired
`INSUFFICIENT_FUNDS`	Customer has insufficient funds
`CARD_DECLINED`	Card was declined
`FRAUD_DETECTED`	Suspicious activity detected
`LIMIT_EXCEEDED`	Transaction limit exceeded

Channel Limit Errors (422)

These errors occur during payment confirmation when the amount violates min/max limits configured per payment method and currency.

Code	Description
`CHANNEL_LIMIT_MIN_EXCEEDED`	Amount is below the minimum for the selected payment method
`CHANNEL_LIMIT_MAX_EXCEEDED`	Amount exceeds the maximum for the selected payment method


```
{
  "success": false,
  "error": {
    "code": "CHANNEL_LIMIT_MIN_EXCEEDED",
    "message": "The payment amount of 100 INR is below the minimum limit of 500 INR for UPI. Please
use a different payment method or increase the amount.",
    "type": "validation_error",
    "details": {
      "amount": 100,
      "currency": "INR",
      "paymentMethod": "UPI",
      "minimumAmount": 500,
      "maximumAmount": 49967
    }
  }
}
```


Refund Errors

Code	Description
`REFUND_FAILED`	Refund processing failed
`REFUND_NOT_ALLOWED`	Transaction cannot be refunded
`REFUND_AMOUNT_EXCEEDED`	Refund amount exceeds original
`REFUND_WINDOW_CLOSED`	Refund period has expired

Rate Limit Errors (429)

Code	Description
`RATE_LIMIT_EXCEEDED`	Too many requests

Response includes retry information:

```
{
  "success": false,
  "error": {
    "code": "RATE_LIMIT_EXCEEDED",
    "message": "Rate limit exceeded. Retry after 60 seconds",
    "details": {
      "limit": 1000,
      "remaining": 0,
      "resetAt": "2024-01-15T10:31:00.000Z"
    }
  }
}
```

Provider Errors (502)

Code	Description
`TSP_ERROR`	Payment provider returned an error
`TSP_TIMEOUT`	Payment provider request timed out
`TSP_UNAVAILABLE`	Payment provider is unavailable

Operational Errors

These codes describe higher-level "we couldn't even attempt this" failures — distinct from numeric transaction-state codes (bank decline, insufficient funds, etc.). They surface in `error.code` of the standard error envelope and the checkout SDK's `payment-failed` socket event.

Code	HTTP	Customer-facing message	What it means	Suggested action
`NO_TSP_AVAILABLE`	503	"We can't process this payment right now. Please try again in a few minutes, or contact support if the issue persists."	No active payment routes are configured to fulfill the request.	Retry after a short delay; escalate to support if persistent.
`UNSUPPORTED_REGION`	400	"This payment route isn't available in your region right now. Please try a different method or contact support."	TSPs exist but none support the customer's country.	Offer an alternate method or surface a region-specific support flow.
`UNSUPPORTED_CURRENCY_PAIR`	400	"We couldn't set up this payment with our provider. Try another currency or contact support."	The provider rejected the source/destination currency combination.	Suggest a supported currency.
`PROVIDER_VALIDATION_ERROR`	400	"We hit a problem setting up your payment. Please double-check your details and try again, or contact support."	Upstream provider returned a 4xx the platform couldn't normalize further.	Inspect `details.providerResponse` server-side; ask the customer to retry.
`PROVIDER_UNAVAILABLE`	502	"Our payment partner is having trouble right now. Please try again in a few minutes."	Upstream provider returned 5xx or timed out.	Retry with backoff; consider failover routing.
`PAYMENT_STUCK`	200	"This is taking longer than usual. We're still trying — feel free to wait, or come back via your dashboard."	The intent has been in `processing` past the heartbeat threshold (~90s) with no transitions. Advisory only — *not* a terminal failure.	Show a friendlier "still working" UI; keep the subscription alive.
`PAYMENT_RETRIES_EXHAUSTED`	400	"We tried several times but couldn't complete this payment. Please try a different payment method or contact support."	Retry budget exhausted; intent moved to `canceled`.	Offer a different method; investigate the last TSP failure for the underlying cause.
`TSP_PAYMENT_FAILED`	400	"Your payment couldn't be completed. Please try a different payment method or contact your bank."	TSP / webhook reported a terminal payment failure.	Surface to the customer; inspect `details.providerResponse` for upstream reason.

Code	HTTP	Customer-facing message	What it means	Suggested action
`CRYPTO_DEPOSIT_FAILED`	400	"We couldn't confirm your crypto deposit. If you've already sent funds, contact support with the transaction hash."	Crypto deposit reported failure (chain reorg, address mismatch, amount divergence).	Capture the tx hash and route to support.

These codes also appear as `code` on the WebSocket `payment-failed` event (see Payment Webhooks).

Validation Error Details

Validation errors include field-specific details:

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Request validation failed",
    "details": {
      "fields": [
        {
          "field": "amount",
          "message": "Amount must be greater than 0",
          "code": "min"
        },
        {
          "field": "currency",
          "message": "Currency must be a valid ISO 4217 code",
          "code": "invalid"
        }
      ]
    }
  }
}
```

Error Handling Examples

JAVASCRIPT

```
async function createPayment(amount, currency) {
  try {
    const response = await fetch('https://api.zenpays.com/api/v1/payment-intents', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${apiKey}`,
        'X-Timestamp': timestamp,
        'X-Signature': signature,
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ amount, currency }),
    });

    const data = await response.json();

    if (!response.ok) {
      // Handle specific error codes
      switch (data.error?.code) {
        case 'RATE_LIMIT_EXCEEDED':
          const retryAfter = data.error.details?.resetAt;
          console.log(`Rate limited. Retry after: ${retryAfter}`);
          break;
        case 'VALIDATION_ERROR':
          data.error.details?.fields?.forEach(field => {
            console.log(`${field.field}: ${field.message}`);
          });
          break;
        case 'UNAUTHORIZED':
        case 'INVALID_SIGNATURE':
          console.log('Authentication failed. Check your API key and signature.');
          break;
        default:
          console.log(`Error: ${data.error?.message}`);
      }
      return null;
    }

    return data.data;
  } catch (error) {
    console.error('Network error:', error.message);
    return null;
  }
}
```

PYTHON

```python
import requests

def create_payment(amount: int, currency: str):
    try:
        response = requests.post(
            "https://api.zenpays.com/api/v1/payment-intents",
            headers={
                "Authorization": f"Bearer {api_key}",
                "X-Timestamp": timestamp,
                "X-Signature": signature,
                "Content-Type": "application/json",
            },
            json={"amount": amount, "currency": currency},
        )

        data = response.json()

        if not response.ok:
            error = data.get("error", {})
            code = error.get("code")

            if code == "RATE_LIMIT_EXCEEDED":
                retry_after = error.get("details", {}).get("resetAt")
                print(f"Rate limited. Retry after: {retry_after}")
            elif code == "VALIDATION_ERROR":
                for field in error.get("details", {}).get("fields", []):
                    print(f"{field['field']}: {field['message']}")
            elif code in ("UNAUTHORIZED", "INVALID_SIGNATURE"):
                print("Authentication failed. Check your API key and signature.")
            else:
                print(f"Error: {error.get('message')}")
            return None

        return data.get("data")

    except requests.exceptions.RequestException as e:
        print(f"Network error: {e}")
        return None
```

Retry Strategy

For transient errors, implement exponential backoff:

JAVASCRIPT

```
async function requestWithRetry(fn, maxRetries = 3) {
  let lastError;

  for (let attempt = 0; attempt < maxRetries; attempt++) {
    try {
      return await fn();
    } catch (error) {
      lastError = error;
      const code = error.code;

      // Only retry on transient errors
      const retriableErrors = [
        'RATE_LIMIT_EXCEEDED',
        'TSP_TIMEOUT',
        'TSP_UNAVAILABLE',
      ];

      if (!retriableErrors.includes(code)) {
        throw error;
      }

      // Exponential backoff: 1s, 2s, 4s...
      const delay = Math.pow(2, attempt) * 1000;
      console.log(`Retrying in ${delay}ms (attempt ${attempt + 1}/${maxRetries})`);
      await new Promise(resolve => setTimeout(resolve, delay));
    }
  }

  throw lastError;
}
```

PYTHON

```python
import time

def request_with_retry(fn, max_retries=3):
    last_error = None

    for attempt in range(max_retries):
        try:
            return fn()
        except Exception as error:
            last_error = error
            code = getattr(error, "code", None)

            # Only retry on transient errors
            retriable_errors = [
                "RATE_LIMIT_EXCEEDED",
                "TSP_TIMEOUT",
                "TSP_UNAVAILABLE",
            ]

            if code not in retriable_errors:
                raise error

            # Exponential backoff: 1s, 2s, 4s...
            delay = (2 ** attempt)
            print(f"Retrying in {delay}s (attempt {attempt + 1}/{max_retries})")
            time.sleep(delay)

    raise last_error
```

Idempotency

To prevent duplicate operations, include an `X-Idempotency-Key` header:

```bash
curl -X POST https://api.zenpays.com/api/v1/payment-intents \
  -H "Authorization: Bearer zp_live_xxxxx" \
  -H "X-Idempotency-Key: unique-request-id-123" \
  -H "Content-Type: application/json" \
  -d '{"amount": 1000, "currency": "USD"}'
```

If the same idempotency key is used within 24 hours, the API returns the original response instead of creating a duplicate.

Next Steps

- Authentication - Learn about request signing
- Error Handling Guide - SDK error handling
- Webhooks - Handle async notifications

API objects

REST API / API OBJECTS

The ApiKey object

Represents an API key for authenticating with the ZenPays API. API keys are used to authenticate requests and can be scoped to specific permissions and environments.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the API key
`name`	`string`	Yes	Descriptive name for the API key
`keyPrefix`	`string`	Yes	Prefix of the key for identification (e.g., 'sk_live_abc...')
`environment`	`'sandbox' | 'production'`	Yes	Environment the key is valid for
`scopes`	`string[]`	Yes	Permission scopes granted to this key
`lastUsedAt`	`string`	No	ISO 8601 timestamp when the key was last used
`expiresAt`	`string`	No	ISO 8601 timestamp when the key expires (if applicable)
`createdAt`	`string`	Yes	ISO 8601 timestamp when the key was created

Example


```
{
  "id": "obj_1a2b3c",
  "name": "string",
  "keyPrefix": "string",
  "environment": "sandbox",
  "scopes": [],
  "lastUsedAt": "2026-01-15T09:30:00Z",
  "expiresAt": "2026-01-15T09:30:00Z",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The BankAccount object

Bank account for settlements

> **Note**
> 15 of 15 fields on this object have no description yet. Field documentation lives in the SDK types (`api/settlements.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	
`accountHolderName`	`string`	Yes	
`accountNumber`	`string`	Yes	
`maskedAccountNumber`	`string`	No	
`accountType`	`string`	Yes	
`ifscCode`	`string`	Yes	
`swiftCode`	`string`	No	
`bankName`	`string`	Yes	
`branchName`	`string`	No	
`branchAddress`	`string`	No	
`isPrimary`	`boolean`	Yes	
`isVerified`	`boolean`	Yes	
`status`	`string`	Yes	
`createdAt`	`string`	Yes	
`updatedAt`	`string`	No	

Example


```
{
  "id": "obj_1a2b3c",
  "accountHolderName": "string",
  "accountNumber": "string",
  "maskedAccountNumber": "string",
  "accountType": "string",
  "ifscCode": "string",
  "swiftCode": "string",
  "bankName": "string",
  "isPrimary": false,
  "isVerified": false,
  "status": "string",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The BatchPayout object

Represents a batch payout with status and progress tracking.

> **Note**
> 13 of 13 fields on this object have no description yet. Field documentation lives in the SDK types (`types/payout.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`batchId`	`string`	Yes	
`status`	`string`	Yes	
`totalPayouts`	`number`	Yes	
`successfulPayouts`	`number`	Yes	
`failedPayouts`	`number`	Yes	
`pendingPayouts`	`number`	Yes	
`totalAmount`	`number`	Yes	
`currency`	`string`	Yes	
`estimatedFees`	`number`	Yes	
`totalDebitedAmount`	`number`	Yes	
`progressPercentage`	`number`	Yes	
`createdAt`	`string`	Yes	
`updatedAt`	`string`	Yes	

Example

```
{
  "batchId": "batch_1a2b3c",
  "status": "string",
  "totalPayouts": 5000,
  "successfulPayouts": 1,
  "failedPayouts": 1,
  "pendingPayouts": 1,
  "totalAmount": 5000,
  "currency": "USD",
  "estimatedFees": 5000,
  "totalDebitedAmount": 5000,
  "progressPercentage": 1,
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The Chargeback object

Chargeback entity

> **Note**
> 22 of 22 fields on this object have no description yet. Field documentation lives in the SDK types (`types/chargeback.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`chargebackId`	`string`	Yes	
`merchantId`	`string`	Yes	
`transactionId`	`string`	Yes	
`customerId`	`string`	No	
`amount`	`number`	Yes	
`currency`	`string`	Yes	
`status`	`ChargebackStatus`	Yes	See ChargebackStatus. ChargebackStatus
`reason`	`ChargebackReason`	Yes	See ChargebackReason. ChargebackReason
`reasonCode`	`string`	No	
`reasonDescription`	`string`	No	
`source`	`ChargebackSource`	Yes	See ChargebackSource. ChargebackSource
`outcome`	`ChargebackOutcome`	No	See ChargebackOutcome. ChargebackOutcome
`priority`	`ChargebackPriority`	No	See ChargebackPriority. ChargebackPriority
`tspProvider`	`string`	No	
`externalChargebackId`	`string`	No	
`disputeDeadline`	`string`	No	
`evidenceDeadline`	`string`	No	
`evidenceSubmittedAt`	`string`	No	
`resolvedAt`	`string`	No	
`resolutionNotes`	`string`	No	
`createdAt`	`string`	Yes	
`updatedAt`	`string`	No	

Example


```
{
  "chargebackId": "chargeback_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "transactionId": "transaction_1a2b3c",
  "customerId": "customer_1a2b3c",
  "amount": 5000,
  "currency": "USD",
  "status": "initiated",
  "reason": "fraudulent",
  "source": "customer",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The CheckoutSession object

Represents a checkout session for payment processing. Checkout sessions are created to initiate a payment flow and can be used for hosted checkout pages or embedded payment experiences.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the checkout session
`merchantId`	`string`	Yes	ID of the merchant who created the session
`playerId`	`string`	Yes	ID of the player/user making the payment
`amount`	`number`	Yes	Payment amount in the source currency
`targetSettlementAsset`	`string`	Yes	Target cryptocurrency for settlement (e.g., 'USDC')
`status`	`'created' | 'active' | 'completed' | 'cancelled' | 'expired'`	Yes	Current status of the checkout session
`paymentIntentId`	`string`	No	Associated payment intent ID (if payment initiated)
`rampOrderId`	`string`	No	Associated ramp order ID (if on-ramp initiated)
`successUrl`	`string`	No	URL to redirect on successful payment
`cancelUrl`	`string`	No	URL to redirect on cancelled payment
`paymentPageUrl`	`string`	No	URL to the hosted payment page
`expiresAt`	`string`	No	ISO 8601 timestamp when the session expires
`createdAt`	`string`	Yes	ISO 8601 timestamp when the session was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the session was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "playerId": "player_1a2b3c",
  "amount": 5000,
  "targetSettlementAsset": "string",
  "status": "created",
  "paymentIntentId": "paymentintent_1a2b3c",
  "rampOrderId": "ramporder_1a2b3c",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```


Usage

```
const session = await client.checkout.create({
  playerId: 'player_123',
  amount: 100,
  targetSettlementAsset: 'USDC',
  successUrl: 'https://myapp.com/success'
});
// Redirect user to session.paymentPageUrl
```

REST API / API OBJECTS

The Commission object

Represents a commission charged on a transaction. Commissions are fees collected by ZenPays for processing payments, payouts, or refunds.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the commission record
`merchantId`	`string`	Yes	ID of the merchant who was charged
`transactionId`	`string`	Yes	ID of the transaction that generated the commission
`amount`	`number`	Yes	Commission amount in the smallest currency unit
`rate`	`number`	Yes	Commission rate applied (as a decimal, e.g., 0.029 for 2.9%)
`currency`	`string`	Yes	Three-letter ISO currency code
`type`	`'payment' | 'payout' | 'refund'`	Yes	Type of transaction the commission was charged on
`createdAt`	`string`	Yes	ISO 8601 timestamp when the commission was charged

Example

```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "transactionId": "transaction_1a2b3c",
  "amount": 5000,
  "rate": 1,
  "currency": "USD",
  "type": "payment",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The Customer object

Represents a customer in the ZenPays system. Customers are end-users who make payments through your application. Creating customer records allows you to track payment history, manage KYC status, and provide a better checkout experience.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the customer
`merchantId`	`string`	Yes	ID of the merchant this customer belongs to
`email`	`string`	Yes	Email address of the customer
`name`	`string`	Yes	Full name of the customer
`phone`	`string`	No	Phone number of the customer
`address`	`CustomerAddress`	No	Customer's address
`kycStatus`	`KycStatus`	No	KYC verification status KycStatus
`riskLevel`	`RiskLevel`	No	Risk level assigned to this customer RiskLevel
`isBlocked`	`boolean`	No	Whether the customer is blocked from making payments
`totalTransactions`	`number`	No	Total number of transactions by this customer
`totalVolume`	`number`	No	Total transaction volume for this customer
`lastTransactionAt`	`string`	No	ISO 8601 timestamp of the customer's last transaction
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the customer
`createdAt`	`string`	Yes	ISO 8601 timestamp when the customer was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the customer was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "email": "customer@example.com",
  "name": "string",
  "phone": "string",
  "address": "string",
  "kycStatus": "pending",
  "riskLevel": "low",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```


Usage

```
const customer = await client.customers.get('cus_123');
console.log(`Customer ${customer.name} has made ${customer.totalTransactions} transactions`);
```

REST API / API OBJECTS

The CustomerRiskProfile object

Risk assessment profile for a customer. Contains detailed risk scoring information used for fraud prevention and compliance decisions.

Fields

Field	Type	Required	Description
`customerId`	`string`	Yes	Customer ID this profile belongs to
`riskLevel`	`RiskLevel`	Yes	Overall risk level RiskLevel
`riskScore`	`number`	Yes	Numeric risk score (0-100, higher is riskier)
`factors`	`{ /** Name of the risk factor */ name: string /** Score contribution from this factor */ score: number /** Human-readable description of the factor */ description: string }[]`	Yes	Individual risk factors contributing to the score
`lastAssessedAt`	`string`	Yes	ISO 8601 timestamp when the risk was last assessed
`recommendations`	`string[]`	No	Recommended actions based on the risk assessment

Example

```
{
  "customerId": "customer_1a2b3c",
  "riskLevel": "low",
  "riskScore": 1,
  "factors": {},
  "lastAssessedAt": "2026-01-15T09:30:00Z",
  "recommendations": []
}
```

REST API / API OBJECTS

The IPWhitelistEntry object

Represents an IP address in the whitelist. IP whitelisting restricts API access to specific IP addresses for enhanced security.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the whitelist entry
`merchantId`	`string`	Yes	ID of the merchant this entry belongs to
`ipAddress`	`string`	Yes	Whitelisted IP address (IPv4 or IPv6)
`description`	`string`	No	Description to identify this IP address
`isActive`	`boolean`	Yes	Whether this whitelist entry is active
`createdBy`	`string`	Yes	User ID who created this entry
`createdAt`	`string`	Yes	ISO 8601 timestamp when the entry was created

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "ipAddress": "string",
  "description": "string",
  "isActive": false,
  "createdBy": "string",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The KycDocument object

Represents an uploaded KYC document. Contains information about the document and its verification status.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the document
`merchantId`	`string`	Yes	ID of the merchant who uploaded the document
`type`	`KycDocumentType`	Yes	Type of document KycDocumentType
`fileName`	`string`	Yes	Original filename
`fileSize`	`number`	Yes	File size in bytes
`mimeType`	`string`	Yes	MIME type of the file
`status`	`'pending' | 'approved' | 'rejected'`	Yes	Verification status
`rejectionReason`	`string`	No	Reason for rejection (if rejected)
`uploadedAt`	`string`	Yes	ISO 8601 timestamp when the document was uploaded
`reviewedAt`	`string`	No	ISO 8601 timestamp when the document was reviewed

Example

```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "type": "id_card",
  "fileName": "string",
  "fileSize": 1,
  "mimeType": "string",
  "status": "pending",
  "rejectionReason": "string",
  "uploadedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The LedgerBalance object

Summary of a ledger account balance. Provides totals for debits, credits, and the net balance for a specific account.

Fields

Field	Type	Required	Description
`account`	`string`	Yes	Account name in the ledger
`debit`	`number`	Yes	Total debit amount
`credit`	`number`	Yes	Total credit amount
`balance`	`number`	Yes	Net balance (credit - debit)

Example

```
{
  "account": "string",
  "debit": 1,
  "credit": 1,
  "balance": 1
}
```

REST API / API OBJECTS

The LedgerEntry object

Ledger entry

> **Note**
> 8 of 8 fields on this object have no description yet. Field documentation lives in the SDK types (`api/ledger.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	
`entryType`	`string`	Yes	
`amount`	`number`	Yes	
`currency`	`string`	Yes	
`balance`	`number`	Yes	
`description`	`string`	Yes	
`reference`	`string`	Yes	
`createdAt`	`string`	Yes	

Example

```
{
  "id": "obj_1a2b3c",
  "entryType": "string",
  "amount": 5000,
  "currency": "USD",
  "balance": 1,
  "description": "string",
  "reference": "string",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The MerchantLimits object

Transaction and payout limits for a merchant. Limits are enforced to manage risk and comply with regulations. Merchants can request limit increases as their business grows.

Fields

Field	Type	Required	Description
`dailyTransactionLimit`	`number`	Yes	Maximum daily transaction volume allowed
`dailyTransactionUsed`	`number`	Yes	Daily transaction volume already used
`monthlyTransactionLimit`	`number`	Yes	Maximum monthly transaction volume allowed
`monthlyTransactionUsed`	`number`	Yes	Monthly transaction volume already used
`singleTransactionLimit`	`number`	Yes	Maximum single transaction amount allowed
`dailyPayoutLimit`	`number`	Yes	Maximum daily payout volume allowed
`dailyPayoutUsed`	`number`	Yes	Daily payout volume already used
`monthlyPayoutLimit`	`number`	Yes	Maximum monthly payout volume allowed
`monthlyPayoutUsed`	`number`	Yes	Monthly payout volume already used
`currency`	`string`	Yes	Currency for limit amounts

Example

```
{
  "dailyTransactionLimit": 1,
  "dailyTransactionUsed": 1,
  "monthlyTransactionLimit": 1,
  "monthlyTransactionUsed": 1,
  "singleTransactionLimit": 1,
  "dailyPayoutLimit": 1,
  "dailyPayoutUsed": 1,
  "monthlyPayoutLimit": 1,
  "monthlyPayoutUsed": 1,
  "currency": "USD"
}
```

REST API / API OBJECTS

The MerchantProfile object

Represents a merchant's profile in the ZenPays system. The merchant profile contains business information and configuration settings for payment processing.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the merchant
`name`	`string`	Yes	Business or display name
`email`	`string`	Yes	Primary email address
`phone`	`string`	No	Contact phone number
`logo`	`string`	No	URL to the merchant's logo
`website`	`string`	No	Business website URL
`description`	`string`	No	Brief description of the business
`businessType`	`string`	No	Type of business (e.g., 'ecommerce', 'saas', 'marketplace')
`country`	`string`	No	Two-letter ISO country code where the business is registered
`timezone`	`string`	No	Timezone identifier (e.g., 'America/New_York')
`defaultCurrency`	`string`	No	Default currency for transactions
`kycStatus`	`KycStatus`	No	KYC verification status KycStatus
`isActive`	`boolean`	Yes	Whether the merchant account is active
`createdAt`	`string`	Yes	ISO 8601 timestamp when the merchant was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the merchant was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "name": "string",
  "email": "customer@example.com",
  "phone": "string",
  "logo": "string",
  "website": "string",
  "description": "string",
  "businessType": "string",
  "isActive": false,
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The PaymentIntent object

Represents a payment intent - the core object for initiating payments. A payment intent tracks the lifecycle of a payment from creation through completion or failure. It holds all the information needed to process the payment and is updated as the payment progresses.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the payment intent
`merchantId`	`string`	Yes	ID of the merchant receiving the payment
`merchantName`	`string`	No	Display name of the merchant
`merchantLogo`	`string`	No	URL to the merchant's logo for display on payment pages
`description`	`string`	No	Description of what the payment is for
`amount`	`number`	Yes	Payment amount in the smallest currency unit (e.g., cents for USD)
`currency`	`string`	Yes	Three-letter ISO currency code
`status`	`PaymentStatus`	Yes	Current status of the payment intent PaymentStatus
`customerName`	`string`	No	Name of the customer making the payment
`customerEmail`	`string`	No	Email of the customer making the payment
`customerPhone`	`string`	No	Phone number of the customer making the payment
`processingFee`	`number`	No	Processing fee charged for this payment
`expiresAt`	`string`	No	ISO 8601 timestamp when the payment intent expires
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the payment
`supportedPaymentMethods`	`string[]`	No	Payment methods available for this intent (e.g. 'UPI', 'UPIQR-H5', 'CARD')

Field	Type	Required	Description
`supportedCountries`	`string[]`	No	Supported countries for this intent (ISO 2-letter codes)
`channelLimits`	`Record<string, Record<string, { min: number, max: number }>>`	No	Per-currency, per-method min/max transaction limits
`selectedTsp`	`string`	No	TSP selected for processing this payment
`createdAt`	`string`	No	ISO 8601 timestamp when the payment intent was created
`updatedAt`	`string`	No	ISO 8601 timestamp when the payment intent was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "merchantName": "string",
  "merchantLogo": "string",
  "description": "string",
  "amount": 5000,
  "currency": "USD",
  "status": "pending"
}
```


Usage

```
const intent = await client.payments.create({
  amount: 5000, // $50.00
  currency: 'USD',
  description: 'Order #12345'
});
console.log(`Payment URL: ${intent.paymentUrl}`);
```

REST API / API OBJECTS

The Payout object

Represents a payout to a beneficiary. Payouts transfer funds from your ZenPays wallet to an external beneficiary account (bank, wallet, or crypto address).

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the payout
`merchantId`	`string`	Yes	ID of the merchant initiating the payout
`customerId`	`string`	No	ID of the customer receiving the payout (if registered)
`customerName`	`string`	No	Name of the customer receiving the payout
`customerEmail`	`string`	No	Email of the customer receiving the payout
`amount`	`number`	Yes	Payout amount in the smallest currency unit (e.g., cents for USD)
`currency`	`string`	Yes	Three-letter ISO currency code
`status`	`PayoutStatus`	Yes	Current status of the payout PayoutStatus
`payoutMethod`	`string`	Yes	Payout method identifier used
`beneficiaryDetails`	`BeneficiaryDetails`	Yes	Beneficiary details for the payout recipient
`tspPayoutId`	`string`	No	External payout ID from the payment provider
`failureReason`	`string`	No	Reason for failure if the payout failed
`processedAt`	`string`	No	ISO 8601 timestamp when the payout was processed
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the payout
`createdAt`	`string`	Yes	ISO 8601 timestamp when the payout was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the payout was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "customerId": "customer_1a2b3c",
  "customerName": "string",
  "customerEmail": "customer@example.com",
  "amount": 5000,
  "currency": "USD",
  "status": "pending",
  "payoutMethod": "string",
  "beneficiaryDetails": "string",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```


Usage

```
const payout = await client.payouts.get('pay_123');
if (payout.status === 'completed') {
  console.log(`Payout of ${payout.amount} ${payout.currency} completed`);
}
```

REST API / API OBJECTS

The PayoutIntent object

A payout intent represents a two-step payout. Step 1: Create the intent with basic details. Step 2: Submit the required fields and confirm.

> **Note**
> 23 of 23 fields on this object have no description yet. Field documentation lives in the SDK types (`types/payout.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`intentId`	`string`	Yes	
`status`	`PayoutIntentStatus`	Yes	See PayoutIntentStatus. PayoutIntentStatus
`amount`	`number`	Yes	
`currency`	`string`	Yes	
`country`	`string`	No	
`beneficiaryName`	`string`	Yes	
`beneficiaryEmail`	`string`	No	
`payoutType`	`string`	No	
`isCrypto`	`boolean`	No	
`purpose`	`string`	No	
`customerId`	`string`	No	
`requiredFields`	`PayoutFieldDefinition[]`	No	
`optionalFields`	`PayoutFieldDefinition[]`	No	
`oneOfGroups`	`string[][]`	No	
`payoutId`	`string`	No	
`processingFee`	`number`	No	
`totalDebitedAmount`	`number`	No	
`failureReason`	`string`	No	
`expiresAt`	`string`	Yes	
`confirmedAt`	`string`	No	
`completedAt`	`string`	No	
`createdAt`	`string`	Yes	
`updatedAt`	`string`	No	

Example


```
{
  "intentId": "intent_1a2b3c",
  "status": "requires_confirmation",
  "amount": 5000,
  "currency": "USD",
  "country": "string",
  "beneficiaryName": "string",
  "beneficiaryEmail": "customer@example.com",
  "payoutType": "string",
  "expiresAt": "2026-01-15T09:30:00Z",
  "createdAt": "2026-01-15T09:30:00Z"
}
```


Usage

```
// Step 1
const intent = await zenpays.payouts.createPayoutIntent({
  amount: 10000,
  currency: 'INR',
  country: 'IN',
  beneficiaryName: 'John Doe',
})

// Step 2 — fill the required fields from intent.requiredFields
const result = await zenpays.payouts.confirmPayoutIntent(intent.intentId, {
  beneficiaryAccount: '1234567890',
  beneficiaryIfsc: 'HDFC0001234',
})
```

REST API / API OBJECTS

The PayoutMethod object

Describes an available payout method and its constraints. Use this information to display available payout options to users and validate payout requests.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the payout method
`type`	`'bank_transfer' | 'upi' | 'wallet' | 'crypto'`	Yes	Type of payout method
`name`	`string`	Yes	Display name for the payout method
`currencies`	`string[]`	Yes	Currencies supported by this payout method
`countries`	`string[]`	Yes	Countries where this payout method is available
`minAmount`	`number`	No	Minimum payout amount (in smallest currency unit)
`maxAmount`	`number`	No	Maximum payout amount (in smallest currency unit)
`processingTime`	`string`	No	Estimated processing time (e.g., '1-2 business days')
`fees`	`{ /** Fixed fee amount per payout */ fixed?: number /** Percentage fee (e.g., 0.5 for 0.5%) */ percentage?: number }`	No	Fee structure for this payout method

Example

```
{
  "id": "obj_1a2b3c",
  "type": "bank_transfer",
  "name": "string",
  "currencies": [],
  "countries": [],
  "minAmount": 5000,
  "maxAmount": 5000,
  "processingTime": "string"
}
```

REST API / API OBJECTS

The RampOrder object

Represents a fiat-to-crypto on-ramp order. Ramp orders track the conversion of fiat currency to cryptocurrency through a third-party on-ramp provider.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the ramp order
`checkoutSessionId`	`string`	Yes	Associated checkout session ID
`paymentIntentId`	`string`	Yes	Associated payment intent ID
`provider`	`string`	Yes	On-ramp provider handling the order
`fiatAmount`	`number`	Yes	Fiat amount being converted
`fiatCurrency`	`string`	Yes	Fiat currency code
`cryptoAmount`	`number`	Yes	Cryptocurrency amount to be received
`cryptoCurrency`	`string`	Yes	Cryptocurrency code
`kycStatus`	`KycStatus`	Yes	KYC verification status for this order KycStatus
`paymentMethod`	`'card' | 'bank' | 'local'`	Yes	Payment method used
`status`	`'pending' | 'pending_provider' | 'pending_funds' | 'confirmed' | 'matched' | 'cleared' | 'failed'`	Yes	Current status of the ramp order
`creditSplit`	`{ /** Amount immediately available */ available: number /** Amount pending confirmation */ pending: number }`	No	Split of credited amounts between available and pending
`reserveAmount`	`number`	No	Amount reserved for this order
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the order
`createdAt`	`string`	Yes	ISO 8601 timestamp when the order was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the order was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "checkoutSessionId": "checkoutsession_1a2b3c",
  "paymentIntentId": "paymentintent_1a2b3c",
  "provider": "string",
  "fiatAmount": 5000,
  "fiatCurrency": "USD",
  "cryptoAmount": 5000,
  "cryptoCurrency": "USD",
  "kycStatus": "pending",
  "paymentMethod": "card",
  "status": "pending",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The Refund object

Represents a refund for a previously completed transaction. Refunds allow merchants to return funds to customers for completed payments. A refund can be for the full amount or a partial amount of the original transaction.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the refund
`merchantId`	`string`	Yes	ID of the merchant that issued the refund
`transactionId`	`string`	Yes	ID of the original transaction being refunded
`customerId`	`string`	No	ID of the customer receiving the refund (if registered)
`customerName`	`string`	No	Name of the customer receiving the refund
`customerEmail`	`string`	No	Email of the customer receiving the refund
`amount`	`number`	Yes	Refund amount in the smallest currency unit (e.g., cents for USD)
`currency`	`string`	Yes	Three-letter ISO currency code
`status`	`RefundStatus`	Yes	Current status of the refund RefundStatus
`reason`	`string`	No	Reason provided for the refund
`failureReason`	`string`	No	Reason for failure if the refund failed
`tspRefundId`	`string`	No	External refund ID from the payment provider
`processedAt`	`string`	No	ISO 8601 timestamp when the refund was processed
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the refund
`createdAt`	`string`	Yes	ISO 8601 timestamp when the refund was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the refund was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "transactionId": "transaction_1a2b3c",
  "customerId": "customer_1a2b3c",
  "customerName": "string",
  "customerEmail": "customer@example.com",
  "amount": 5000,
  "currency": "USD",
  "status": "pending_approval",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```


Usage

```
const refund = await client.refunds.get('ref_123');
if (refund.status === 'completed') {
  console.log(`Refund of ${refund.amount} ${refund.currency} completed`);
}
```

REST API / API OBJECTS

The Report object

Represents a generated or in-progress report. Reports are generated asynchronously. Poll the status until it reaches 'completed' to get the download URL.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the report
`merchantId`	`string`	Yes	ID of the merchant who requested the report
`type`	`ReportType`	Yes	Type of report ReportType
`format`	`ReportFormat`	Yes	Output format of the report ReportFormat
`status`	`'pending' | 'processing' | 'completed' | 'failed'`	Yes	Current generation status
`downloadUrl`	`string`	No	URL to download the report (when completed)
`expiresAt`	`string`	No	ISO 8601 timestamp when the download URL expires
`filters`	`Record<string, unknown>`	No	Filters that were applied to generate the report
`createdAt`	`string`	Yes	ISO 8601 timestamp when the report was requested
`completedAt`	`string`	No	ISO 8601 timestamp when the report generation completed

Example

```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "type": "transactions",
  "format": "csv",
  "status": "pending",
  "downloadUrl": "string",
  "expiresAt": "2026-01-15T09:30:00Z",
  "filters": {},
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The ReportSchedule object

Configuration for a scheduled recurring report. Scheduled reports are automatically generated and delivered to specified recipients on a recurring basis.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the schedule
`merchantId`	`string`	Yes	ID of the merchant who owns the schedule
`type`	`ReportType`	Yes	Type of report to generate ReportType
`format`	`ReportFormat`	Yes	Output format for the report ReportFormat
`frequency`	`'daily' | 'weekly' | 'monthly'`	Yes	How often to generate the report
`dayOfWeek`	`number`	No	Day of week for weekly reports (0=Sunday, 6=Saturday)
`dayOfMonth`	`number`	No	Day of month for monthly reports (1-31)
`time`	`string`	Yes	Time to generate the report (HH:MM in merchant's timezone)
`recipients`	`string[]`	Yes	Email addresses to receive the report
`isActive`	`boolean`	Yes	Whether the schedule is currently active
`lastRunAt`	`string`	No	ISO 8601 timestamp of the last report generation
`nextRunAt`	`string`	No	ISO 8601 timestamp of the next scheduled generation
`createdAt`	`string`	Yes	ISO 8601 timestamp when the schedule was created

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "type": "transactions",
  "format": "csv",
  "frequency": "daily",
  "dayOfWeek": 1,
  "dayOfMonth": 1,
  "time": "string",
  "recipients": [],
  "isActive": false,
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The SecurityEvent object

Represents a security-related event in the audit log. Security events track authentication attempts, configuration changes, and suspicious activities for compliance and investigation purposes.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the event
`merchantId`	`string`	Yes	ID of the merchant this event relates to
`type`	`SecurityEventType`	Yes	Type of security event SecurityEventType
`severity`	`'info' | 'warning' | 'critical'`	Yes	Severity level of the event
`description`	`string`	Yes	Human-readable description of the event
`ipAddress`	`string`	No	IP address associated with the event
`userAgent`	`string`	No	User agent string from the request
`userId`	`string`	No	ID of the user who triggered the event
`metadata`	`Record<string, unknown>`	No	Additional event-specific data
`createdAt`	`string`	Yes	ISO 8601 timestamp when the event occurred

Example

```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "type": "login_success",
  "severity": "info",
  "description": "string",
  "ipAddress": "string",
  "userAgent": "string",
  "userId": "user_1a2b3c",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The Settlement object

Represents a settlement of funds to a merchant's bank account. Settlements aggregate transaction proceeds and transfer them to the merchant's designated bank account.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the settlement
`merchantId`	`string`	Yes	ID of the merchant receiving the settlement
`amount`	`number`	Yes	Settlement amount in the smallest currency unit
`currency`	`string`	Yes	Three-letter ISO currency code
`status`	`SettlementStatus`	Yes	Current status of the settlement SettlementStatus
`bankAccountId`	`string`	No	ID of the bank account receiving the settlement
`transactionCount`	`number`	Yes	Number of transactions included in this settlement
`periodStart`	`string`	Yes	Start of the settlement period (ISO 8601)
`periodEnd`	`string`	Yes	End of the settlement period (ISO 8601)
`processedAt`	`string`	No	ISO 8601 timestamp when the settlement was processed
`failureReason`	`string`	No	Reason for failure if the settlement failed
`createdAt`	`string`	Yes	ISO 8601 timestamp when the settlement was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the settlement was last updated

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "amount": 5000,
  "currency": "USD",
  "status": "pending",
  "bankAccountId": "bankaccount_1a2b3c",
  "transactionCount": 1,
  "periodStart": "string",
  "periodEnd": "string",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The StoredPaymentMethod object

The `StoredPaymentMethod` object.

> **Note**
> 11 of 11 fields on this object have no description yet. Field documentation lives in the SDK types (`types/subscription.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`storedPaymentMethodId`	`string`	Yes	
`merchantId`	`string`	Yes	
`customerId`	`string`	Yes	
`customerEmail`	`string`	Yes	
`tspProvider`	`string`	Yes	
`type`	`string`	Yes	
`brand`	`string`	No	
`expiryMonth`	`number`	No	
`expiryYear`	`number`	No	
`isActive`	`boolean`	Yes	
`metadata`	`Record<string, unknown>`	No	

Example

```
{
  "storedPaymentMethodId": "storedpaymentmethod_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "customerId": "customer_1a2b3c",
  "customerEmail": "customer@example.com",
  "tspProvider": "string",
  "type": "string",
  "brand": "string",
  "expiryMonth": 1,
  "isActive": false
}
```

REST API / API OBJECTS

The Subscription object

The `Subscription` object.

> **Note**
> 16 of 16 fields on this object have no description yet. Field documentation lives in the SDK types (`types/subscription.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`subscriptionId`	`string`	Yes	
`merchantId`	`string`	Yes	
`customerId`	`string`	Yes	
`customerEmail`	`string`	Yes	
`planId`	`string`	Yes	
`status`	`SubscriptionStatus`	Yes	See SubscriptionStatus. SubscriptionStatus
`collectionMethod`	`CollectionMethod`	Yes	See CollectionMethod. CollectionMethod
`storedPaymentMethodId`	`string`	No	
`tspProvider`	`string`	No	
`currentPeriodStart`	`string`	Yes	
`currentPeriodEnd`	`string`	Yes	
`nextBillingAt`	`string`	Yes	
`trialEnd`	`string`	No	
`cancelAtPeriodEnd`	`boolean`	Yes	
`cancelledAt`	`string`	No	
`pausedAt`	`string`	No	

Example


```
{
  "subscriptionId": "subscription_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "customerId": "customer_1a2b3c",
  "customerEmail": "customer@example.com",
  "planId": "plan_1a2b3c",
  "status": "trialing",
  "collectionMethod": "auto_charge",
  "storedPaymentMethodId": "storedpaymentmethod_1a2b3c",
  "currentPeriodStart": "string",
  "currentPeriodEnd": "string",
  "nextBillingAt": "2026-01-15T09:30:00Z",
  "cancelAtPeriodEnd": false
}
```

REST API / API OBJECTS

The SubscriptionPlan object

The `SubscriptionPlan` object.

> **Note**
> 11 of 11 fields on this object have no description yet. Field documentation lives in the SDK types (`types/subscription.ts`) so it stays in one place — this page is generated from it.

Fields

Field	Type	Required	Description
`planId`	`string`	Yes	
`merchantId`	`string`	Yes	
`name`	`string`	Yes	
`description`	`string`	No	
`amount`	`number`	Yes	
`currency`	`string`	Yes	
`interval`	`BillingInterval`	Yes	See BillingInterval. BillingInterval
`intervalCount`	`number`	Yes	
`trialDays`	`number`	Yes	
`status`	`SubscriptionPlanStatus`	Yes	See SubscriptionPlanStatus. SubscriptionPlanStatus
`metadata`	`Record<string, unknown>`	No	

Example

```
{
  "planId": "plan_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "name": "string",
  "description": "string",
  "amount": 5000,
  "currency": "USD",
  "interval": "day",
  "intervalCount": 1,
  "trialDays": 1,
  "status": "active"
}
```

REST API / API OBJECTS

The Transaction object

Represents a completed or in-progress transaction. Transactions are created when a payment intent is confirmed and track the actual movement of funds.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the transaction
`intentId`	`string`	Yes	Associated payment intent ID
`merchantId`	`string`	Yes	Merchant ID that received the payment
`customerId`	`string`	No	Customer ID if the customer was registered
`customerName`	`string`	No	Name of the customer
`customerEmail`	`string`	No	Email of the customer
`amount`	`number`	Yes	Transaction amount in the smallest currency unit
`currency`	`string`	Yes	Three-letter ISO currency code
`status`	`PaymentStatus`	Yes	Current status of the transaction PaymentStatus
`paymentMethod`	`PaymentMethodType`	No	Payment method used for this transaction PaymentMethodType
`tspProvider`	`string`	No	Third-party service provider that processed the payment
`externalTransactionId`	`string`	No	External transaction ID from the payment provider
`processingFee`	`number`	No	Processing fee charged for this transaction
`netAmount`	`number`	No	Net amount after fees
`failureReason`	`string`	No	Reason for failure if the transaction failed
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the transaction
`createdAt`	`string`	Yes	ISO 8601 timestamp when the transaction was created

Field	Type	Required	Description
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the transaction was last updated

Example

```
{
  "id": "obj_1a2b3c",
  "intentId": "intent_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "customerId": "customer_1a2b3c",
  "customerName": "string",
  "customerEmail": "customer@example.com",
  "amount": 5000,
  "currency": "USD",
  "status": "pending",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The WalletAdjustment object

Represents a manual adjustment to a wallet balance. Adjustments are typically made by administrators to correct errors or apply promotional credits.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the adjustment
`merchantId`	`string`	Yes	ID of the merchant whose wallet was adjusted
`type`	`'credit' | 'debit'`	Yes	Type of adjustment
`amount`	`number`	Yes	Adjustment amount in the smallest currency unit
`currency`	`string`	Yes	Three-letter ISO currency code
`reason`	`string`	Yes	Reason for the adjustment
`approvedBy`	`string`	No	ID of the administrator who approved the adjustment
`createdAt`	`string`	Yes	ISO 8601 timestamp when the adjustment was made

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "type": "credit",
  "amount": 5000,
  "currency": "USD",
  "reason": "string",
  "approvedBy": "string",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The WalletBalance object

Represents the balance of a merchant's wallet for a specific currency. The wallet balance is divided into different categories to help understand available and committed funds.

Fields

Field	Type	Required	Description
`currency`	`string`	Yes	Three-letter ISO currency code
`available`	`number`	Yes	Amount available for withdrawal or payout
`pending`	`number`	Yes	Amount pending from recent transactions (not yet available)
`reserved`	`number`	Yes	Amount reserved for pending payouts or refunds
`total`	`number`	Yes	Total balance (available + pending + reserved)

Example

```
{
  "currency": "USD",
  "available": 1,
  "pending": 1,
  "reserved": 1,
  "total": 5000
}
```

Usage

```
const balances = await client.wallet.getBalances();
const usdBalance = balances.find(b => b.currency === 'USD');
console.log(`Available: ${usdBalance.available}, Pending: ${usdBalance.pending}`);
```

REST API / API OBJECTS

The WalletTransaction object

Represents a transaction in the merchant's wallet. Wallet transactions track all movements of funds including credits from payments, debits from payouts, and adjustments.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the wallet transaction
`merchantId`	`string`	Yes	ID of the merchant whose wallet was affected
`type`	`'credit' | 'debit' | 'adjustment' | 'settlement' | 'fee' | 'refund'`	Yes	Type of wallet transaction
`amount`	`number`	Yes	Transaction amount in the smallest currency unit
`currency`	`string`	Yes	Three-letter ISO currency code
`balance`	`number`	Yes	Wallet balance after this transaction
`description`	`string`	Yes	Human-readable description of the transaction
`referenceId`	`string`	No	ID of the related resource (payment, refund, etc.)
`referenceType`	`'payment' | 'refund' | 'payout' | 'settlement' | 'adjustment'`	No	Type of the related resource
`metadata`	`Record<string, unknown>`	No	Custom key-value data attached to the transaction
`createdAt`	`string`	Yes	ISO 8601 timestamp when the transaction occurred

Example


```
{
  "id": "obj_1a2b3c",
  "merchantId": "merchant_1a2b3c",
  "type": "credit",
  "amount": 5000,
  "currency": "USD",
  "balance": 1,
  "description": "string",
  "referenceId": "reference_1a2b3c",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The WebhookConfig object

Configuration for a webhook endpoint. Webhooks allow you to receive real-time notifications about events in your ZenPays account.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the webhook
`url`	`string`	Yes	URL to receive webhook notifications
`events`	`string[]`	Yes	Event types this webhook subscribes to
`secret`	`string`	No	Secret for verifying webhook signatures
`isActive`	`boolean`	Yes	Whether the webhook is currently active
`failureCount`	`number`	No	Number of consecutive delivery failures
`lastDeliveryAt`	`string`	No	ISO 8601 timestamp of the last successful delivery
`createdAt`	`string`	Yes	ISO 8601 timestamp when the webhook was created
`updatedAt`	`string`	Yes	ISO 8601 timestamp when the webhook was last updated

Example

```
{
  "id": "obj_1a2b3c",
  "url": "string",
  "events": [],
  "secret": "string",
  "isActive": false,
  "failureCount": 1,
  "lastDeliveryAt": "2026-01-15T09:30:00Z",
  "createdAt": "2026-01-15T09:30:00Z",
  "updatedAt": "2026-01-15T09:30:00Z"
}
```

REST API / API OBJECTS

The WebhookDeliveryLog object

Record of a webhook delivery attempt. Use delivery logs to debug webhook issues and verify that notifications are being received correctly.

Fields

Field	Type	Required	Description
`id`	`string`	Yes	Unique identifier for the delivery attempt
`webhookId`	`string`	Yes	ID of the webhook that was triggered
`eventType`	`string`	Yes	Type of event that triggered the webhook
`payload`	`Record<string, unknown>`	Yes	Payload that was sent to the webhook URL
`statusCode`	`number`	No	HTTP status code received from the webhook URL
`response`	`string`	No	Response body received from the webhook URL
`attempts`	`number`	Yes	Number of delivery attempts made
`deliveredAt`	`string`	No	ISO 8601 timestamp when the webhook was successfully delivered
`createdAt`	`string`	Yes	ISO 8601 timestamp when the delivery was initiated

Example

```
{
  "id": "obj_1a2b3c",
  "webhookId": "webhook_1a2b3c",
  "eventType": "string",
  "payload": {},
  "statusCode": 1,
  "response": "string",
  "attempts": 1,
  "deliveredAt": "2026-01-15T09:30:00Z",
  "createdAt": "2026-01-15T09:30:00Z"
}
```

Statuses and enums

REST API / STATUSES AND ENUMS

Chargeback status and reasons

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

ChargebackStatus

Chargeback status values

Value	Meaning
`initiated`	—
`pending`	—
`evidence_submitted`	—
`under_review`	—
`won`	—
`lost`	—
`cancelled`	—

ChargebackReason

Chargeback reason categories

Value	Meaning
`fraudulent`	—
`duplicate`	—
`product_not_received`	—
`product_unacceptable`	—
`subscription_cancelled`	—
`credit_not_processed`	—
`general`	—
`other`	—

ChargebackSource

Chargeback source

Value	Meaning
`customer`	—
`issuer`	—
`network`	—
`merchant`	—

ChargebackOutcome

Chargeback outcome

Value	Meaning
`won`	—
`lost`	—
`cancelled`	—

ChargebackPriority

Chargeback priority

Value	Meaning
`low`	—
`medium`	—
`high`	—
`urgent`	—

REST API / STATUSES AND ENUMS

Compliance and risk

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

KycStatus

Know Your Customer (KYC) verification status.

Value	Meaning
`pending`	KYC verification is awaiting review
`pass`	KYC verification was successful
`fail`	KYC verification failed

KycDocumentType

Types of documents that can be submitted for KYC verification.

Value	Meaning
`id_card`	Government-issued ID card
`passport`	International passport
`drivers_license`	Driver's license
`business_registration`	Business registration certificate
`tax_certificate`	Tax registration certificate
`bank_statement`	Recent bank statement
`utility_bill`	Utility bill for address verification
`proof_of_address`	Other proof of address document

RiskLevel

Risk assessment levels used for fraud detection and compliance.

Value	Meaning
`low`	Minimal risk, standard processing
`medium`	Moderate risk, may require additional verification
`high`	High risk, requires careful review
`critical`	Critical risk, may be blocked automatically

REST API / STATUSES AND ENUMS

Payment status and methods

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

PaymentStatus

Possible statuses for a payment throughout its lifecycle.

Value	Meaning
`pending`	Payment has been created but not yet processed
`processing`	Payment is currently being processed
`succeeded`	Payment completed successfully
`failed`	Payment failed to process
`cancelled`	Payment was cancelled before completion
`expired`	Payment expired before being completed
`confirmed`	Payment has been confirmed by the network/provider

PaymentMethodType

Available payment method types supported by the SDK.

Value	Meaning
`crypto`	Cryptocurrency payments
`credit_card`	Credit card payments
`debit_card`	Debit card payments
`net_banking`	Internet banking/direct bank transfers
`upi`	Unified Payments Interface (India)
`upiqr-h5`	UPI QR code payment (India)
`wallet`	Digital wallet payments
`bank_transfer`	Traditional bank transfers
`mobile_money`	Mobile money services (Africa)

REST API / STATUSES AND ENUMS

Payout status

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

PayoutStatus

Possible statuses for a payout throughout its lifecycle.

Value	Meaning
`pending`	Payout has been created but not yet processed
`processing`	Payout is currently being processed
`completed`	Payout has been successfully completed
`failed`	Payout failed to process
`cancelled`	Payout was cancelled before completion

PayoutIntentStatus

Status of a payout intent.

Value	Meaning
`requires_confirmation`	—
`processing`	—
`succeeded`	—
`failed`	—
`cancelled`	—
`expired`	—

REST API / STATUSES AND ENUMS

Refund status

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

RefundStatus

Possible statuses for a refund throughout its lifecycle.

Value	Meaning
`pending_approval`	Refund awaiting admin approval
`approved`	Approved, being sent to payment provider
`processing`	Refund is currently being processed by TSP
`completed`	Refund has been successfully completed
`failed`	Refund failed to process
`rejected`	Refund was rejected by admin
`cancelled`	Refund was cancelled before completion
`requires_beneficiary`	Refund needs beneficiary details before processing

REST API / STATUSES AND ENUMS

Report types and formats

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

ReportType

Available report types that can be generated.

Value	Meaning
`transactions`	Detailed transaction listing
`settlements`	Settlement history and details
`refunds`	Refund transaction report
`payouts`	Payout history report
`revenue`	Revenue summary report
`customers`	Customer activity report
`reconciliation`	Financial reconciliation report

ReportFormat

Supported export formats for reports.

Value	Meaning
`csv`	Comma-separated values (for spreadsheets)
`xlsx`	Microsoft Excel format
`pdf`	PDF document
`json`	JSON data format

REST API / STATUSES AND ENUMS

Security events

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

SecurityEventType

All possible security event types.

Value	Meaning
`login_success`	Successful authentication
`login_failed`	Failed authentication attempt
`logout`	User logged out
`password_changed`	Password was changed
`two_factor_enabled`	2FA was enabled
`two_factor_disabled`	2FA was disabled
`api_key_created`	New API key was created
`api_key_revoked`	API key was revoked
`ip_whitelist_added`	IP address added to whitelist
`ip_whitelist_removed`	IP address removed from whitelist
`suspicious_activity`	Suspicious activity detected
`rate_limit_exceeded`	Rate limit was exceeded
`unauthorized_access`	Unauthorized access attempt

REST API / STATUSES AND ENUMS

Settlement status

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

SettlementStatus

Possible statuses for a settlement throughout its lifecycle.

Value	Meaning
`pending`	Settlement has been scheduled but not yet initiated
`processing`	Settlement is currently being processed
`completed`	Settlement has been successfully completed
`failed`	Settlement failed to process

REST API / STATUSES AND ENUMS

Subscription and billing

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

SubscriptionStatus

Value	Meaning
`trialing`	—
`active`	—
`past_due`	—
`paused`	—
`cancelled`	—

SubscriptionPlanStatus

Value	Meaning
`active`	—
`archived`	—

BillingInterval

Subscription types

Value	Meaning
`day`	—
`week`	—
`month`	—
`year`	—

CollectionMethod

Value	Meaning
`auto_charge`	—
`send_invoice`	—

REST API / STATUSES AND ENUMS

Webhook events

These values are generated from the SDK's own types, so they always match what the shipped client accepts.

WebhookEventType

All available webhook event types. Subscribe to these events to receive real-time notifications about activity in your ZenPays account.

Value	Meaning
`payment.intent.created`	—
`payment.intent.processing`	—
`payment.intent.succeeded`	—
`payment.intent.failed`	—
`payment.intent.expired`	—
`refund.created`	—
`refund.processing`	—
`refund.completed`	—
`refund.failed`	—
`payout.created`	—
`payout.processing`	—
`payout.completed`	—
`payout.failed`	—
`settlement.initiated`	—
`settlement.completed`	—
`customer.created`	—
`customer.updated`	—

REST API

Errors

Every error the SDK throws extends `ZenPaysError`, so a single `catch` can narrow with `instanceof`.

SDK error classes

Generated from `errors/index.ts` and the status mapping in `utils/http.ts`.

Class	code	HTTP	Description
`AuthenticationError`	`AUTHENTICATION_ERROR`	401	Thrown when API key authentication fails. Check that your API key is valid and not expired.
`AuthorizationError`	`AUTHORIZATION_ERROR`	403	Thrown when the authenticated user lacks permission for the requested action.
`NotFoundError`	`NOT_FOUND`	404	Thrown when the requested resource does not exist.
`ValidationError`	`VALIDATION_ERROR`	400	Thrown when request parameters fail validation.
`RateLimitError`	`RATE_LIMIT_EXCEEDED`	429	Thrown when API rate limits are exceeded. Wait for the specified duration before retrying.
`NetworkError`	`NETWORK_ERROR`	—	Thrown when a network error occurs (timeout, connection refused, etc.).
`PaymentError`	`PAYMENT_ERROR`	402	Thrown when a payment operation fails.
`ChannelLimitError`	–	422	Thrown when a payment amount violates configured channel limits (min/max per payment method per currency). The `limits` property contains the configured boundaries and the attempted amount.
`ConfigurationError`	`CONFIGURATION_ERROR`	—	Thrown when SDK configuration is invalid.

```
import { RateLimitError, ValidationError, ZenPaysError } from '@zenxdigitalholdings/zenpays'

try {
  await zenpays.payments.createPaymentIntent({ amount: 5000, currency: 'USD' })
}
catch (error) {
  if (error instanceof ValidationError)
    console.error('Bad request:', error.fields)
  else if (error instanceof RateLimitError)
    await sleep((error.retryAfter ?? 1) * 1000)
  else if (error instanceof ZenPaysError)
    console.error(error.code, error.status)
  else throw error
}
```

Note

`PaymentError`, `ChannelLimitError` and `ConfigurationError` are exported for `instanceof` narrowing but are not currently thrown by the client itself — `ConfigurationError` only on a missing API key. They exist so API-supplied codes can be matched without string comparison.

Wire-level error codes

The classes above are the SDK's view. The API also returns its own `code` strings in the response envelope, which overlap the list above on only `VALIDATION_ERROR`. Those are documented separately under REST API errors — the two lists are deliberately not merged, because they are different vocabularies.

REST API

Supported currencies

Every `amount` crossing the API is an **integer in the currency's smallest unit**. The divisor is not always 100 — `BTC` has 8 decimal places and `ETH` has 18 — so hard-coding `* 100` is wrong for several supported currencies.

> **Amounts are in minor units**
>
> `{ "amount": 5000, "currency": "USD" }` is **$50.00**. `{ "amount": 5000, "currency": "BTC" }` is **0.00005 BTC**, not 5000 BTC.
>
> Use `toMinorUnits(amount, currency)` and `fromMinorUnits(amount, currency)` from the SDK rather than multiplying by hand.

Currencies

Generated from the SDK's own currency tables, so this page cannot disagree with what the client accepts.

Currency	Symbol	Decimals	Minor units per unit	In `Currency` type
`AED`	د.إ	2	100	Yes
`AUD`	A$	2	100	Yes
`BTC`	฿	8	100,000,000	Yes
`CAD`	C$	2	100	Yes
`CNY`	¥	2	100	**No**
`ETH`	Ξ	18	1,000,000,000,000,000,000	Yes
`EUR`	€	2	100	Yes
`GBP`	£	2	100	Yes
`GHS`	GH₵	2	100	Yes
`INR`	₹	2	100	Yes
`JPY`	¥	0	1	**No**
`KES`	KSh	2	100	Yes
`KRW`	—	0	1	**No**
`NGN`	₦	2	100	Yes
`SGD`	S$	2	100	Yes
`SOL`	SOL	9	1,000,000,000	Yes
`USD`	$	2	100	Yes
`USDC`	USDC	6	1,000,000	Yes
`USDT`	USDT	6	1,000,000	Yes
`ZAR`	R	2	100	Yes

A currency absent from the decimals table falls back to **2**.

> **Note**
> A **No** in the last column means the currency appears in the SDK's symbol or decimals tables but not in its `Currency` union — so it formats correctly but is not an accepted type-level value. Those rows are a real inconsistency in the SDK worth reconciling.

Payments

REST API / PAYMENTS

Create Payment Intent

Create a new payment intent to initiate a payment flow.

`POST` **`/api/v1/payment-intents`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Idempotency-Key`	No	Unique key to prevent duplicates

Body Parameters

Parameter	Type	Required	Description
`amount`	`integer`	Yes	Amount in smallest currency unit (e.g., cents)
`currency`	`string`	Yes	ISO 4217 currency code (e.g., `USD`, `INR`)
`paymentMethod`	`string`	No	Preferred payment method
`customerEmail`	`string`	No	Customer email address
`customerPhone`	`string`	No	Customer phone number
`customerFirstName`	`string`	No	Customer first name
`customerLastName`	`string`	No	Customer last name
`customerCountry`	`string`	No	ISO country code (e.g., `IN`, `US`)
`description`	`string`	No	Payment description
`successUrl`	`string`	No	Redirect URL on success
`cancelUrl`	`string`	No	Redirect URL on cancel

```
{
  "amount": 100,
  "currency": "INR",
  "paymentMethod": "upi",
  "customerEmail": "test@example.com",
  "customerPhone": "+911234567890",
  "customerFirstName": "Test",
  "customerLastName": "Customer",
  "customerCountry": "IN",
  "description": "Test payment"
}
```

Payment Methods

Value	Description
`credit_card`	Credit card
`upi`	UPI (India)
`net_banking`	Net banking
`crypto`	Cryptocurrency

Note

- `credit_card` is currently supported for TWD (Taiwan Dollar) payments. The `customerPhone` field must be a 10-digit number starting with `09` (e.g., `0912345678`).
- `upi` is used for INR (Indian Rupee) payments.
- For crypto payments (USDT, USDC, BTC, ETH, SOL, BNB), the `paymentMethod` field is optional — the system auto-detects the payment method from the crypto currency.

Try it out

Test payment intent creation interactively using the API Simulator.


```
{
  "amount": 100,
  "currency": "INR",
  "paymentMethod": "upi",
  "customerEmail": "test@example.com",
  "customerPhone": "+911234567890",
  "customerFirstName": "Test",
  "customerLastName": "Customer",
  "customerCountry": "IN",
  "description": "Test payment"
}
```

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "intentId": "pi_1771910100598_prckpcg01",
    "merchantId": "mer_3630198df5cb479b",
    "customerId": null,
    "amount": 100,
    "currency": "INR",
    "status": "requires_payment_method",
    "description": "Test payment",
    "expiresAt": "2026-02-24T05:20:00.598Z",
    "estimatedCompletionTime": 30000,
    "processingFee": 0,
    "createdAt": "2026-02-24T05:15:00.599Z",
    "metadata": {},
    "paymentPageUrl": "https://checkout.zenpays.com/pay/pi_1771910100598_prckpcg01?
token=pi_1771910100598_secret_csit7h",
    "routingDecision": {
      "reason": "Initial routing - will be optimized with customer intelligence",
      "paymentMethods": ["UPI", "PIX", "FPS"],
      "supportedCountries": ["IN", "BR", "HK"]
    }
  }
}
```


Error Responses

Code	HTTP	Message
`INVALID_AMOUNT`	400	Amount must be greater than 0
`INVALID_CURRENCY`	400	Currency not supported
`UNAUTHORIZED`	401	Invalid API key
`LIMIT_EXCEEDED`	403	Transaction limit exceeded
`VALIDATION_ERROR`	422	Request validation failed

Examples

CURL

```
curl -X POST https://api.zenpays.com/api/v1/payment-intents \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2026-02-24T05:15:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "amount": 100,
    "currency": "INR",
    "paymentMethod": "upi",
    "customerEmail": "test@example.com",
    "customerPhone": "+911234567890",
    "customerFirstName": "Test",
    "customerLastName": "Customer",
    "customerCountry": "IN",
    "description": "Test payment"
  }'
```

JAVASCRIPT

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

const apiKey = process.env.ZENPAYS_API_KEY;
const secretSalt = process.env.ZENPAYS_SECRET_SALT;
const timestamp = new Date().toISOString();

const body = {
  amount: 100,
  currency: 'INR',
  paymentMethod: 'upi',
  customerEmail: 'test@example.com',
  customerPhone: '+911234567890',
  customerFirstName: 'Test',
  customerLastName: 'Customer',
  customerCountry: 'IN',
  description: 'Test payment',
};

const signature = crypto
  .createHmac('sha256', secretSalt)
  .update(timestamp + JSON.stringify(body))
  .digest('hex');

const response = await fetch('https://api.zenpays.com/api/v1/payment-intents', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(body),
});

const result = await response.json();
console.log(result.data.intentId); // pi_1771910100598_prckpcg01
console.log(result.data.paymentPageUrl); // checkout URL
```

PYTHON

```python
import hmac
import hashlib
import json
import os
from datetime import datetime
import requests

api_key = os.environ["ZENPAYS_API_KEY"]
secret_salt = os.environ["ZENPAYS_SECRET_SALT"]
timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")

body = {
    "amount": 100,
    "currency": "INR",
    "paymentMethod": "upi",
    "customerEmail": "test@example.com",
    "customerPhone": "+911234567890",
    "customerFirstName": "Test",
    "customerLastName": "Customer",
    "customerCountry": "IN",
    "description": "Test payment",
}

data = timestamp + json.dumps(body, separators=(",", ":"))
signature = hmac.new(
    secret_salt.encode(),
    data.encode(),
    hashlib.sha256
).hexdigest()

response = requests.post(
    "https://api.zenpays.com/api/v1/payment-intents",
    headers={
        "Authorization": f"Bearer {api_key}",
        "X-Timestamp": timestamp,
        "X-Signature": signature,
        "X-Secret-Salt": secret_salt,
        "Content-Type": "application/json",
    },
    json=body,
)

result = response.json()
print(result["data"]["intentId"])  # pi_1771910100598_prckpcg01
print(result["data"]["paymentPageUrl"])  # checkout URL
```

SDK (RECOMMENDED)

```
// JavaScript SDK - handles authentication automatically
const intent = await zenpays.payments.createPaymentIntent({
  amount: 100,
  currency: 'INR',
  paymentMethod: 'upi',
  customerEmail: 'test@example.com',
  customerPhone: '+911234567890',
  customerFirstName: 'Test',
  customerLastName: 'Customer',
  customerCountry: 'IN',
  description: 'Test payment',
});

console.log(intent.intentId); // pi_1771910100598_prckpcg01
console.log(intent.paymentPageUrl); // checkout URL
```

Payment Intent Status

Status	Description
`requires_payment_method`	Awaiting payment method selection
`processing`	Payment is being processed
`succeeded`	Payment completed successfully
`failed`	Payment failed
`cancelled`	Payment was cancelled
`expired`	Payment intent expired

Next Steps

After creating a payment intent:

1. Redirect customer to checkout or confirm payment
2. Get Payment Intent to check status
3. Confirm Payment with customer details
4. Set up Webhooks for async status updates

REST API / PAYMENTS

Confirm Payment

Confirm a payment intent with customer and payment method details. This initiates the actual payment processing.

`POST` **`/api/v1/payment-intents/{intentId}/confirm`**

Request

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Payment intent ID (e.g., `pi_xxxxx`)

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`X-Request-Id`	No	Unique request identifier (auto-generated if not provided)

> **Note**
> This endpoint is **public** — no API key or signature required. It is called from the checkout page on behalf of the customer.

Body

UPI

```
{
  "customerDetails": {
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "+911234567890",
    "address": {
      "country": "IN"
    }
  },
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "UPI",
    "upiId": "john@upi"
  },
  "deviceInfo": {
    "userAgent": "Mozilla/5.0 ...",
    "browserInfo": "Mozilla/5.0 ...",
    "screenWidth": 1512,
    "screenHeight": 982,
    "deviceFingerprint": "a1b2c3d4e5f6",
    "metadata": {
      "selectedPaymentMethod": "fiat",
      "selectedMethod": "upi",
      "specificPaymentMethod": "UPI",
      "customerCurrency": "INR"
    }
  }
}
```

CARD

> **Tip**
>
> Card details (`cardNumber` , `expiryMonth` , `expiryYear` , `cvv` , `cardHolderName`) are optional. When omitted, the customer will be redirected to the TSP's hosted payment page to enter card details securely.

```
{
  "customerDetails": {
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "+911234567890",
    "address": {
      "country": "IN"
    }
  },
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "CARD",
    "cardNumber": "4242424242424242",
    "expiryMonth": "12",
    "expiryYear": "2028",
    "cvv": "123",
    "cardHolderName": "John Doe"
  },
  "deviceInfo": {
    "userAgent": "Mozilla/5.0 ...",
    "browserInfo": "Mozilla/5.0 ...",
    "screenWidth": 1512,
    "screenHeight": 982,
    "deviceFingerprint": "a1b2c3d4e5f6",
    "metadata": {
      "selectedPaymentMethod": "fiat",
      "selectedMethod": "card",
      "specificPaymentMethod": "CARD",
      "customerCurrency": "INR"
    }
  }
}
```

BANK TRANSFER

```
{
  "customerDetails": {
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "+250793045233",
    "address": {
      "country": "RW"
    }
  },
  "paymentMethodDetails": {
    "paymentMethod": "FIAT",
    "paymentType": "BANK_TRANSFER"
  },
  "deviceInfo": {
    "userAgent": "Mozilla/5.0 ...",
    "browserInfo": "Mozilla/5.0 ...",
    "screenWidth": 1512,
    "screenHeight": 982,
    "deviceFingerprint": "a1b2c3d4e5f6",
    "metadata": {
      "selectedPaymentMethod": "fiat",
      "selectedMethod": "card",
      "specificPaymentMethod": "BANK_TRANSFER",
      "customerCurrency": "USD"
    }
  }
}
```

Field Reference

`customerDetails` (required)

Parameter	Type	Required	Description
`name`	`string`	Yes	Customer full name (max 100 chars)
`email`	`string`	Yes	Customer email
`phone`	`string`	No	Customer phone (max 15 chars)
`address.country`	`string`	No	ISO 3166-1 alpha-2 country code

`paymentMethodDetails` (optional)

Parameter	Type	Required	Description
`paymentMethod`	`string`	Yes	`FIAT` or `CRYPTO`
`paymentType`	`string`	Yes	Payment type within the method (see below)

Fiat payment types (`paymentMethod: "FIAT"`):

`paymentType`	Additional Fields	Description
`UPI`	`upiId` (string) — UPI VPA e.g. `user@upi`	UPI payment
`CARD`	`cardNumber` , `expiryMonth` , `expiryYear` , `cvv` , `cardHolderName`	Card payment
`NETBANKING`	`bankCode` (string) — bank code	Net banking
`BANK_TRANSFER`	—	Bank transfer
`PIX`	—	PIX (Brazil)
`WALLET`	—	Wallet payment

Crypto payment types (`paymentMethod: "CRYPTO"`):

`paymentType`	Additional Fields	Description
`DIRECT_CRYPTO`	`chain` (string) — blockchain e.g. `ethereum` , `tron`	Direct crypto payment
`BUY_WITH_WALLET`	`chain` (string) — blockchain	Crypto purchase via wallet

`deviceInfo` (optional)

Parameter	Type	Description
`userAgent`	`string`	Browser user agent
`browserInfo`	`string`	Browser info string
`screenWidth`	`number`	Screen width in pixels
`screenHeight`	`number`	Screen height in pixels
`deviceFingerprint`	`string`	Device fingerprint hash
`metadata`	`object`	Additional context (selected method, customer currency, etc.)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "intentId": "pi_1771910100598_prckpcg01",
    "status": "processing",
    "redirectUrl": "https://checkout.zenpays.com/pay/xxxxx",
    "externalTransactionId": "txn_abc123"
  },
  "message": "Payment confirmed successfully",
  "meta": {
    "request_id": "req_abc123",
    "timestamp": "2026-02-24T05:15:00.599Z",
    "processing_time_ms": 1250,
    "api_version": "v1",
    "endpoint": "POST /payment-intent/:intentId/confirm"
  }
}
```


Error Responses

Code	HTTP	Message
`INVALID_REQUEST`	400	Missing required fields
`UNAUTHORIZED`	401	Invalid API key
`PAYMENT_INTENT_NOT_FOUND`	404	Intent doesn't exist
`STATE_CONFLICT`	409	Intent not in valid state
`PAYMENT_FAILED`	422	Payment processing failed
`CHANNEL_LIMIT_MIN_EXCEEDED`	422	Amount below minimum for payment method
`CHANNEL_LIMIT_MAX_EXCEEDED`	422	Amount exceeds maximum for payment method

Examples

CURL

```
# No authentication required — this is a public endpoint
curl -X POST "https://api.zenpays.com/api/v1/payment-intents/pi_1771910100598_prckpcg01/confirm" \
  -H "Content-Type: application/json" \
  -d '{
    "customerDetails": {
      "name": "John Doe",
      "email": "john@example.com",
      "phone": "+911234567890",
      "address": { "country": "IN" }
    },
    "paymentMethodDetails": {
      "paymentMethod": "FIAT",
      "paymentType": "UPI",
      "upiId": "john@upi"
    },
    "deviceInfo": {
      "userAgent": "Mozilla/5.0...",
      "deviceFingerprint": "a1b2c3d4e5f6"
    }
  }'
```

JAVASCRIPT

```
const intentId = 'pi_1771910100598_prckpcg01';

const body = {
  customerDetails: {
    name: 'John Doe',
    email: 'john@example.com',
    phone: '+911234567890',
    address: { country: 'IN' },
  },
  paymentMethodDetails: {
    paymentMethod: 'FIAT',
    paymentType: 'UPI',
    upiId: 'john@upi',
  },
  deviceInfo: {
    userAgent: navigator.userAgent,
    screenWidth: window.screen.width,
    screenHeight: window.screen.height,
    deviceFingerprint: 'a1b2c3d4e5f6',
    metadata: {
      selectedPaymentMethod: 'fiat',
      selectedMethod: 'upi',
      specificPaymentMethod: 'UPI',
      customerCurrency: 'INR',
    },
  },
};

// No authentication needed — public endpoint
const response = await fetch(
  `https://api.zenpays.com/api/v1/payment-intents/${intentId}/confirm`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(body),
  }
);

const result = await response.json();

if (result.data.redirectUrl) {
  window.location.href = result.data.redirectUrl;
}
```

PYTHON

```python
import requests

intent_id = "pi_1771910100598_prckpcg01"

body = {
    "customerDetails": {
        "name": "John Doe",
        "email": "john@example.com",
        "phone": "+911234567890",
        "address": {"country": "IN"},
    },
    "paymentMethodDetails": {
        "paymentMethod": "FIAT",
        "paymentType": "UPI",
        "upiId": "john@upi",
    },
    "deviceInfo": {
        "userAgent": "Mozilla/5.0...",
        "deviceFingerprint": "a1b2c3d4e5f6",
    },
}

# No authentication needed — public endpoint
response = requests.post(
    f"https://api.zenpays.com/api/v1/payment-intents/{intent_id}/confirm",
    headers={"Content-Type": "application/json"},
    json=body,
)

result = response.json()
if result["data"].get("redirectUrl"):
    print(f"Redirect to: {result['data']['redirectUrl']}")
```

SDK (RECOMMENDED)

```javascript
// JavaScript SDK
const result = await zenpays.payments.confirmPayment('pi_1771910100598_prckpcg01', {
  customerDetails: {
    name: 'John Doe',
    email: 'john@example.com',
    phone: '+911234567890',
    address: { country: 'IN' },
  },
  paymentMethodDetails: {
    paymentMethod: 'FIAT',
    paymentType: 'UPI',
    upiId: 'john@upi',
  },
});

if (result.redirectUrl) {
  window.location.href = result.redirectUrl;
}
```

Handling Payment Actions

The response may require additional actions:

Action Type	Description	Next Step
`redirect`	Redirect to payment page	Navigate to `redirectUrl`
`3ds_authentication`	3D Secure required	Show 3DS popup/iframe
`otp_verification`	OTP required	Display OTP input
`none`	No action needed	Payment complete

Related Endpoints

- Create Payment Intent
- Get Payment Intent
- Webhooks - Handle async status updates

REST API / PAYMENTS

Get Payment Intent

Retrieve details of a payment intent by its ID.

`GET` **`/api/v1/payment-intents/{intentId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Payment intent ID (e.g., `pi_xxxxx`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Note

For GET requests, use an empty object `{}` when calculating the signature.

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "intentId": "pi_1771910100598_prckpcg01",
    "status": "succeeded",
    "amount": 100,
    "currency": "INR",
    "transactionId": "txn_abc123",
    "externalTransactionId": "ext_xyz789",
    "createdAt": "2026-02-24T05:15:00.599Z",
    "completedAt": "2026-02-24T05:16:30.000Z",
    "metadata": {},
    "events": [
      {
        "event": "payment_intent.created",
        "timestamp": "2026-02-24T05:15:00.599Z"
      },
      {
        "event": "payment_intent.confirmed",
        "timestamp": "2026-02-24T05:15:05.000Z"
      },
      {
        "event": "payment_intent.succeeded",
        "timestamp": "2026-02-24T05:16:30.000Z"
      }
    ]
  },
  "message": "Payment intent retrieved successfully",
  "meta": {
    "request_id": "req_abc123",
    "timestamp": "2026-02-24T05:17:00.000Z",
    "processing_time_ms": 45,
    "api_version": "v1",
    "endpoint": "GET /payment-intent/:intentId"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`PAYMENT_INTENT_NOT_FOUND`	404	Payment intent doesn't exist

Examples

CURL

```
curl -X GET "https://api.zenpays.com/api/v1/payment-intents/pi_1234567890abcdef" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

JAVASCRIPT

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

const apiKey = process.env.ZENPAYS_API_KEY;
const secretSalt = process.env.ZENPAYS_SECRET_SALT;
const timestamp = new Date().toISOString();
const intentId = 'pi_1234567890abcdef';

// For GET requests, use empty object
const signature = crypto
  .createHmac('sha256', secretSalt)
  .update(timestamp + '{}')
  .digest('hex');

const response = await fetch(
  `https://api.zenpays.com/api/v1/payment-intents/${intentId}`,
  {
    method: 'GET',
    headers: {
      'Authorization': `Bearer ${apiKey}`,
      'X-Timestamp': timestamp,
      'X-Signature': signature,
      'X-Secret-Salt': secretSalt,
    },
  }
);

const result = await response.json();
console.log(result.data.status); // "succeeded"
```

PYTHON

```python
import hmac
import hashlib
import os
from datetime import datetime
import requests

api_key = os.environ["ZENPAYS_API_KEY"]
secret_salt = os.environ["ZENPAYS_SECRET_SALT"]
timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")
intent_id = "pi_1234567890abcdef"

# For GET requests, use empty object
data = timestamp + "{}"
signature = hmac.new(
    secret_salt.encode(),
    data.encode(),
    hashlib.sha256
).hexdigest()

response = requests.get(
    f"https://api.zenpays.com/api/v1/payment-intents/{intent_id}",
    headers={
        "Authorization": f"Bearer {api_key}",
        "X-Timestamp": timestamp,
        "X-Signature": signature,
        "X-Secret-Salt": secret_salt,
    },
)

result = response.json()
print(result["data"]["status"])  # "succeeded"
```

SDK (RECOMMENDED)

```javascript
// JavaScript SDK - handles authentication automatically
const intent = await zenpays.payments.getPaymentIntent('pi_1234567890abcdef');
console.log(intent.status); // "succeeded"
```

Payment Intent Status

Status	Description
`requires_payment_method`	Awaiting payment method selection
`processing`	Payment is being processed
`succeeded`	Payment completed successfully
`failed`	Payment failed
`cancelled`	Payment was cancelled
`expired`	Payment intent expired

Related Endpoints

- Create Payment Intent
- Confirm Payment

- List Transactions (https://docs.zenpays.com/docs/rest-api/endpoints/payments/list-transactions)

Transactions

REST API / TRANSACTIONS

List Transactions

Retrieve a paginated list of transactions with optional filters.

`GET` **`/merchant/api/v1/transactions`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page (max 100)
`status`	`string`	`-`	Filter by status (`initiated` , `processing` , `success` , `failed` , `cancelled` , `refunded` , `partially_refunded`)
`currency`	`string`	`-`	Filter by currency code
`paymentMethod`	`string`	`-`	Filter by payment method
`customerId`	`string`	`-`	Filter by customer ID
`startDate`	`string`	`-`	Created after (ISO 8601)
`endDate`	`string`	`-`	Created before (ISO 8601)
`minAmount`	`number`	`-`	Minimum amount
`maxAmount`	`number`	`-`	Maximum amount
`search`	`string`	`-`	Search by transaction ID or reference

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "data": [
      {
        "id": "txn_1234567890abcdef",
        "intent_id": "pi_xxxxx",
        "merchant_id": "mer_xxxxx",
        "customer_id": "cus_xxxxx",
        "customer_name": "John Doe",
        "customer_email": "john@example.com",
        "customer_phone": "+1234567890",
        "amount": 10000,
        "currency": "USD",
        "status": "success",
        "payment_method": "upi",
        "description": "Order #123",
        "metadata": {
          "orderId": "order_123"
        },
        "created_at": "2024-01-15T10:30:00.000Z",
        "updated_at": "2024-01-15T10:31:00.000Z"
      }
    ],
    "pagination": {
      "total": 1500,
      "page": 1,
      "limit": 20,
      "totalPages": 75
    }
  },
  "message": "Transactions retrieved successfully",
  "meta": {}
}
```


Transaction Status

Status	Description
`initiated`	Transaction initiated
`processing`	Payment being processed
`success`	Payment completed successfully
`failed`	Payment failed
`cancelled`	Transaction cancelled
`refunded`	Transaction refunded
`partially_refunded`	Transaction partially refunded

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/transactions?status=success&limit=50" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const response = await zenpays.transactions.list({
  status: 'success',
  limit: 50,
});

const { data: transactions, pagination } = response.data;
console.log(`Found ${pagination.total} transactions`);
transactions.forEach(txn => {
  console.log(`${txn.id}: ${txn.amount} ${txn.currency}`);
});
```

Related Endpoints

- Get Transaction
- Export Transactions

REST API / TRANSACTIONS

Get Transaction

Retrieve details of a specific transaction by ID.

`GET` **`/merchant/api/v1/transactions/{transactionId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`transactionId`	`string`	Yes	Transaction ID (e.g., `txn_xxxxx`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "id": "txn_xxxxx",
    "intent_id": "pi_xxxxx",
    "merchant_id": "mer_xxxxx",
    "customer_id": "cus_xxxxx",
    "customer_name": "John Doe",
    "customer_email": "john@example.com",
    "customer_phone": "+1234567890",
    "amount": 1000,
    "currency": "USD",
    "status": "success",
    "payment_method": "upi",
    "description": "Order #123",
    "metadata": {
      "orderId": "order_123"
    },
    "created_at": "2024-01-15T10:30:00.000Z",
    "updated_at": "2024-01-15T10:35:00.000Z"
  },
  "message": "Transaction retrieved successfully",
  "meta": {}
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`TRANSACTION_NOT_FOUND`	404	Transaction doesn't exist

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/transactions/txn_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const transaction = await zenpays.payments.getTransaction('txn_xxxxx');
console.log(`Status: ${transaction.status}`);
```

REST API / TRANSACTIONS

Export Transactions

Export transactions to CSV or XLSX format.

`POST` **`/merchant/api/v1/transactions/export`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Query Parameters

Parameter	Type	Default	Description
`format`	`string`	`csv`	Export format (`csv` , `xlsx`)
`status`	`string`	`-`	Filter by status
`paymentMethod`	`string`	`-`	Filter by payment method
`startDate`	`string`	`-`	Start date (ISO 8601)
`endDate`	`string`	`-`	End date (ISO 8601)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "downloadUrl": "https://exports.zenpays.com/transactions/exp_xxxxx.csv",
    "expiresAt": "2024-01-15T11:30:00.000Z",
    "recordCount": 15000,
    "format": "csv"
  },
  "message": "Export initiated successfully",
  "meta": {}
}
```

Examples

CURL

```
curl -X POST "https://api.zenpays.com/merchant/api/v1/transactions/export?
format=csv&status=success" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

SDK

```
const exportUrl = await zenpays.payments.exportTransactions(
  { status: 'success', from: '2024-01-01' },
  'csv'
);
console.log(`Download: ${exportUrl}`);
```

REST API / TRANSACTIONS

Get Customer Transactions

Retrieve transactions for a specific customer.

`GET` **`/merchant/api/v1/transactions/customer/{customerId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer ID (e.g., `cus_xxxxx`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page (max 100)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "data": [
      {
        "id": "txn_xxxxx",
        "intent_id": "pi_xxxxx",
        "merchant_id": "mer_xxxxx",
        "customer_id": "cus_xxxxx",
        "customer_name": "John Doe",
        "customer_email": "john@example.com",
        "customer_phone": "+1234567890",
        "amount": 1000,
        "currency": "USD",
        "status": "success",
        "payment_method": "upi",
        "description": "Order #456",
        "metadata": {
          "orderId": "order_456"
        },
        "created_at": "2024-01-15T10:30:00.000Z",
        "updated_at": "2024-01-15T10:31:00.000Z"
      }
    ],
    "pagination": {
      "total": 45,
      "page": 1,
      "limit": 20,
      "totalPages": 3
    }
  },
  "message": "Customer transactions retrieved successfully",
  "meta": {}
}
```


Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/transactions/customer/cus_xxxxx?limit=50" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const { data } = await zenpays.payments.getTransactionsByCustomer('cus_xxxxx', {
  limit: 50,
});
```

Customers

REST API / CUSTOMERS

Create Customer

Create a new customer record for tracking payments and transactions.

`POST` **`/merchant/api/v1/customers`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`email`	`string`	Yes	Customer email address
`phone`	`string`	No	Phone number (10-15 characters)
`firstName`	`string`	No	Customer first name (1-100 characters)
`lastName`	`string`	No	Customer last name (1-100 characters)
`dateOfBirth`	`string`	No	Date of birth (YYYY-MM-DD)
`country`	`string`	Yes	ISO country code
`ipAddress`	`string`	Yes	Customer IPv4 address
`deviceFingerprint`	`string`	No	Device fingerprint for fraud detection (10-255 characters)
`userAgent`	`string`	No	Browser user agent string (1-500 characters)
`metadata`	`object`	No	Custom key-value pairs

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "email": "john@example.com",
    "phone": "+1234567890",
    "firstName": "John",
    "lastName": "Doe",
    "country": "US",
    "status": "active",
    "riskLevel": "low",
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Customer created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 120,
    "api_version": "v1",
    "endpoint": "POST /customers",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`DUPLICATE_CUSTOMER`	409	Customer with email already exists

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/customers \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "email": "john@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "phone": "+1234567890",
    "country": "US",
    "ipAddress": "203.0.113.42"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/customers', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    email: 'john@example.com',
    firstName: 'John',
    lastName: 'Doe',
    phone: '+1234567890',
    country: 'US',
    ipAddress: '203.0.113.42',
  }),
});

const result = await response.json();
console.log(result.data.customerId); // cust_a1b2c3d4e5f6g7h8
```

SDK

```
const customer = await zenpays.customers.create({
  email: 'john@example.com',
  firstName: 'John',
  lastName: 'Doe',
  phone: '+1234567890',
  country: 'US',
  ipAddress: '203.0.113.42',
});

console.log(customer.customerId); // cust_a1b2c3d4e5f6g7h8
```

Related Endpoints

- Get Customer
- List Customers
- Update Customer

REST API / CUSTOMERS

Get Customer

Retrieve details of a specific customer by ID.

`GET` **`/merchant/api/v1/customers/{customerId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer ID (e.g., `cust_a1b2c3d4e5f6g7h8`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "email": "john@example.com",
    "phone": "+1234567890",
    "firstName": "John",
    "lastName": "Doe",
    "country": "US",
    "status": "active",
    "riskLevel": "low",
    "riskScore": 15,
    "totalTransactions": 15,
    "totalAmount": 25000,
    "successfulTransactions": 14,
    "failedTransactions": 1,
    "averageTransactionAmount": 1785.71,
    "lastTransactionDate": "2024-01-14T15:30:00.000Z",
    "createdAt": "2024-01-01T10:00:00.000Z",
    "updatedAt": "2024-01-14T15:30:00.000Z"
  },
  "message": "Customer retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 45,
    "api_version": "v1",
    "endpoint": "GET /customers/:id",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`CUSTOMER_NOT_FOUND`	404	Customer doesn't exist

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/customers/cust_a1b2c3d4e5f6g7h8" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const customer = await zenpays.customers.get('cust_a1b2c3d4e5f6g7h8');
console.log(customer.email); // john@example.com
```

Related Endpoints

- Update Customer
- Customer History
- Customer Risk Profile

REST API / CUSTOMERS

List Customers

Retrieve a paginated list of customers with optional filters.

`GET` **`/merchant/api/v1/customers`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`10`	Items per page (max 100)
`search`	`string`	`-`	Search by email, name, or customer ID (alias for `searchTerm`)
`searchTerm`	`string`	`-`	Search by email, name, or customer ID
`status`	`string`	`-`	Filter by status (`active`, `inactive`, `blocked`, `suspended`)
`riskLevel`	`string`	`-`	Filter by risk level (`low`, `medium`, `high`, `warning`, `critical`, `risky`, `vip`)
`currency`	`string`	`-`	Filter by transaction currency
`sortBy`	`string`	`createdAt`	Sort field (`createdAt`, `updatedAt`, `riskScore`, `totalAmount`, `lastTransactionDate`)
`sortOrder`	`string`	`DESC`	Sort direction (`ASC`, `DESC`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "customers": [
      {
        "customerId": "cust_a1b2c3d4e5f6g7h8",
        "email": "john@example.com",
        "firstName": "John",
        "lastName": "Doe",
        "phone": "+1234567890",
        "country": "US",
        "status": "active",
        "riskLevel": "low",
        "riskScore": 10,
        "totalTransactions": 15,
        "totalSpent": 25000,
        "lastPurchase": "2024-01-14T15:30:00.000Z",
        "createdAt": "2024-01-01T10:00:00.000Z"
      }
    ],
    "total": 150,
    "page": 1,
    "limit": 10,
    "hasMore": true
  },
  "message": "Customers retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 85,
    "api_version": "v1",
    "endpoint": "GET /customers",
    "user_type": "merchant"
  }
}
```


Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/customers?status=active&limit=50" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const { data } = await zenpays.customers.list({
  status: 'active',
  limit: 50,
});

console.log(`Found ${data.total} customers`);
```

Related Endpoints

- Create Customer
- Get Customer

REST API / CUSTOMERS

Update Customer

Update an existing customer's information.

`PUT` **`/merchant/api/v1/customers/{customerId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer ID (e.g., `cust_a1b2c3d4e5f6g7h8`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Description
`email`	`string`	Customer email address
`phone`	`string`	Phone number (10-15 characters)
`firstName`	`string`	Customer first name (1-100 characters)
`lastName`	`string`	Customer last name (1-100 characters)
`dateOfBirth`	`string`	Date of birth (YYYY-MM-DD)
`country`	`string`	ISO country code
`ipAddress`	`string`	Customer IPv4 address
`deviceFingerprint`	`string`	Device fingerprint (10-255 characters)
`metadata`	`object`	Custom key-value pairs

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "email": "john.updated@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "phone": "+1234567890",
    "country": "US",
    "status": "active",
    "riskLevel": "low",
    "createdAt": "2024-01-01T10:00:00.000Z",
    "updatedAt": "2024-01-15T10:35:00.000Z"
  },
  "message": "Customer updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:35:00.000Z",
    "processing_time_ms": 60,
    "api_version": "v1",
    "endpoint": "PUT /customers/:id",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`CUSTOMER_NOT_FOUND`	404	Customer doesn't exist

Examples

CURL

```
curl -X PUT "https://api.zenpays.com/merchant/api/v1/customers/cust_a1b2c3d4e5f6g7h8" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "email": "john.updated@example.com",
    "phone": "+1987654321"
  }'
```

SDK

```
const customer = await zenpays.customers.update('cust_a1b2c3d4e5f6g7h8', {
  email: 'john.updated@example.com',
  phone: '+1987654321',
});
```

REST API / CUSTOMERS

Get Customer History

Retrieve transaction history for a specific customer.

`GET` **`/merchant/api/v1/customers/{customerId}/history`**

Request

Path Parameters

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer ID (e.g., `cust_a1b2c3d4e5f6g7h8`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`limit`	`integer`	`50`	Number of records to return

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "transactions": [
      {
        "id": "txn_xxxxx",
        "amount": 1000,
        "currency": "USD",
        "status": "success",
        "paymentMethod": "upi",
        "createdAt": "2024-01-14T15:30:00.000Z"
      }
    ],
    "activities": [
      {
        "id": 1,
        "activityType": "payment_completed",
        "description": "Payment of 1000 USD completed",
        "ipAddress": "203.0.113.42",
        "deviceFingerprint": "fp_xxxxx",
        "riskLevel": "low",
        "isSuspicious": false,
        "metadata": {},
        "createdAt": "2024-01-14T15:30:00.000Z"
      }
    ]
  },
  "message": "Customer history retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 95,
    "api_version": "v1",
    "endpoint": "GET /customers/:id/history",
    "user_type": "merchant"
  }
}
```


Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/customers/cust_a1b2c3d4e5f6g7h8/history?
limit=50" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const history = await zenpays.customers.getHistory('cust_a1b2c3d4e5f6g7h8', { limit: 50 });
console.log(`Transactions: ${history.transactions.length}, Activities:
${history.activities.length}`);
```

REST API / CUSTOMERS

Get Customer Risk Profile

Retrieve the risk assessment profile for a customer.

`GET` **`/merchant/api/v1/customers/{customerId}/risk`**

Request

Path Parameters

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer ID (e.g., `cust_a1b2c3d4e5f6g7h8`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "riskLevel": "low",
    "riskScore": 15,
    "fraudScore": 5,
    "reasons": [],
    "recommendations": ["Approve transaction"],
    "requiresManualReview": false,
    "allowTransaction": true,
    "blockedReasons": []
  },
  "message": "Customer risk profile retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 70,
    "api_version": "v1",
    "endpoint": "GET /customers/:id/risk",
    "user_type": "merchant"
  }
}
```


Risk Levels

Level	Score Range	Description
`vip`	-	VIP customer, trusted with high success rate
`low`	0-29	Low risk, standard processing
`medium`	30-59	Moderate risk, enhanced monitoring
`high`	60-84	High risk, manual review recommended
`warning`	-	Warning level, elevated monitoring
`risky`	-	Risky behavior detected
`critical`	85-100	Very high risk, transactions may be blocked

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/customers/cust_a1b2c3d4e5f6g7h8/risk" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const risk = await zenpays.customers.getRiskProfile('cust_a1b2c3d4e5f6g7h8');
console.log(`Risk level: ${risk.riskLevel}, Score: ${risk.riskScore}`);
```

REST API / CUSTOMERS

Get Top Customers

Retrieve top customers by transaction volume.

`GET` **`/merchant/api/v1/customers/top`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`limit`	`integer`	`10`	Number of customers (max 100)
`startDate`	`string`	`-`	Start date (ISO 8601)
`endDate`	`string`	`-`	End date (ISO 8601)
`currency`	`string`	`-`	Filter by currency

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "customerId": "cust_a1b2c3d4e5f6g7h8",
      "email": "vip@example.com",
      "name": "Jane Smith",
      "lifetimeValue": 150000,
      "totalTransactions": 45
    },
    {
      "customerId": "cust_c3d4e5f6g7h8i9j0",
      "email": "john@example.com",
      "name": "John Doe",
      "lifetimeValue": 125000,
      "totalTransactions": 38
    }
  ],
  "message": "Top customers retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 150,
    "api_version": "v1",
    "endpoint": "GET /customers/top",
    "user_type": "merchant"
  }
}
```


Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/customers/top?limit=20&currency=USD" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

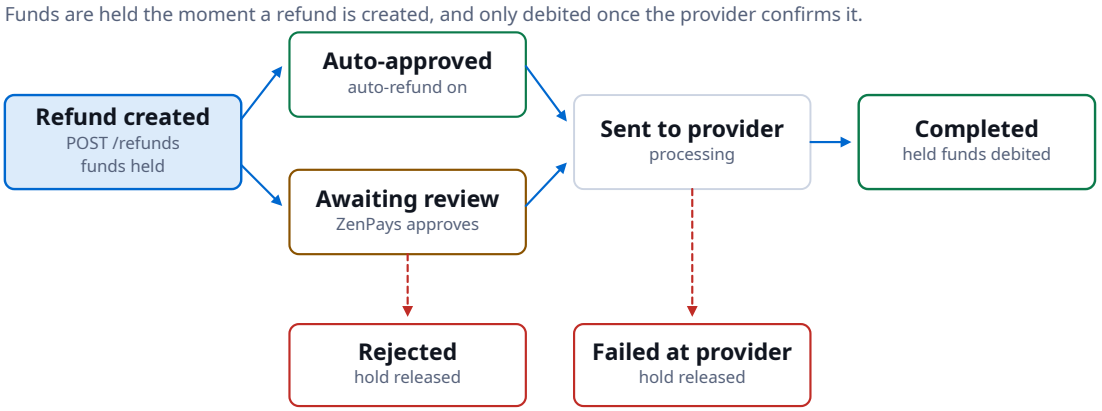
```
const topCustomers = await zenpays.customers.getTop({
  limit: 20,
  currency: 'USD',
});
```

Refunds

REST API / REFUNDS

Create Refund

Create a refund for a completed INR transaction. Supports full and partial refunds. Refunds are processed as payouts to the customer's bank account via IMPS.



How a refund is approved, sent and settled

`POST` **`/merchant/api/v1/refund-intents`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Idempotency-Key`	No	Unique key to prevent duplicate refunds

Body Parameters

Parameter	Type	Description
`transactionId`	`string`	ID of the original transaction to refund
`reason`	`string`	Reason for the refund
`amount`	`number`	Refund amount. Omit for a full refund. Must be > 0 and ≤ the remaining refundable amount.

Parameter	Type	Description
`customerId`	`string`	Customer ID. Resolved from the transaction if omitted.
`beneficiaryEmail`	`string`	Beneficiary email address
`beneficiaryAccount`	`string`	Bank account number
`beneficiaryIfsc`	`string`	IFSC code (e.g. `HDFC0001234`)
`beneficiaryName`	`string`	Full name of the beneficiary
`beneficiaryMobile`	`string`	Beneficiary mobile number with country code
`beneficiaryCity`	`string`	Beneficiary city

Beneficiary Fields

INR refunds are processed as IMPS payouts to the customer's bank account. You must provide:

Field	Required	Example
`beneficiaryEmail`	Yes	`"rahul@example.com"`
`beneficiaryAccount`	Yes	`"50100012345678"`
`beneficiaryIfsc`	Yes	`"HDFC0001234"`

Optional fields

Field	Description	Example
`beneficiaryName`	Account holder name	`"Rahul Sharma"`
`beneficiaryMobile`	Mobile with country code	`"+919876543210"`
`beneficiaryCity`	City	`"Mumbai"`

> **Warning**
> If `beneficiaryAccount` or `beneficiaryIfsc` is missing, the API will return an error.

> **Try it out**
> Test refund creation interactively using the API Simulator.

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "refundId": "refund_1774683026458_oann5ibjf",
    "merchantId": "ZP_FIN_1774592804_0781001264",
    "customerId": "cust_mn8yfewm_37ntht",
    "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
    "paymentIntentId": "pi_1774619159530_66d8s8v25",
    "amount": 100,
    "originalAmount": 100,
    "currency": "INR",
    "refundType": "full",
    "status": "pending_approval",
    "tspProvider": "sulifu_pay",
    "externalRefundId": null,
    "reason": "Customer requested refund",
    "createdAt": "2026-04-10T14:22:01.000Z",
    "completedAt": null,
    "failureReason": null,
    "processingTimeMs": null
  }
}
```


Error Responses

Code	HTTP	Message
`REFUND_NOT_ALLOWED`	400	Transaction cannot be refunded (must be `success` or `partial_refund` status)
`REFUND_AMOUNT_EXCEEDED`	400	Requested amount exceeds the remaining refundable amount
`MISSING_BENEFICIARY_FIELDS`	400	`beneficiaryEmail`, `beneficiaryAccount`, or `beneficiaryIfsc` is missing
`UNAUTHORIZED`	401	Invalid or missing API key
`TRANSACTION_NOT_FOUND`	404	Transaction does not exist or does not belong to this merchant
`DUPLICATE_REQUEST`	409	A refund with this idempotency key already exists
`ALREADY_FULLY_REFUNDED`	422	Transaction has already been fully refunded

Request Examples

Full refund

Refund the entire transaction amount to the customer's bank account:

```
{
  "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
  "reason": "Customer requested full refund",
  "beneficiaryEmail": "rahul@example.com",
  "beneficiaryAccount": "50100012345678",
  "beneficiaryIfsc": "HDFC0001234"
}
```

Response:

```
{
  "success": true,
  "data": {
    "refundId": "refund_1774683026458_oann5ibjf",
    "amount": 100,
    "originalAmount": 100,
    "currency": "INR",
    "refundType": "full",
    "status": "pending_approval",
    "tspProvider": "sulifu_pay",
    "reason": "Customer requested full refund",
    "createdAt": "2026-04-10T14:22:01.000Z"
  }
}
```

Partial refund

Refund 500 INR out of a 2000 INR transaction:

```
{
  "transactionId": "txn_1774700012000_1_abc12345_9z8y7x",
  "amount": 500,
  "reason": "Partial item return",
  "beneficiaryEmail": "priya@example.com",
  "beneficiaryAccount": "91020034567890",
  "beneficiaryIfsc": "SBIN0001234",
  "beneficiaryName": "Priya Sharma"
}
```

Response:

```
{
  "success": true,
  "data": {
    "refundId": "refund_1774700099000_kp3m7dxw2",
    "amount": 500,
    "originalAmount": 2000,
    "currency": "INR",
    "refundType": "partial",
    "status": "pending_approval",
    "tspProvider": "sulifu_pay",
    "reason": "Partial item return",
    "createdAt": "2026-04-10T14:35:00.000Z"
  }
}
```

Amount exceeds refundable balance

If you try to refund more than the remaining balance:

```
{
  "success": false,
  "error": "Requested refund amount 1800 exceeds refundable amount 1250.00 INR"
}
```

Refund Lifecycle

After a refund is created, it moves through these statuses:

Status	What happens
`pending_approval`	Refund created. The refund amount is **blocked (escrowed)** in the merchant wallet. Waiting for admin to approve or reject.
`approved`	Admin approved. The refund is being sent to the payment provider.
`processing`	Payment provider is executing the payout to the customer.
`completed`	Payout succeeded. Blocked funds are **debited** from the merchant wallet. Transaction `refundedAmount` is updated.
`failed`	Payment provider failed to process. Blocked funds are **released back** to the merchant wallet.
`rejected`	Admin rejected the refund. Blocked funds are **released back**.
`cancelled`	Merchant cancelled the refund before processing. Blocked funds are **released back**.

Flow

```
                                  +---> rejected (funds released)
                                  |
pending_approval ---> approved ---> processing ---> completed (funds debited)
       |                                  |
       +---> cancelled (funds released)   +---> failed (funds released)
```

Auto-approval

If `auto_refund_enabled` is `true` in the merchant configuration, refunds skip `pending_approval` and go directly to `processing`.

Auto-cancellation

If a refund stays in `pending_approval` for **2 days** without admin action, it is automatically cancelled and the blocked funds are released.

Partial Refunds

You can issue multiple partial refunds against the same transaction.

- Specify `amount` to refund less than the full transaction amount.
- The system tracks cumulative `refundedAmount` on the transaction.
- Each completed refund adds to the running total.
- Validation: `amount` must be ≤ `originalAmount - refundedAmount`.
- When cumulative refunds equal the transaction amount, the transaction becomes `refunded` and no more refunds are accepted.

Example: three partial refunds on a 1000 INR transaction

Refund	Amount	Transaction `refundedAmount`	Transaction Status
1st	300	300	`partial_refund`
2nd	200	500	`partial_refund`
3rd	500	1000	`refunded`
4th (rejected)	1	--	Error: already fully refunded

Info

Transactions in `partial_refund` status still accept new refund requests. You do not need to wait for a pending refund to complete before creating the next one -- the system validates against cumulative **completed** refunds only.

Code Examples

CURL

```
# Full refund
curl -X POST https://api.zenpays.com/merchant/api/v1/refund-intents \
  -H "Authorization: Bearer zp_live_abc123def456" \
  -H "X-Timestamp: 2026-04-10T14:22:01.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Idempotency-Key: refund-order-8842" \
  -d '{
    "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
    "reason": "Customer requested full refund",
    "beneficiaryEmail": "rahul@example.com",
    "beneficiaryAccount": "50100012345678",
    "beneficiaryIfsc": "HDFC0001234"
  }'

# Partial refund
curl -X POST https://api.zenpays.com/merchant/api/v1/refund-intents \
  -H "Authorization: Bearer zp_live_abc123def456" \
  -H "X-Timestamp: 2026-04-10T14:35:00.000Z" \
  -H "X-Signature: e5f6a7b8c9d0..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "transactionId": "txn_1774700012000_1_abc12345_9z8y7x",
    "amount": 500,
    "reason": "Partial item return",
    "beneficiaryEmail": "priya@example.com",
    "beneficiaryAccount": "91020034567890",
    "beneficiaryIfsc": "SBIN0001234",
    "beneficiaryName": "Priya Sharma"
  }'
```

JAVASCRIPT

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

const apiKey = process.env.ZENPAYS_API_KEY;
const secretSalt = process.env.ZENPAYS_SECRET_SALT;
const timestamp = new Date().toISOString();

// Full refund
const body = {
  transactionId: 'txn_1774619159530_1_mn8yfmph_5iwb9b',
  reason: 'Customer requested full refund',
  beneficiaryEmail: 'rahul@example.com',
  beneficiaryAccount: '50100012345678',
  beneficiaryIfsc: 'HDFC0001234',
};

const signature = crypto
  .createHmac('sha256', secretSalt)
  .update(timestamp + JSON.stringify(body))
  .digest('hex');

const response = await fetch('https://api.zenpays.com/merchant/api/v1/refund-intents', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Idempotency-Key': 'refund-order-8842',
  },
  body: JSON.stringify(body),
});

const result = await response.json();
console.log(result.data.refundId);  // refund_1774683026458_oann5ibjf
console.log(result.data.status);    // pending_approval
```

PYTHON

```python
import hmac
import hashlib
import json
import os
from datetime import datetime, timezone
import requests

api_key = os.environ["ZENPAYS_API_KEY"]
secret_salt = os.environ["ZENPAYS_SECRET_SALT"]
timestamp = datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%S.000Z")

# Full refund
body = {
    "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
    "reason": "Customer requested full refund",
    "beneficiaryEmail": "rahul@example.com",
    "beneficiaryAccount": "50100012345678",
    "beneficiaryIfsc": "HDFC0001234",
}

data = timestamp + json.dumps(body, separators=(",", ":"))
signature = hmac.new(
    secret_salt.encode(),
    data.encode(),
    hashlib.sha256
).hexdigest()

response = requests.post(
    "https://api.zenpays.com/merchant/api/v1/refund-intents",
    headers={
        "Authorization": f"Bearer {api_key}",
        "X-Timestamp": timestamp,
        "X-Signature": signature,
        "X-Secret-Salt": secret_salt,
        "Content-Type": "application/json",
        "X-Idempotency-Key": "refund-order-8842",
    },
    json=body,
)

result = response.json()
print(result["data"]["refundId"])  # refund_1774683026458_oann5ibjf
print(result["data"]["status"])    # pending_approval
```

SDK (RECOMMENDED)

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({ apiKey: process.env.ZENPAYS_API_KEY })

// Full refund
const refund = await zenpays.refunds.create({
  transactionId: 'txn_1774619159530_1_mn8yfmph_5iwb9b',
  reason: 'Customer requested full refund',
  beneficiaryEmail: 'rahul@example.com',
  beneficiaryAccount: '50100012345678',
  beneficiaryIfsc: 'HDFC0001234',
})
console.log(refund.refundId) // refund_xxxxx

// Partial refund
const partialRefund = await zenpays.refunds.create({
  transactionId: 'txn_1774700012000_1_abc12345_9z8y7x',
  amount: 500,
  reason: 'Partial item return',
  beneficiaryEmail: 'priya@example.com',
  beneficiaryAccount: '91020034567890',
  beneficiaryIfsc: 'SBIN0001234',
  beneficiaryName: 'Priya Sharma',
})
```

Related Endpoints

- Get Refund -- Check refund status
- List Refunds -- List all refunds with filters
- Cancel Refund -- Cancel a pending refund
- Refund Stats -- Aggregate refund statistics
- Bulk Upload -- Create multiple refunds at once

REST API / REFUNDS

Get Refund

Retrieve details of a specific refund by ID.

`GET` **`/merchant/api/v1/refund-intents/{refundId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`refundId`	`string`	Yes	Refund ID (e.g., `ref_xxxxx`)

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "refundId": "refund_1774683026458_oann5ibjf",
    "merchantId": "ZP_FIN_1774592804_0781001264",
    "customerId": "cust_mn8yfewm_37ntht",
    "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
    "paymentIntentId": "pi_1774619159530_66d8s8v25",
    "amount": 30,
    "originalAmount": 100,
    "currency": "INR",
    "refundType": "partial",
    "status": "completed",
    "tspProvider": "sulifu_pay",
    "externalRefundId": "ext_ref_123",
    "reason": "Customer requested refund",
    "createdAt": "2026-03-28T07:30:26.479Z",
    "completedAt": "2026-03-28T07:35:12.000Z",
    "failureReason": null,
    "processingTimeMs": 285633
  }
}
```

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/refund-intents/ref_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const refund = await zenpays.refunds.get('ref_xxxxx');
console.log(`Status: ${refund.status}`);
```

REST API / REFUNDS

List Refunds

Retrieve a paginated list of refunds with optional filters.

`GET` **`/merchant/api/v1/refund-intents`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page (max 100)
`customerId`	`string`	`-`	Filter by customer
`transactionId`	`string`	`-`	Filter by transaction
`status`	`string`	`-`	Filter by status
`startDate`	`string`	`-`	Start date (ISO 8601)
`endDate`	`string`	`-`	End date (ISO 8601)
`currency`	`string`	`-`	Filter by currency

Refund Status Values

Status	Description
`requires_beneficiary`	Beneficiary details needed — re-submit with required fields
`pending_approval`	Awaiting admin approval
`approved`	Approved, being sent to provider
`processing`	Being processed by payment provider
`completed`	Refund completed successfully
`failed`	Refund failed — check `failureReason`
`rejected`	Rejected by admin
`cancelled`	Cancelled before processing

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "refundId": "refund_1774683026458_oann5ibjf",
      "merchantId": "ZP_FIN_1774592804_0781001264",
      "customerId": "cust_mn8yfewm_37ntht",
      "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
      "paymentIntentId": "pi_1774619159530_66d8s8v25",
      "amount": 30,
      "originalAmount": 100,
      "currency": "INR",
      "refundType": "partial",
      "status": "pending_approval",
      "tspProvider": "sulifu_pay",
      "externalRefundId": null,
      "reason": "Customer requested refund",
      "createdAt": "2026-03-28T07:30:26.479Z",
      "completedAt": null,
      "failureReason": null,
      "processingTimeMs": null
    }
  ],
  "metadata": {
    "pagination": {
      "page": 1,
      "limit": 20,
      "total": 45,
      "totalPages": 3
    }
  }
}
```

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/refund-intents?status=completed&limit=50" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

JAVASCRIPT

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

const apiKey = process.env.ZENPAYS_API_KEY;
const secretSalt = process.env.ZENPAYS_SECRET_SALT;
const timestamp = new Date().toISOString();

const signature = crypto
  .createHmac('sha256', secretSalt)
  .update(timestamp + '{}')
  .digest('hex');

const params = new URLSearchParams({
  status: 'completed',
  limit: '50',
});

const response = await fetch(
  `https://api.zenpays.com/merchant/api/v1/refund-intents?${params}`,
  {
    method: 'GET',
    headers: {
      'Authorization': `Bearer ${apiKey}`,
      'X-Timestamp': timestamp,
      'X-Signature': signature,
      'X-Secret-Salt': secretSalt,
    },
  }
);

const result = await response.json();
console.log(`Found ${result.pagination.total} refunds`);
```

SDK (RECOMMENDED)

```
const { data, total } = await zenpays.refunds.list({
  status: 'completed',
  limit: 50,
});

console.log(`Found ${total} refunds`);
```

Related Endpoints

- Confirm Refund Intent
- Get Refund
- Refund Statistics

REST API / REFUNDS

Cancel Refund

Cancel a pending refund.

`DELETE` **`/merchant/api/v1/refund-intents/{refundId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`refundId`	`string`	Yes	Refund ID

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "refundId": "refund_1774683026458_oann5ibjf",
    "merchantId": "ZP_FIN_1774592804_0781001264",
    "customerId": "cust_mn8yfewm_37ntht",
    "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
    "paymentIntentId": "pi_1774619159530_66d8s8v25",
    "amount": 30,
    "originalAmount": 100,
    "currency": "INR",
    "refundType": "partial",
    "status": "cancelled",
    "tspProvider": "sulifu_pay",
    "externalRefundId": null,
    "reason": "Customer requested refund",
    "createdAt": "2026-03-28T07:30:26.479Z",
    "completedAt": null,
    "failureReason": null,
    "processingTimeMs": null
  }
}
```

Error Responses

Code	HTTP	Message
`REFUND_NOT_CANCELLABLE`	400	Refund cannot be cancelled
`REFUND_NOT_FOUND`	404	Refund doesn't exist

Examples

CURL

```
curl -X DELETE "https://api.zenpays.com/merchant/api/v1/refund-intents/ref_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
await zenpays.refunds.cancel('ref_xxxxx');
```

REST API / REFUNDS

Refund Statistics

Get refund statistics and metrics.

`GET` **`/merchant/api/v1/refund-intents/stats`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`startDate`	`string`	`30 days ago`	Start date (ISO format)
`endDate`	`string`	`now`	End date (ISO format)
`currency`	`string`	`-`	Filter by currency code

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "total": 150,
    "pending": 5,
    "approved": 3,
    "rejected": 1,
    "processing": 2,
    "completed": 135,
    "failed": 3,
    "cancelled": 1,
    "totalAmount": 250000,
    "averageAmount": 1666.67,
    "averageProcessingTime": 285633,
    "refundRate": 90
  }
}
```

Response Fields

Parameter	Type	Description
`total`	`number`	Total number of refunds
`pending`	`number`	Refunds pending approval
`approved`	`number`	Approved, awaiting processing
`rejected`	`number`	Rejected by admin
`processing`	`number`	Currently being processed by TSP
`completed`	`number`	Successfully completed refunds
`failed`	`number`	Failed refunds
`cancelled`	`number`	Cancelled refunds
`totalAmount`	`number`	Sum of all refund amounts
`averageAmount`	`number`	Average refund amount
`averageProcessingTime`	`number`	Average processing time in milliseconds
`refundRate`	`number`	Percentage of completed refunds (completed / total * 100)

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/refund-intents/stats?startDate=2026-01-01&endDate=2026-03-28" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2026-03-28T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const stats = await zenpays.refunds.getStats('2026-01-01', '2026-03-28');
console.log(`Total: ${stats.total}, Completed: ${stats.completed}`);
console.log(`Refund rate: ${stats.refundRate}%`);
console.log(`Total amount: ${stats.totalAmount}`);
```

Related Endpoints

- List Refunds — Get individual refund records
- Get Refund — Get details of a specific refund

Bulk Refunds

REST API / REFUNDS / BULK REFUNDS

Bulk Upload Refunds

Upload multiple refunds in a single API call. Supports all the same beneficiary fields as the Confirm Refund Intent endpoint. Use Create Refund Intent (https://docs.zenpays.com/docs/rest-api/endpoints/refunds/get-template) to discover the required fields first.

`POST` **`/merchant/api/v1/refund-intents/bulk-upload`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`refunds`	`array`	Yes	Array of refund requests
`refunds[].transactionId`	`string`	Yes	Original transaction ID
`refunds[].amount`	`number`	No	Refund amount. Omit for full refund.
`refunds[].reason`	`string`	No	Reason for refund
`refunds[].customerId`	`string`	No	Customer ID (resolved from transaction if omitted)
`refunds[].beneficiaryVpa`	`string`	No	UPI VPA for UPI refunds
`refunds[].beneficiaryAccount`	`string`	No	Bank account number
`refunds[].beneficiaryIfsc`	`string`	No	IFSC code (India)
`refunds[].beneficiaryName`	`string`	No	Beneficiary name
`refunds[].beneficiaryMobile`	`string`	No	Mobile number
`refunds[].beneficiaryEmail`	`string`	No	Beneficiary email address

Parameter	Type	Required	Description
`refunds[].beneficiaryCity`	`string`	No	Beneficiary city
`refunds[].beneficiaryWalletAddress`	`string`	No	Crypto wallet address (crypto refunds)
`refunds[].beneficiaryWalletMemo`	`string`	No	Wallet memo/tag for XRP/XLM
`refunds[].chain`	`string`	No	Blockchain chain (ethereum, tron, bitcoin)

Which beneficiary fields do I need?

Call Create Refund Intent (https://docs.zenpays.com/docs/rest-api/endpoints/refunds/get-template) for any transaction to discover the required fields. You can also use the template response to generate a CSV file for bulk uploads — the `fieldName` values in `requiredFields`, `optionalFields`, and `oneOfGroups` map directly to column headers.

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "successful": [
      {
        "refundId": "refund_1774683026458_oann5ibjf",
        "merchantId": "ZP_FIN_1774592804_0781001264",
        "customerId": "cust_mn8yfewm_37ntht",
        "transactionId": "txn_1774619159530_1_mn8yfmph_5iwb9b",
        "amount": 30,
        "originalAmount": 100,
        "currency": "INR",
        "refundType": "partial",
        "status": "pending_approval",
        "tspProvider": "sulifu_pay",
        "reason": "Customer requested refund",
        "createdAt": "2026-03-28T07:30:26.479Z"
      }
    ],
    "failed": [
      {
        "transactionId": "txn_invalid",
        "error": "Transaction not found"
      }
    ],
    "totalRequested": 2,
    "totalSuccessful": 1,
    "totalFailed": 1
  }
}
```

Requires Beneficiary Details

If any refund in the batch requires beneficiary details that were not provided, that item will appear in the `successful` array with `requiresBeneficiary: true` and field definitions. Re-submit that item with the required fields.

```
{
  "requiresBeneficiary": true,
  "status": "requires_beneficiary",
  "tspProvider": "sulifu_pay",
  "currency": "INR",
  "transactionId": "txn_xxxxx",
  "requiredFields": [...],
  "optionalFields": [...],
  "oneOfGroups": [["beneficiaryVpa"], ["beneficiaryAccount", "beneficiaryIfsc"]]
}
```

Avoid requires_beneficiary responses

To avoid this, call Create Refund Intent (https://docs.zenpays.com/docs/rest-api/endpoints/refunds/get-template) first for a sample transaction and include the required beneficiary fields in every batch item.

CSV Upload Workflow

For bulk refunds, you can use a CSV file workflow:

1. **Create refund intent**: Call `POST /refund-intents` with a sample `transactionId` to discover required fields
2. **Build CSV**: Use `fieldName` values from `requiredFields`, `optionalFields`, and `oneOfGroups` as column headers, alongside base columns (`transactionId`, `amount`, `reason`)
3. **Fill CSV**: Populate rows with refund data — one row per refund
4. **Submit**: Parse the CSV and POST the rows as a JSON array to `POST /refund-intents/bulk-upload`

Example CSV

```
transactionId,amount,reason,beneficiaryEmail,beneficiaryVpa,beneficiaryAccount,beneficiaryIfsc,benefi
ciaryName,beneficiaryMobile,beneficiaryCity
txn_aaa,20,Customer request,john@example.com,john@upi,,,John Doe,+919876543210,Mumbai
txn_bbb,500,Duplicate charge,alice@example.com,,1234567890,HDFC0001234,Alice
Smith,+919123456789,Delhi
txn_ccc,250,Item not received,bob@example.com,bob@ybl,,,Bob Johnson,+918765432109,Bangalore
```

Column headers must match field names exactly

The CSV column headers must match the `fieldName` values from the template response (e.g., `beneficiaryVpa`, `beneficiaryAccount`, `beneficiaryIfsc`). Leave columns empty for fields you don't need — for example, if using UPI (VPA), leave `beneficiaryAccount` and `beneficiaryIfsc` empty.

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/refund-intents/bulk-upload \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "refunds": [
      {
        "transactionId": "txn_aaaaa",
        "amount": 500,
        "reason": "Customer request",
        "beneficiaryEmail": "john@example.com",
        "beneficiaryVpa": "john@upi",
        "beneficiaryName": "John Doe",
        "beneficiaryMobile": "+919876543210"
      },
      {
        "transactionId": "txn_bbbbb",
        "amount": 1000,
        "reason": "Duplicate charge",
        "beneficiaryEmail": "alice@example.com",
        "beneficiaryAccount": "1234567890",
        "beneficiaryIfsc": "HDFC0001234",
        "beneficiaryName": "Alice Smith"
      }
    ]
  }'
```

SDK (RECOMMENDED)

```
const result = await zenpays.refunds.bulkUpload({
  refunds: [
    {
      transactionId: 'txn_aaaaa',
      amount: 500,
      reason: 'Customer request',
      beneficiaryEmail: 'john@example.com',
      beneficiaryVpa: 'john@upi',
      beneficiaryName: 'John Doe',
    },
    {
      transactionId: 'txn_bbbbb',
      amount: 1000,
      reason: 'Duplicate charge',
      beneficiaryEmail: 'alice@example.com',
      beneficiaryAccount: '1234567890',
      beneficiaryIfsc: 'HDFC0001234',
      beneficiaryName: 'Alice Smith',
    },
  ],
});

console.log(`Succeeded: ${result.totalSuccessful}, Failed: ${result.totalFailed}`);
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request body
`UNAUTHORIZED`	401	Invalid API key

Individual item errors are returned in the `failed` array, not as HTTP errors.

Related Endpoints

- Create Refund Intent (https://docs.zenpays.com/docs/rest-api/endpoints/refunds/get-template) — Discover required fields before batch upload
- Confirm Refund Intent — Confirm a single refund
- List Refunds
- Get Refund

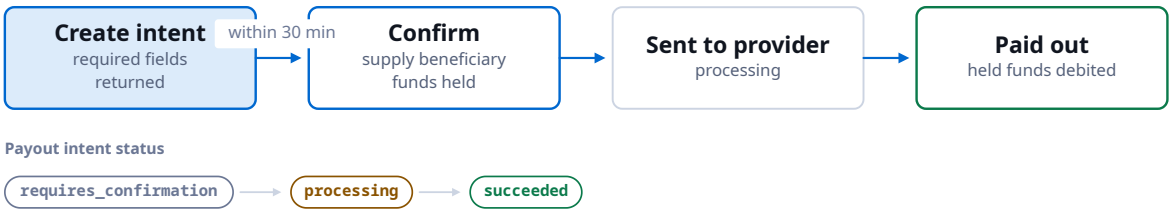
Payout Intents

REST API / PAYOUT INTENTS

Create Payout Intent

Create a payout intent. Returns the fields you need to collect before confirming the payout.

A payout intent tells you exactly which beneficiary fields that corridor needs before you commit.



Payout intent status

WEBHOOKS YOU RECEIVE

payout_intent.created | payout_intent.confirmed | payout.initiated | payout.completed

An intent left unconfirmed for 30 minutes expires, and nothing is held or debited.

How a payout is confirmed, sent and debited

POST `/payment/api/v1/payout-intents`

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Idempotency-Key`	No	Unique key to prevent duplicates

Body Parameters

Parameter	Type	Required	Description
`amount`	`number`	Yes	Payout amount (must be > 0)
`currency`	`string`	Yes	Currency code (e.g. `USD`, `INR`, `USDT`)

Parameter	Type	Required	Description
`country`	`string`	No	ISO 3166-1 alpha-2 country code
`beneficiaryName`	`string`	Yes	Beneficiary full name
`beneficiaryEmail`	`string`	No	Beneficiary email
`beneficiaryPhone`	`string`	No	Beneficiary phone
`payoutType`	`string`	No	`bank_transfer`, `upi`, `imps`, `neft`, `rtgs`
`purpose`	`string`	No	Purpose (e.g. `PAYOUT`, `SALARY`, `REFUND`)
`description`	`string`	No	Description
`customerId`	`string`	No	Your customer ID
`webhookUrl`	`string`	No	URL for payout status webhooks
`metadata`	`object`	No	Custom key-value pairs (e.g. `{ "order_no": "ORD-001" }`). Echoed back in all webhook callbacks for this payout.

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "intentId": "poi_xxxxx",
    "status": "requires_confirmation",
    "amount": 10000,
    "currency": "INR",
    "beneficiaryName": "John Doe",
    "requiredFields": [
      {
        "fieldName": "beneficiaryAccount",
        "label": "Account Number",
        "type": "text",
        "required": true,
        "placeholder": "Bank account number"
      },
      {
        "fieldName": "beneficiaryIfsc",
        "label": "IFSC Code",
        "type": "text",
        "required": true,
        "placeholder": "e.g. HDFC0001234"
      }
    ],
    "optionalFields": [
      {
        "fieldName": "beneficiaryEmail",
        "label": "Email",
        "type": "text",
        "required": false
      }
    ],
    "expiresAt": "2024-01-15T11:00:00.000Z",
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```

For crypto payouts (e.g. `currency: "USDT"`), the required fields will be:

```
{
  "requiredFields": [
    {
      "fieldName": "chain",
      "label": "Network / Chain",
      "type": "select",
      "required": true,
      "options": ["ethereum", "tron", "bitcoin", "solana", "polygon"]
    },
    {
      "fieldName": "walletAddress",
      "label": "Wallet Address",
      "type": "text",
      "required": true
    }
  ]
}
```

Error Responses

Code	HTTP	Message
`PAYOUT_INTENT_CREATE_FAILED`	400	Invalid request or insufficient wallet balance
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X POST https://api.zenpays.com/payment/api/v1/payout-intents \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "amount": 10000,
    "currency": "INR",
    "country": "IN",
    "beneficiaryName": "John Doe",
    "beneficiaryEmail": "john@example.com",
    "purpose": "PAYOUT"
  }'
```

SDK

```
const intent = await zenpays.payouts.createPayoutIntent({
  amount: 10000,
  currency: 'INR',
  country: 'IN',
  beneficiaryName: 'John Doe',
  beneficiaryEmail: 'john@example.com',
  purpose: 'PAYOUT',
})

// Now collect the required fields from intent.requiredFields
console.log(intent.requiredFields)
```

How It Works

1. You send basic payout details (amount, currency, beneficiary name)
2. ZenPays returns the **exact fields** you need to collect
3. You collect those fields from your user or your system
4. You confirm the intent with those fields
5. The payout is executed

The intent expires after **30 minutes**. If not confirmed, create a new one.

Related Endpoints

- Confirm Payout Intent
- Get Payout Intent

- List Payout Intents

REST API / PAYOUT INTENTS

Confirm Payout Intent

Submit the required fields to confirm and execute a payout intent.

`POST` **`/payment/api/v1/payout-intents/{intentId}/confirm`**

Request

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Payout intent ID

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

The body should contain the fields listed in the `requiredFields` array from the create intent response. **Always check `requiredFields`** — the exact fields vary by currency, country, and the TSP selected by smart routing.

> **Always use requiredFields**
> When you create a payout intent, the response includes a `requiredFields` array that tells you exactly which fields are needed to confirm. Some fields may be required for one currency but optional for another. Never hardcode assumptions — always read `requiredFields` from the create response.

Fiat Payout Fields

Parameter	Type	Description
`beneficiaryAccount`	`string`	Bank account number (5-34 characters)
`beneficiaryEmail`	`string`	Beneficiary email address
`beneficiaryBankCode`	`string`	Bank code in Sulifu format (e.g. `dp_axisbank_in`, `dp_hdfcbank_in`, `dp_sboi_in`)

Parameter	Type	Description
`beneficiaryIfsc`	`string`	IFSC code for Indian banks (e.g. `HDFC0001234`)
`beneficiaryName`	`string`	Full name of the beneficiary
`beneficiaryMobile`	`string`	Beneficiary phone number with country code
`beneficiaryAddress`	`string`	Street address
`beneficiaryCity`	`string`	City
`beneficiaryState`	`string`	State or province
`beneficiaryPostalCode`	`string`	Postal/ZIP code
`beneficiaryAccountType`	`string`	Account type: `CHECKING` or `SAVINGS`
`beneficiaryDocumentId`	`string`	Government-issued document ID (required in some regions)

Crypto Payout Fields

Parameter	Type	Description
`chain`	`string`	Blockchain network: `tron`, `ethereum`, `bitcoin`, `solana`, etc.
`walletAddress`	`string`	Destination wallet address
`walletMemo`	`string`	Memo or destination tag (for XRP, XLM, etc.)

Common Optional Fields

Parameter	Type	Description
`customerId`	`string`	Customer ID for tracking
`customerEmail`	`string`	Customer email
`customerPhone`	`string`	Customer phone number

Example: INR Bank Transfer

For INR payouts, the following fields are typically required:

```
{
  "beneficiaryAccount": "1234567890",
  "beneficiaryEmail": "rajesh@example.com",
  "beneficiaryBankCode": "dp_axisbank_in",
  "beneficiaryAddress": "123 Main Street",
  "beneficiaryCity": "Mumbai",
  "beneficiaryState": "Maharashtra",
  "beneficiaryPostalCode": "400001",
  "beneficiaryName": "Rajesh Kumar",
  "beneficiaryMobile": "+919876543210",
  "beneficiaryIfsc": "HDFC0001234"
}
```

Example: Crypto (USDT) Payout

```
{
  "chain": "tron",
  "walletAddress": "TXyz123abc456def789ghi"
}
```

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "intentId": "poi_mnblh1o5_e8190e32",
    "status": "processing",
    "payoutId": "payout_1774778823937_17298d89",
    "externalPayoutId": "po6efbc8502da51774842226226",
    "amount": "100.00000000",
    "currency": "INR",
    "processingFee": "0.00000000",
    "totalDebitedAmount": "100.00000000",
    "confirmedAt": "2026-03-29T10:07:05.144Z"
  },
  "message": "Payout intent confirmed",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2026-03-29T10:07:05.148Z",
    "processing_time_ms": 1515,
    "api_version": "v1",
    "endpoint": "POST /payout-intents/{intentId}/confirm",
    "user_type": "merchant"
  }
}
```

Response Fields

Parameter	Type	Description
`intentId`	`string`	The payout intent ID
`status`	`string`	Intent status (`processing`)
`payoutId`	`string`	Internal payout ID for tracking
`externalPayoutId`	`string`	TSP transaction ID — use this to match with webhook callbacks
`amount`	`string`	Payout amount
`currency`	`string`	Payout currency
`processingFee`	`string`	Processing fee charged
`totalDebitedAmount`	`string`	Total amount debited from wallet (amount + fee)
`confirmedAt`	`string`	ISO 8601 timestamp of confirmation

Linking confirm response to webhooks

The `externalPayoutId` in the confirm response matches the `transactionId` in webhook callbacks. Use this to link the payout intent confirmation to the webhook status updates:

- **Confirm response**: `intentId` + `payoutId` + `externalPayoutId`
- **Webhook callback**: `intentId` + `payoutId` + `transactionId` (same as `externalPayoutId`)

Error Responses

Code	HTTP	Message
`PAYOUT_INTENT_CONFIRM_FAILED`	400	Missing required fields, expired intent, or insufficient wallet balance
`VALIDATION_ERROR`	400	Invalid field values (e.g. unsupported bank code)
`NOT_FOUND`	404	Intent not found or does not belong to this merchant

Common Errors

- **"Required bank_code in extra field"** — You're missing `beneficiaryBankCode` in your confirmation payload. Check `requiredFields` from the create response.
- **"Beneficiary address fields required"** — For non-USDT payouts, address fields (`beneficiaryAddress` , `beneficiaryCity` , `beneficiaryState` , `beneficiaryPostalCode`) are mandatory.
- **"Unsupported bank code"** — Use the ZenPays bank code format: `dp_{bankname}_in` (e.g. `dp_hdfcbank_in` , `dp_axisbank_in` , `dp_sboi_in`). The available codes are listed in the `options` array of the `beneficiaryBankCode` field in `requiredFields` .

Examples

CURL

```
curl -X POST "https://api.zenpays.com/payment/api/v1/payout-intents/poi_mnblh1o5_e8190e32/confirm"
\
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2026-03-29T10:07:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "beneficiaryAccount": "1234567890",
    "beneficiaryEmail": "rajesh@example.com",
    "beneficiaryBankCode": "dp_axisbank_in",
    "beneficiaryAddress": "123 Main Street",
    "beneficiaryCity": "Mumbai",
    "beneficiaryState": "Maharashtra",
    "beneficiaryPostalCode": "400001",
    "beneficiaryName": "Rajesh Kumar",
    "beneficiaryMobile": "+919876543210",
    "beneficiaryIfsc": "HDFC0001234"
  }'
```

JAVASCRIPT

```
const response = await fetch(
  'https://api.zenpays.com/payment/api/v1/payout-intents/poi_mnblh1o5_e8190e32/confirm',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${apiKey}`,
      'X-Timestamp': timestamp,
      'X-Signature': signature,
      'X-Secret-Salt': secretSalt,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      beneficiaryAccount: '1234567890',
      beneficiaryEmail: 'rajesh@example.com',
      beneficiaryBankCode: 'dp_axisbank_in',
      beneficiaryAddress: '123 Main Street',
      beneficiaryCity: 'Mumbai',
      beneficiaryState: 'Maharashtra',
      beneficiaryPostalCode: '400001',
      beneficiaryName: 'Rajesh Kumar',
      beneficiaryMobile: '+919876543210',
      beneficiaryIfsc: 'HDFC0001234',
    }),
  }
);

const result = await response.json();
console.log(result.data.payoutId); // 'payout_xxxxx'
console.log(result.data.status);   // 'processing'
```

SDK

```
const result = await zenpays.payouts.confirmPayoutIntent('poi_mnblh1o5_e8190e32', {
  beneficiaryAccount: '1234567890',
  beneficiaryEmail: 'rajesh@example.com',
  beneficiaryBankCode: 'dp_axisbank_in',
  beneficiaryAddress: '123 Main Street',
  beneficiaryCity: 'Mumbai',
  beneficiaryState: 'Maharashtra',
  beneficiaryPostalCode: '400001',
  beneficiaryName: 'Rajesh Kumar',
  beneficiaryMobile: '+919876543210',
  beneficiaryIfsc: 'HDFC0001234',
});

console.log(result.payoutId); // 'payout_xxxxx'
console.log(result.status);   // 'processing'
```

Related Endpoints

- Create Payout Intent
- Get Payout Intent
- List Payout Intents
- Cancel Payout Intent

REST API / PAYOUT INTENTS

Get Payout Intent

Retrieve the current status and details of a payout intent.

`GET` **`/payment/api/v1/payout-intents/{intentId}`**

Request

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Payout intent ID

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "intentId": "poi_xxxxx",
    "status": "succeeded",
    "amount": 10000,
    "currency": "INR",
    "beneficiaryName": "John Doe",
    "payoutId": "payout_xxxxx",
    "processingFee": 25,
    "totalDebitedAmount": 10025,
    "expiresAt": "2024-01-15T11:00:00.000Z",
    "confirmedAt": "2024-01-15T10:35:00.000Z",
    "completedAt": "2024-01-15T10:36:00.000Z",
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/payout-intents/poi_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const intent = await zenpays.payouts.getPayoutIntent('poi_xxxxx')
console.log(intent.status) // 'succeeded'
```

Intent Statuses

Status	Description
`requires_confirmation`	Waiting for required fields to be submitted
`processing`	Payout is being processed
`succeeded`	Payout completed successfully
`failed`	Payout failed (see `failureReason`)
`cancelled`	Cancelled before confirmation
`expired`	Not confirmed within 30 minutes

Related Endpoints

- Create Payout Intent
- List Payout Intents

REST API / PAYOUT INTENTS

List Payout Intents

List payout intents with optional filters and pagination.

`GET` **`/payment/api/v1/payout-intents`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Description
`status`	`string`	Filter by status
`page`	`number`	Page number (default: 1)
`limit`	`number`	Items per page (default: 20)
`fromDate`	`string`	Start date (ISO format)
`toDate`	`string`	End date (ISO format)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "data": [
      {
        "intentId": "poi_xxxxx",
        "status": "succeeded",
        "amount": 10000,
        "currency": "INR",
        "beneficiaryName": "John Doe",
        "payoutId": "payout_xxxxx",
        "createdAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "total": 25,
    "page": 1,
    "limit": 20,
    "totalPages": 2,
    "hasMore": true
  }
}
```


Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/payout-intents?status=succeeded&limit=10" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const { data, total } = await zenpays.payouts.listPayoutIntents({
  status: 'succeeded',
  limit: 10,
})
```

Related Endpoints

- Create Payout Intent
- Get Payout Intent

REST API / PAYOUT INTENTS

Cancel Payout Intent

Cancel a payout intent that has not yet been confirmed.

`POST` **`/payment/api/v1/payout-intents/{intentId}/cancel`**

Request

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Payout intent ID

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Description
`reason`	`string`	Cancellation reason

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "intentId": "poi_xxxxx",
    "status": "cancelled",
    "message": "Payout intent cancelled"
  }
}
```

Error Responses

Code	HTTP	Message
`PAYOUT_INTENT_CANCEL_FAILED`	400	Intent already confirmed or in terminal state
`NOT_FOUND`	404	Intent not found

Examples

CURL

```
curl -X POST "https://api.zenpays.com/payment/api/v1/payout-intents/poi_xxxxx/cancel" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{"reason": "No longer needed"}'
```

SDK

```
await zenpays.payouts.cancelPayoutIntent('poi_xxxxx', 'No longer needed')
```

Related Endpoints

- Create Payout Intent
- Get Payout Intent

Settlements

REST API / SETTLEMENTS

Create Settlement

Create a settlement request to withdraw funds to your bank account.

`POST` **`/merchant/api/v1/settlements/create`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`amount`	`number`	Yes	Settlement amount
`bankAccountId`	`string`	Yes	Bank account ID to settle to
`description`	`string`	No	Description for the settlement
`priority`	`string`	No	Settlement priority
`type`	`string`	No	Settlement type
`scheduledDate`	`string`	No	ISO 8601 date for scheduled settlement

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "settlementId": "stl_xxxxx",
    "merchantId": "mer_xxxxx",
    "amount": 100000,
    "settlementFee": 500,
    "netAmount": 99500,
    "currency": "USD",
    "type": "standard",
    "priority": "normal",
    "status": "PENDING",
    "description": "Monthly settlement",
    "bankAccountId": "ba_xxxxx",
    "scheduledDate": null,
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```


Response Fields

Parameter	Type	Description
`settlementId`	`string`	Unique settlement identifier
`merchantId`	`string`	Merchant identifier
`amount`	`number`	Requested settlement amount
`settlementFee`	`number`	Fee charged for the settlement
`netAmount`	`number`	Amount after fees
`currency`	`string`	ISO 4217 currency code
`type`	`string`	Settlement type
`priority`	`string`	Settlement priority
`status`	`string`	Settlement status
`description`	`string`	Settlement description
`bankAccountId`	`string`	Target bank account
`scheduledDate`	`string`	Scheduled settlement date, if any
`createdAt`	`string`	ISO 8601 creation timestamp

Error Responses

Code	HTTP	Message
`INVALID_AMOUNT`	400	Amount below minimum
`UNAUTHORIZED`	401	Invalid API key
`INSUFFICIENT_BALANCE`	402	Wallet balance too low
`BANK_ACCOUNT_NOT_FOUND`	404	Bank account doesn't exist

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/settlements/create \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "amount": 100000,
    "bankAccountId": "ba_xxxxx",
    "description": "Monthly settlement",
    "priority": "normal",
    "type": "standard"
  }'
```

SDK

```
const settlement = await zenpays.settlements.create({
  amount: 100000,
  bankAccountId: 'ba_xxxxx',
  description: 'Monthly settlement',
  priority: 'normal',
  type: 'standard',
});
```

Settlement Status

Status	Description
`PENDING`	Settlement created, awaiting processing
`PROCESSING`	Being processed
`COMPLETED`	Funds transferred to bank
`FAILED`	Settlement failed
`CANCELLED`	Settlement was cancelled

REST API / SETTLEMENTS

Get Settlement

Retrieve details of a specific settlement.

`GET` **`/merchant/api/v1/settlements/{id}`**

Request

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Settlement ID

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "settlementId": "stl_xxxxx",
    "merchantId": "mer_xxxxx",
    "amount": 100000,
    "settlementFee": 500,
    "netAmount": 99500,
    "currency": "USD",
    "type": "standard",
    "priority": "normal",
    "status": "COMPLETED",
    "description": "Monthly settlement",
    "bankAccountId": "ba_xxxxx",
    "scheduledDate": null,
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```

Response Fields

Parameter	Type	Description
`settlementId`	`string`	Unique settlement identifier
`merchantId`	`string`	Merchant identifier
`amount`	`number`	Requested settlement amount
`settlementFee`	`number`	Fee charged for the settlement
`netAmount`	`number`	Amount after fees
`currency`	`string`	ISO 4217 currency code
`type`	`string`	Settlement type
`priority`	`string`	Settlement priority
`status`	`string`	Settlement status
`description`	`string`	Settlement description
`bankAccountId`	`string`	Target bank account
`scheduledDate`	`string`	Scheduled settlement date, if any
`createdAt`	`string`	ISO 8601 creation timestamp

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/settlements/stl_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const settlement = await zenpays.settlements.get('stl_xxxxx');
```

REST API / SETTLEMENTS

List Settlements

Retrieve a paginated list of settlements.

`GET` **`/merchant/api/v1/settlements/list`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page
`startDate`	`string`	`-`	Start date filter (ISO 8601)
`endDate`	`string`	`-`	End date filter (ISO 8601)
`status`	`string`	`-`	Filter by status (`PENDING` , `PROCESSING` , `COMPLETED` , `FAILED` , `CANCELLED`)
`type`	`string`	`-`	Filter by settlement type
`priority`	`string`	`-`	Filter by priority
`bankAccountId`	`string`	`-`	Filter by bank account
`minAmount`	`number`	`-`	Minimum settlement amount
`maxAmount`	`number`	`-`	Maximum settlement amount

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "settlementId": "stl_xxxxx",
      "merchantId": "mer_xxxxx",
      "amount": 100000,
      "settlementFee": 500,
      "netAmount": 99500,
      "currency": "USD",
      "type": "standard",
      "priority": "normal",
      "status": "COMPLETED",
      "description": "Monthly settlement",
      "bankAccountId": "ba_xxxxx",
      "scheduledDate": null,
      "createdAt": "2024-01-15T10:30:00.000Z"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 50,
    "totalPages": 3
  }
}
```


Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/settlements/list?
status=COMPLETED&minAmount=50000" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const { data } = await zenpays.settlements.list({
  status: 'COMPLETED',
  minAmount: 50000,
});
```

REST API / SETTLEMENTS

Cancel Settlement

Cancel a pending settlement request.

`PUT` **`/merchant/api/v1/settlements/{id}/cancel`**

Request

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Settlement ID

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "settlementId": "stl_xxxxx",
    "status": "CANCELLED",
    "message": "Settlement cancelled successfully"
  }
}
```

Error Responses

Code	HTTP	Message
`SETTLEMENT_NOT_CANCELLABLE`	400	Settlement cannot be cancelled (not in `PENDING` status)
`SETTLEMENT_NOT_FOUND`	404	Settlement doesn't exist

Examples

CURL

```
curl -X PUT "https://api.zenpays.com/merchant/api/v1/settlements/stl_xxxxx/cancel" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
await zenpays.settlements.cancel('stl_xxxxx');
```

REST API / SETTLEMENTS

Settlement Statistics

Get settlement statistics and summary.

`GET` **`/merchant/api/v1/settlements/stats/summary`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "totalSettlements": 150,
    "totalSettled": 5000000,
    "pendingAmount": 250000,
    "byStatus": {
      "COMPLETED": 140,
      "PENDING": 8,
      "PROCESSING": 2,
      "FAILED": 0,
      "CANCELLED": 0
    },
    "averageSettlementTime": "1.5 days",
    "lastSettlement": {
      "settlementId": "stl_xxxxx",
      "amount": 100000,
      "status": "COMPLETED",
      "createdAt": "2024-01-17T10:00:00.000Z"
    }
  }
}
```

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/settlements/stats/summary" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const stats = await zenpays.settlements.getStats();
```

Bank Accounts

REST API / BANK ACCOUNTS

Add Bank Account

Register a new bank account for receiving settlement payouts.

`POST` **`/merchant/api/v1/settlements/bank-accounts`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`accountHolderName`	`string`	Yes	Account holder name
`accountNumber`	`string`	Yes	Bank account number
`accountType`	`string`	Yes	`CURRENT` or `SAVINGS`
`ifscCode`	`string`	Yes	IFSC/Routing code
`bankName`	`string`	Yes	Name of the bank
`branchName`	`string`	No	Bank branch name
`branchAddress`	`string`	No	Bank branch address
`isPrimary`	`boolean`	No	Set as primary account
`swiftCode`	`string`	No	SWIFT/BIC code for international transfers

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "id": "ba_xxxxx",
    "accountNumber": "****5678",
    "accountHolderName": "Acme Inc",
    "accountType": "CURRENT",
    "bankName": "Bank of America",
    "branchName": "Main Branch",
    "ifscCode": "BOFA0001234",
    "isPrimary": false,
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid input data
`UNAUTHORIZED`	401	Invalid API key
`DUPLICATE_ACCOUNT`	409	Account number already registered

Examples

CURL

```
curl -X POST "https://api.zenpays.com/merchant/api/v1/settlements/bank-accounts" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "accountHolderName": "Acme Inc",
    "accountNumber": "9876543210",
    "accountType": "CURRENT",
    "ifscCode": "BOFA0001234",
    "bankName": "Bank of America",
    "branchName": "Main Branch",
    "isPrimary": false
  }'
```

SDK

```
const account = await zenpays.settlements.addBankAccount({
  accountHolderName: 'Acme Inc',
  accountNumber: '9876543210',
  accountType: 'CURRENT',
  ifscCode: 'BOFA0001234',
  bankName: 'Bank of America',
  branchName: 'Main Branch',
});

console.log(`Bank account added: ${account.id}`);
```

REST API / BANK ACCOUNTS

List Bank Accounts

Retrieve all registered bank accounts for the merchant.

`GET` **`/merchant/api/v1/settlements/bank-accounts/list`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "id": "ba_xxxxx",
      "accountNumber": "****1234",
      "accountHolderName": "Acme Inc",
      "accountType": "CURRENT",
      "bankName": "Chase Bank",
      "branchName": "Downtown Branch",
      "ifscCode": "CHAS0001234",
      "isPrimary": true,
      "createdAt": "2024-01-01T10:00:00.000Z"
    },
    {
      "id": "ba_yyyyy",
      "accountNumber": "****5678",
      "accountHolderName": "Acme Inc",
      "accountType": "SAVINGS",
      "bankName": "Bank of America",
      "branchName": "Main Branch",
      "ifscCode": "BOFA0001234",
      "isPrimary": false,
      "createdAt": "2024-01-10T14:00:00.000Z"
    }
  ]
}
```

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/settlements/bank-accounts/list" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const accounts = await zenpays.settlements.listBankAccounts();

accounts.forEach(account => {
  console.log(`${account.bankName} - ${account.accountNumber} (${account.isPrimary ? 'Primary' :
'Secondary'})`);
});
```

REST API / BANK ACCOUNTS

Update Bank Account

Update an existing bank account's details. Only non-sensitive fields can be modified.

`PUT` **`/merchant/api/v1/settlements/bank-accounts/{id}`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Bank account ID

Body Parameters

All fields are optional. Only provided fields will be updated.

Parameter	Type	Description
`accountHolderName`	`string`	Account holder name
`branchName`	`string`	Bank branch name
`branchAddress`	`string`	Bank branch address
`isPrimary`	`boolean`	Set as primary account
`swiftCode`	`string`	SWIFT/BIC code

> **Note**
> Sensitive fields like `accountNumber`, `ifscCode`, and `bankName` cannot be modified after creation. To change these, delete and re-add the bank account.

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "id": "ba_xxxxx",
    "accountNumber": "****5678",
    "accountHolderName": "Acme Inc Updated",
    "accountType": "CURRENT",
    "bankName": "Bank of America",
    "branchName": "New Branch",
    "ifscCode": "BOFA0001234",
    "isPrimary": true,
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid input data
`BANK_ACCOUNT_NOT_FOUND`	404	Bank account does not exist

Examples

CURL

```
curl -X PUT "https://api.zenpays.com/merchant/api/v1/settlements/bank-accounts/ba_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "isPrimary": true,
    "branchName": "New Branch"
  }'
```

SDK

```
await zenpays.settlements.updateBankAccount('ba_xxxxx', {
  isPrimary: true,
  branchName: 'New Branch',
});
```

REST API / BANK ACCOUNTS

Delete Bank Account

Remove a bank account from the merchant's registered accounts.

`DELETE` **`/merchant/api/v1/settlements/bank-accounts/{id}`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Bank account ID

> **Warning**
> A primary bank account cannot be deleted. Set another account as primary first before deleting.

Response

Success (200 OK)

```
{
  "success": true,
  "message": "Bank account deleted successfully"
}
```

Error Responses

Code	HTTP	Message
`BANK_ACCOUNT_NOT_FOUND`	404	Bank account does not exist
`CANNOT_DELETE_PRIMARY`	400	Cannot delete primary bank account

Examples

CURL

```
curl -X DELETE "https://api.zenpays.com/merchant/api/v1/settlements/bank-accounts/ba_xxxxx" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
await zenpays.settlements.deleteBankAccount('ba_xxxxx');
```

Accounting

REST API / ACCOUNTING

Get Wallet Balances

Retrieve all wallet balances across currencies.

`GET` **`/merchant/api/v1/wallet/balances`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "wallet_id": "wal_xxxxx",
      "currency": "USD",
      "currency_symbol": "$",
      "available_balance": 250000,
      "total_balance": 270000,
      "blocked_balance": 5000,
      "pending_balance": 15000,
      "lifetime_balance": 1500000,
      "lifetime_withdrawn": 1230000,
      "status": "ACTIVE",
      "last_transaction_at": "2024-01-15T10:30:00.000Z"
    },
    {
      "wallet_id": "wal_yyyyy",
      "currency": "INR",
      "currency_symbol": "₹",
      "available_balance": 5000000,
      "total_balance": 5350000,
      "blocked_balance": 100000,
      "pending_balance": 250000,
      "lifetime_balance": 25000000,
      "lifetime_withdrawn": 19650000,
      "status": "ACTIVE",
      "last_transaction_at": "2024-01-15T10:30:00.000Z"
    }
  ]
}
```

Balance Fields

Parameter	Type	Description
`available_balance`	`number`	Funds available for withdrawal/payout
`total_balance`	`number`	Total balance including all holds
`blocked_balance`	`number`	Funds blocked for disputes/reserves
`pending_balance`	`number`	Funds being processed (incoming payments)
`lifetime_balance`	`number`	Total funds ever received
`lifetime_withdrawn`	`number`	Total funds ever withdrawn
`currency`	`string`	ISO 4217 currency code
`currency_symbol`	`string`	Display symbol for the currency
`last_transaction_at`	`string`	ISO 8601 timestamp of last wallet activity
`status`	`string`	Wallet status

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/wallet/balances" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const balances = await zenpays.wallet.getAllBalances();
balances.forEach(b => {
  console.log(`${b.currency}: ${b.available} available`);
});
```

REST API / ACCOUNTING

Get Currency Balance

Retrieve wallet balance for a specific currency. Also used to create a new currency wallet if one does not exist.

Get Balance

`GET` **`/merchant/api/v1/wallet/currency/{currency}`**

Path Parameters

Parameter	Type	Required	Description
`currency`	`string`	Yes	ISO 4217 currency code

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response (200 OK)

```
{
  "success": true,
  "data": {
    "wallet_id": "wal_xxxxx",
    "currency": "USD",
    "currency_symbol": "$",
    "available_balance": 250000,
    "total_balance": 270000,
    "blocked_balance": 5000,
    "pending_balance": 15000,
    "lifetime_balance": 1500000,
    "lifetime_withdrawn": 1230000,
    "status": "ACTIVE",
    "last_transaction_at": "2024-01-15T10:30:00.000Z"
  }
}
```

Create Currency Wallet

`POST` **`/merchant/api/v1/wallet/currency/{currency}`**

Creates a new wallet for the specified currency.

Path Parameters

Parameter	Type	Required	Description
`currency`	`string`	Yes	ISO 4217 currency code

Response (201 Created)

```
{
  "success": true,
  "data": {
    "wallet_id": "wal_eur_001",
    "currency": "EUR",
    "currency_symbol": "€",
    "available_balance": 0,
    "total_balance": 0,
    "blocked_balance": 0,
    "pending_balance": 0,
    "lifetime_balance": 0,
    "lifetime_withdrawn": 0,
    "status": "ACTIVE",
    "last_transaction_at": "2024-01-15T10:30:00.000Z"
  }
}
```

Examples

CURL

```
# Get balance for a currency
curl -X GET "https://api.zenpays.com/merchant/api/v1/wallet/currency/USD" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"

# Create a new currency wallet
curl -X POST "https://api.zenpays.com/merchant/api/v1/wallet/currency/EUR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// Get balance for a currency
const balance = await zenpays.wallet.getBalance('USD');
console.log(`Available: ${balance.available}`);

// Create a new currency wallet via REST
// POST /merchant/api/v1/wallet/currency/EUR
```

REST API / ACCOUNTING

Get Wallet Transactions

Retrieve transaction history for a specific currency wallet.

List Transactions

`GET` **`/merchant/api/v1/wallet/{currency}/transactions`**

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Path Parameters

Parameter	Type	Required	Description
`currency`	`string`	Yes	ISO 4217 currency code for the wallet

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page
`startDate`	`string`	`-`	Start date filter (ISO 8601)
`endDate`	`string`	`-`	End date filter (ISO 8601)
`type`	`string`	`-`	Filter by type (`CREDIT` , `DEBIT` , `SETTLEMENT` , `COMMISSION` , `REFUND` , `CHARGEBACK`)
`status`	`string`	`-`	Filter by status (`PENDING` , `PROCESSING` , `COMPLETED` , `FAILED` , `REVERSED`)
`search`	`string`	`-`	Search across transaction id, source transaction id, source order id, and amount-like fields

Response (200 OK)


```
{
  "success": true,
  "data": {
    "data": [
      {
        "transactionId": "wtx_xxxxx",
        "type": "CREDIT",
        "amount": 1000,
        "balanceBefore": 249000,
        "balanceAfter": 250000,
        "description": "Payment received",
        "sourceType": "PAYMENT",
        "sourceTransactionId": "txn_xxxxx",
        "sourceOrderId": "ord_xxxxx",
        "status": "COMPLETED",
        "processedAt": "2024-01-15T10:30:00.000Z",
        "settlementDate": "2024-01-16T00:00:00.000Z",
        "currency": "USD",
        "feeAmount": 0,
        "grossAmount": 1000,
        "createdAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "pagination": {
      "page": 1,
      "limit": 20,
      "total": 500,
      "total_pages": 25,
      "has_next": true,
      "has_prev": false
    }
  }
}
```


Important Note

`GET /merchant/api/v1/wallet/{currency}/transactions/{transactionId}` is **not currently exposed** on the merchant wallet route surface. Use the list endpoint with filters/search.

Transaction Types

Type	Description
`CREDIT`	Money added to wallet
`DEBIT`	Money withdrawn from wallet
`SETTLEMENT`	Settlement to bank account
`COMMISSION`	Commission or fee
`REFUND`	Refund processed
`CHARGEBACK`	Chargeback deduction

Transaction Statuses

Status	Description
`PENDING`	Transaction initiated, awaiting processing
`PROCESSING`	Transaction being processed
`COMPLETED`	Transaction completed
`FAILED`	Transaction failed
`REVERSED`	Transaction reversed

Examples

CURL

```
# List wallet transactions
curl -X GET "https://api.zenpays.com/merchant/api/v1/wallet/USD/transactions?
type=CREDIT&limit=50&page=1&search=txn_123" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// SDK currently provides a filter-based wallet transactions API.
// Use `currency` in filters to scope to one wallet.
const result = await zenpays.wallet.listTransactions({
  currency: 'USD',
  type: 'credit',
  limit: 50,
  page: 1,
});
```

REST API / ACCOUNTING

Get Wallet Summary

Get a summary of wallet activity and balances. Additional endpoints are available for monthly analytics and trend analytics.

Wallet Summary

`GET` **/merchant/api/v1/wallet/summary**

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response (200 OK)


```
{
  "success": true,
  "data": {
    "total_wallets": 3,
    "active_currencies": ["USD", "INR", "EUR"],
    "total_balance_in_base_currency": 6200000,
    "base_currency": "USD",
    "currency_balances": [
      {
        "currency": "USD",
        "available_balance": 250000,
        "total_balance": 270000
      },
      {
        "currency": "INR",
        "available_balance": 5000000,
        "total_balance": 5350000
      }
    ]
  }
}
```

Monthly Analytics

`GET` **/merchant/api/v1/wallet/analytics/monthly**

Retrieve monthly wallet analytics and trends.

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Response (200 OK)

```
{
  "success": true,
  "data": {
    "monthly_data": [
      {
        "month": "2026-01",
        "total_credited": 500000,
        "total_debited": 350000,
        "transaction_count": 1200
      }
    ]
  }
}
```

Wallet Analytics

GET `/merchant/api/v1/wallet/analytics`

Retrieve wallet analytics with credit/debit trends over time.

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Required	Description
`currency`	`string`	Yes	ISO 4217 currency code
`startDate`	`string`	No	Start date (ISO 8601)
`endDate`	`string`	No	End date (ISO 8601)
`granularity`	`string`	No	`daily`, `weekly`, or `monthly` (default: `daily`)

Examples

CURL

```
# Get wallet summary
curl -X GET "https://api.zenpays.com/merchant/api/v1/wallet/summary" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"

# Get monthly analytics
curl -X GET "https://api.zenpays.com/merchant/api/v1/wallet/analytics/monthly" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"

# Get wallet analytics
curl -X GET "https://api.zenpays.com/merchant/api/v1/wallet/analytics?
currency=USD&granularity=daily" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// Wallet summary and wallet analytics endpoints are REST-first in the
// merchant API surface.
// Use direct REST calls:
// GET /merchant/api/v1/wallet/summary
// GET /merchant/api/v1/wallet/analytics/monthly
// GET /merchant/api/v1/wallet/analytics?currency=USD&granularity=daily
```

Ledger

REST API / LEDGER

Get Ledger Overview

Retrieve a comprehensive ledger overview including account balances, totals, and the latest daily summary.

`GET` **`/merchant/api/v1/ledger/overview`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`currency`	`string`	`-`	Filter by currency code (e.g., `INR`, `USD`)
`startDate`	`string`	`-`	Start date for period (YYYY-MM-DD)
`endDate`	`string`	`-`	End date for period (YYYY-MM-DD)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "balances": [
      {
        "accountCategory": "MERCHANT_AVAILABLE",
        "currency": "INR",
        "balance": 10000
      },
      {
        "accountCategory": "MERCHANT_PENDING",
        "currency": "INR",
        "balance": 2500
      },
      {
        "accountCategory": "MERCHANT_BLOCKED",
        "currency": "INR",
        "balance": 500
      }
    ],
    "totalAvailable": 10000,
    "totalPending": 2500,
    "totalBlocked": 500,
    "totalReserved": 1000,
    "currency": "INR",
    "latestSummary": {
      "summaryId": "sum_xxxxx",
      "summaryDate": "2024-01-15",
      "merchantId": "mer_xxxxx",
      "currency": "INR",
      "openingAvailableBalance": 8000,
      "closingAvailableBalance": 10000,
      "totalDepositsGross": 5000,
      "totalDepositsNet": 4750,
      "depositCount": 10,
      "totalPayouts": 2000,
      "payoutCount": 2,
      "netMovement": 2000,
      "isFinalized": true
    }
  },
  "message": "Ledger overview retrieved successfully"
}
```

Account Categories

Category	Description
`MERCHANT_AVAILABLE`	Funds available for withdrawal/payout
`MERCHANT_PENDING`	Funds being processed (incoming payments)
`MERCHANT_BLOCKED`	Funds blocked for pending operations
`RESERVE_PAYOUT`	Reserved for pending payouts
`RESERVE_RISK`	Risk reserve allocation
`RESERVE_CHARGEBACK`	Chargeback reserve allocation

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/overview?currency=INR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const overview = await zenpays.ledger.getOverview({ currency: 'INR' });
console.log(`Available: ${overview.totalAvailable}`);
console.log(`Pending: ${overview.totalPending}`);
console.log(`Reserved: ${overview.totalReserved}`);
```

REST API / LEDGER

Get Ledger Entries

Retrieve paginated ledger entries (double-entry bookkeeping records) with optional filters.

`GET` **`/merchant/api/v1/ledger/entries`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page (max 100)
`startDate`	`string`	`-`	Filter from date (YYYY-MM-DD)
`endDate`	`string`	`-`	Filter to date (YYYY-MM-DD)
`currency`	`string`	`-`	Filter by currency code
`entryType`	`string`	`-`	Filter by type: `DEBIT` or `CREDIT`

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "entries": [
      {
        "entryId": "le_xxxxx",
        "batchId": "batch_xxxxx",
        "accountId": "acc_xxxxx",
        "merchantId": "mer_xxxxx",
        "entryType": "CREDIT",
        "entryCategory": "DEPOSIT_NET",
        "amount": 950,
        "currency": "INR",
        "balanceBefore": 9050,
        "balanceAfter": 10000,
        "sourceType": "PAYMENT",
        "sourceTransactionId": "txn_xxxxx",
        "sourceOrderId": "order_xxxxx",
        "description": "Payment deposit (net after fees)",
        "postedAt": "2024-01-15T10:30:00.000Z",
        "createdAt": "2024-01-15T10:30:00.000Z"
      },
      {
        "entryId": "le_yyyyy",
        "batchId": "batch_xxxxx",
        "accountId": "acc_xxxxx",
        "merchantId": "mer_xxxxx",
        "entryType": "DEBIT",
        "entryCategory": "PAYOUT",
        "amount": 5000,
        "currency": "INR",
        "balanceBefore": 15000,
        "balanceAfter": 10000,
        "sourceType": "PAYOUT",
        "sourceTransactionId": "po_xxxxx",
        "description": "Payout withdrawal",
        "postedAt": "2024-01-15T09:00:00.000Z",
        "createdAt": "2024-01-15T09:00:00.000Z"
      }
    ],
    "pagination": {
      "page": 1,
      "limit": 20,
      "total": 150,
      "totalPages": 8,
      "hasNext": true,
      "hasPrev": false
    },
    "total": 150
  },
  "message": "Ledger entries retrieved successfully"
}
```

Entry Types

Type	Description
`DEBIT`	Decrease in account balance
`CREDIT`	Increase in account balance

Entry Categories

Category	Description
`DEPOSIT_GROSS`	Gross deposit amount before fees
`DEPOSIT_NET`	Net deposit amount after fees
`DEPOSIT_FEE`	Fee deducted from deposit
`PAYOUT`	Payout withdrawal
`PAYOUT_FEE`	Payout processing fee
`REFUND`	Refund to customer
`CHARGEBACK`	Chargeback deduction
`SETTLEMENT`	Settlement transfer
`RESERVE_ALLOCATION`	Reserve allocation
`RESERVE_RELEASE`	Reserve release

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/entries?
page=1&limit=20&currency=INR&entryType=CREDIT" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const { entries, pagination } = await zenpays.ledger.getEntries({
  page: 1,
  limit: 20,
  currency: 'INR',
  entryType: 'CREDIT'
});

entries.forEach(entry => {
  console.log(`${entry.entryType}: ${entry.amount} ${entry.currency} - ${entry.description}`);
});
```

REST API / LEDGER

Get Ledger Balances

Retrieve authoritative ledger account balances for the merchant. This is the source of truth for all balance calculations.

`GET` **`/merchant/api/v1/ledger/balances`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`currency`	`string`	`-`	Filter by currency code (e.g., `INR` , `USD`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "accountCategory": "MERCHANT_AVAILABLE",
      "currency": "INR",
      "balance": 10000
    },
    {
      "accountCategory": "MERCHANT_PENDING",
      "currency": "INR",
      "balance": 2500
    },
    {
      "accountCategory": "MERCHANT_BLOCKED",
      "currency": "INR",
      "balance": 500
    },
    {
      "accountCategory": "MERCHANT_AVAILABLE",
      "currency": "USD",
      "balance": 250
    }
  ],
  "message": "Ledger balances retrieved successfully"
}
```


Account Categories

Category	Description
`MERCHANT_AVAILABLE`	Funds available for withdrawal/payout
`MERCHANT_PENDING`	Funds being processed (incoming payments not yet settled)
`MERCHANT_BLOCKED`	Funds blocked for pending operations (payouts, settlements)
`RESERVE_PAYOUT`	Reserved funds for pending payout operations
`RESERVE_RISK`	Risk reserve allocation based on merchant risk profile
`RESERVE_CHARGEBACK`	Funds reserved for potential chargebacks

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/balances?currency=INR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const balances = await zenpays.ledger.getBalances({ currency: 'INR' });

balances.forEach(balance => {
  console.log(`${balance.accountCategory}: ${balance.balance} ${balance.currency}`);
});

// Get available balance
const available = balances.find(b => b.accountCategory === 'MERCHANT_AVAILABLE');
console.log(`Available for withdrawal: ${available?.balance || 0}`);
```

REST API / LEDGER

Get Ledger Accounts

Retrieve all ledger accounts for the merchant. This is an alias for ledger balances with a semantic focus on account listing.

`GET` **`/merchant/api/v1/ledger/accounts`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`currency`	`string`	`-`	Filter by currency code (e.g., `INR` , `USD`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "accountCategory": "MERCHANT_AVAILABLE",
      "currency": "INR",
      "balance": 10000
    },
    {
      "accountCategory": "MERCHANT_PENDING",
      "currency": "INR",
      "balance": 2500
    },
    {
      "accountCategory": "MERCHANT_BLOCKED",
      "currency": "INR",
      "balance": 500
    },
    {
      "accountCategory": "RESERVE_PAYOUT",
      "currency": "INR",
      "balance": 1000
    },
    {
      "accountCategory": "RESERVE_RISK",
      "currency": "INR",
      "balance": 250
    }
  ],
  "message": "Ledger accounts retrieved successfully"
}
```


Account Categories

Category	Description
`MERCHANT_AVAILABLE`	Funds available for withdrawal/payout
`MERCHANT_PENDING`	Funds being processed (incoming payments not yet settled)
`MERCHANT_BLOCKED`	Funds blocked for pending operations (payouts, settlements)
`COLLECTION_POOL`	Pool for collecting incoming payments
`ADMIN_WALLET`	Administrative wallet for platform operations
`RESERVE_PAYOUT`	Reserved funds for pending payout operations
`RESERVE_RISK`	Risk reserve allocation based on merchant risk profile
`RESERVE_CHARGEBACK`	Funds reserved for potential chargebacks
`TSP_PAYABLE`	Amounts payable to third-party service providers
`SYSTEM_SUSPENSE`	Suspense account for unreconciled transactions

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/accounts?currency=INR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const accounts = await zenpays.ledger.getAccounts({ currency: 'INR' });

// List all accounts
accounts.forEach(account => {
  console.log(`${account.accountCategory}: ${account.balance} ${account.currency}`);
});

// Get specific account type
const reserveAccounts = accounts.filter(a => a.accountCategory.startsWith('RESERVE_'));
console.log('Reserve accounts:', reserveAccounts);
```

REST API / LEDGER

Get Ledger Reserves

Retrieve reserve allocations for the merchant. Reserves are funds set aside for payouts, risk mitigation, and potential chargebacks.

`GET` **`/merchant/api/v1/ledger/reserves`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`reserveType`	`string`	`-`	Filter by reserve type (`PAYOUT_RESERVE` , `RISK_RESERVE` , `CHARGEBACK_RESERVE`)
`status`	`string`	`-`	Filter by status (`ACTIVE` , `PARTIALLY_RELEASED` , `FULLY_RELEASED` , `FORFEITED`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "reserves": [
      {
        "reserveId": "rsv_abc123",
        "merchantId": "mer_xyz789",
        "walletId": "wal_def456",
        "reserveType": "PAYOUT_RESERVE",
        "reserveReason": "Pending payout batch processing",
        "originalAmount": 50000,
        "currentAmount": 50000,
        "releasedAmount": 0,
        "currency": "INR",
        "status": "ACTIVE",
        "reservedAt": "2024-01-15T10:00:00.000Z",
        "releaseScheduledAt": "2024-01-16T10:00:00.000Z",
        "releasedAt": null,
        "releasePeriodDays": 1,
        "reservePercentage": null,
        "sourceTransactionId": "payout_batch_001",
        "createdBy": "system",
        "releasedBy": null,
        "releaseNotes": null
      },
      {
        "reserveId": "rsv_def456",
        "merchantId": "mer_xyz789",
        "walletId": "wal_def456",
        "reserveType": "RISK_RESERVE",
        "reserveReason": "Monthly rolling risk reserve",
        "originalAmount": 25000,
        "currentAmount": 20000,
        "releasedAmount": 5000,
        "currency": "INR",
        "status": "PARTIALLY_RELEASED",
        "reservedAt": "2024-01-01T00:00:00.000Z",
        "releaseScheduledAt": "2024-02-01T00:00:00.000Z",
        "releasedAt": null,
        "releasePeriodDays": 30,
        "reservePercentage": 5.0,
        "sourceTransactionId": null,
        "createdBy": "risk_engine",
        "releasedBy": null,
        "releaseNotes": null
      },
      {
        "reserveId": "rsv_ghi789",
        "merchantId": "mer_xyz789",
        "walletId": "wal_def456",
        "reserveType": "CHARGEBACK_RESERVE",
        "reserveReason": "Chargeback dispute CBK-2024-001",
        "originalAmount": 1500,
        "currentAmount": 1500,
        "releasedAmount": 0,
        "currency": "INR",
        "status": "ACTIVE",
        "reservedAt": "2024-01-10T14:30:00.000Z",
        "releaseScheduledAt": "2024-04-10T14:30:00.000Z",
```

```
        "releasedAt": null,
        "releasePeriodDays": 90,
        "reservePercentage": null,
        "sourceTransactionId": "txn_chargeback_001",
        "createdBy": "chargeback_service",
        "releasedBy": null,
        "releaseNotes": null
      }
    ],
    "totalReserved": 71500
  },
  "message": "Reserves retrieved successfully"
}
```


Reserve Types

Type	Description
`PAYOUT_RESERVE`	Funds reserved for pending payout operations
`RISK_RESERVE`	Rolling reserve based on merchant risk profile (typically 5-10% of volume)
`CHARGEBACK_RESERVE`	Funds held during chargeback dispute resolution

Reserve Statuses

Status	Description
`ACTIVE`	Reserve is currently held, full amount blocked
`PARTIALLY_RELEASED`	Part of the reserve has been released
`FULLY_RELEASED`	Reserve has been completely released back to merchant
`FORFEITED`	Reserve was forfeited (e.g., chargeback lost)

Examples

CURL

```
# Get all reserves
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/reserves" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"

# Get only active payout reserves
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/reserves?
reserveType=PAYOUT_RESERVE&status=ACTIVE" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// Get all reserves
const { reserves, totalReserved } = await zenpays.ledger.getReserves();

console.log(`Total reserved: ${totalReserved}`);

reserves.forEach(reserve => {
  console.log(`${reserve.reserveType}: ${reserve.currentAmount} ${reserve.currency}
(${reserve.status})`);
});

// Get only active risk reserves
const riskReserves = await zenpays.ledger.getReserves({
  reserveType: 'RISK_RESERVE',
  status: 'ACTIVE'
});

// Calculate upcoming releases
const upcomingReleases = reserves.filter(r => {
  const releaseDate = new Date(r.releaseScheduledAt);
  const nextWeek = new Date();
  nextWeek.setDate(nextWeek.getDate() + 7);
  return releaseDate <= nextWeek && r.status === 'ACTIVE';
});

console.log('Reserves releasing in next 7 days:', upcomingReleases);
```

REST API / LEDGER

Get Daily Ledger Summary

Retrieve the daily ledger summary for a specific date. This provides a comprehensive snapshot of all financial activity for that day, including opening/closing balances and transaction counts.

`GET` **`/merchant/api/v1/ledger/summary/daily`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Required	Default	Description
`date`	`string`	Yes	`-`	Date in `YYYY-MM-DD` format
`currency`	`string`	No	`-`	Filter by currency code (e.g., `INR`, `USD`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "summaryId": "sum_20240115_mer_xyz789",
    "summaryDate": "2024-01-15",
    "merchantId": "mer_xyz789",
    "currency": "INR",
    "openingAvailableBalance": 100000,
    "closingAvailableBalance": 125000,
    "openingPendingBalance": 5000,
    "closingPendingBalance": 7500,
    "openingBlockedBalance": 2000,
    "closingBlockedBalance": 3000,
    "totalDepositsGross": 35000,
    "totalDepositsNet": 33250,
    "depositCount": 15,
    "totalDepositFees": 1750,
    "totalPayouts": 8000,
    "payoutCount": 3,
    "totalPayoutFees": 250,
    "totalRefunds": 1500,
    "refundCount": 2,
    "totalChargebacks": 500,
    "chargebackCount": 1,
    "totalSettlements": 0,
    "settlementCount": 0,
    "totalFeesCollected": 2000,
    "totalTransactionCount": 21,
    "grossTransactionValue": 45000,
    "netMovement": 25000,
    "generatedAt": "2024-01-16T00:05:00.000Z",
    "isFinalized": true
  },
  "message": "Daily summary retrieved successfully"
}
```


Response Fields

Parameter	Type	Description
`summaryId`	`string`	Unique identifier for the summary
`summaryDate`	`string`	Date of the summary (YYYY-MM-DD)
`merchantId`	`string`	Merchant identifier
`currency`	`string`	Currency code
`openingAvailableBalance`	`number`	Available balance at start of day
`closingAvailableBalance`	`number`	Available balance at end of day
`openingPendingBalance`	`number`	Pending balance at start of day
`closingPendingBalance`	`number`	Pending balance at end of day
`openingBlockedBalance`	`number`	Blocked balance at start of day

Parameter	Type	Description
`closingBlockedBalance`	`number`	Blocked balance at end of day
`totalDepositsGross`	`number`	Total deposits before fees
`totalDepositsNet`	`number`	Total deposits after fees
`depositCount`	`number`	Number of deposit transactions
`totalDepositFees`	`number`	Total fees charged on deposits
`totalPayouts`	`number`	Total payout amount
`payoutCount`	`number`	Number of payout transactions
`totalPayoutFees`	`number`	Total fees charged on payouts
`totalRefunds`	`number`	Total refund amount
`refundCount`	`number`	Number of refund transactions
`totalChargebacks`	`number`	Total chargeback amount
`chargebackCount`	`number`	Number of chargeback transactions
`totalSettlements`	`number`	Total settlement amount
`settlementCount`	`number`	Number of settlements
`totalFeesCollected`	`number`	Total fees collected (all types)
`totalTransactionCount`	`number`	Total number of transactions
`grossTransactionValue`	`number`	Gross value of all transactions
`netMovement`	`number`	Net change in merchant balance
`generatedAt`	`string`	When the summary was generated
`isFinalized`	`boolean`	Whether the day is closed and summary is final

Examples

CURL

```
# Get daily summary for specific date
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/summary/daily?date=2024-01-
15&currency=INR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// Get today's summary
const today = new Date().toISOString().split('T')[0];
const summary = await zenpays.ledger.getDailySummary({
  date: today,
  currency: 'INR'
});

console.log(`Date: ${summary.summaryDate}`);
console.log(`Opening Balance: ${summary.openingAvailableBalance}`);
console.log(`Closing Balance: ${summary.closingAvailableBalance}`);
console.log(`Net Movement: ${summary.netMovement}`);
console.log(`Total Transactions: ${summary.totalTransactionCount}`);
console.log(`Finalized: ${summary.isFinalized}`);

// Calculate daily metrics
const depositSuccessRate = summary.depositCount > 0
  ? (summary.totalDepositsNet / summary.totalDepositsGross * 100).toFixed(2)
  : 0;
console.log(`Effective deposit rate: ${depositSuccessRate}%`);

// Get yesterday's summary for comparison
const yesterday = new Date();
yesterday.setDate(yesterday.getDate() - 1);
const yesterdaySummary = await zenpays.ledger.getDailySummary({
  date: yesterday.toISOString().split('T')[0],
  currency: 'INR'
});

const growthRate = ((summary.totalDepositsGross - yesterdaySummary.totalDepositsGross) /
                    yesterdaySummary.totalDepositsGross * 100).toFixed(2);
console.log(`Day-over-day deposit growth: ${growthRate}%`);
```

REST API / LEDGER

Get Daily Ledger Summaries

Retrieve historical daily ledger summaries for a date range. This endpoint returns an array of daily summaries, useful for trend analysis and reporting.

`GET` **`/merchant/api/v1/ledger/summaries`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Required	Default	Description
`startDate`	`string`	Yes	-	Start date in `YYYY-MM-DD` format
`endDate`	`string`	Yes	-	End date in `YYYY-MM-DD` format
`currency`	`string`	No	-	Filter by currency code (e.g., `INR`, `USD`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "summaryId": "sum_20240113_mer_xyz789",
      "summaryDate": "2024-01-13",
      "merchantId": "mer_xyz789",
      "currency": "INR",
      "openingAvailableBalance": 85000,
      "closingAvailableBalance": 95000,
      "openingPendingBalance": 3000,
      "closingPendingBalance": 4500,
      "openingBlockedBalance": 1500,
      "closingBlockedBalance": 2000,
      "totalDepositsGross": 15000,
      "totalDepositsNet": 14250,
      "depositCount": 8,
      "totalDepositFees": 750,
      "totalPayouts": 4000,
      "payoutCount": 2,
      "totalPayoutFees": 100,
      "totalRefunds": 500,
      "refundCount": 1,
      "totalChargebacks": 0,
      "chargebackCount": 0,
      "totalSettlements": 0,
      "settlementCount": 0,
      "totalFeesCollected": 850,
      "totalTransactionCount": 11,
      "grossTransactionValue": 19500,
      "netMovement": 10000,
      "generatedAt": "2024-01-14T00:05:00.000Z",
      "isFinalized": true
    },
    {
      "summaryId": "sum_20240114_mer_xyz789",
      "summaryDate": "2024-01-14",
      "merchantId": "mer_xyz789",
      "currency": "INR",
      "openingAvailableBalance": 95000,
      "closingAvailableBalance": 100000,
      "openingPendingBalance": 4500,
      "closingPendingBalance": 5000,
      "openingBlockedBalance": 2000,
      "closingBlockedBalance": 2000,
      "totalDepositsGross": 8000,
      "totalDepositsNet": 7600,
      "depositCount": 5,
      "totalDepositFees": 400,
      "totalPayouts": 2500,
      "payoutCount": 1,
      "totalPayoutFees": 75,
      "totalRefunds": 0,
      "refundCount": 0,
      "totalChargebacks": 0,
      "chargebackCount": 0,
      "totalSettlements": 0,
      "settlementCount": 0,
      "totalFeesCollected": 475,
```

```
      "totalTransactionCount": 6,
      "grossTransactionValue": 10500,
      "netMovement": 5000,
      "generatedAt": "2024-01-15T00:05:00.000Z",
      "isFinalized": true
    },
    {
      "summaryId": "sum_20240115_mer_xyz789",
      "summaryDate": "2024-01-15",
      "merchantId": "mer_xyz789",
      "currency": "INR",
      "openingAvailableBalance": 100000,
      "closingAvailableBalance": 125000,
      "openingPendingBalance": 5000,
      "closingPendingBalance": 7500,
      "openingBlockedBalance": 2000,
      "closingBlockedBalance": 3000,
      "totalDepositsGross": 35000,
      "totalDepositsNet": 33250,
      "depositCount": 15,
      "totalDepositFees": 1750,
      "totalPayouts": 8000,
      "payoutCount": 3,
      "totalPayoutFees": 250,
      "totalRefunds": 1500,
      "refundCount": 2,
      "totalChargebacks": 500,
      "chargebackCount": 1,
      "totalSettlements": 0,
      "settlementCount": 0,
      "totalFeesCollected": 2000,
      "totalTransactionCount": 21,
      "grossTransactionValue": 45000,
      "netMovement": 25000,
      "generatedAt": "2024-01-16T00:05:00.000Z",
      "isFinalized": true
    }
  ],
  "message": "Daily summaries retrieved successfully"
}
```


Notes

- Summaries are returned in chronological order (oldest first)
- Days without any transactions may not have a summary record
- The `isFinalized` field indicates whether the day is closed; today's summary may not be finalized
- Maximum date range is typically 90 days; for longer periods, make multiple requests

Examples

CURL

```
# Get last 7 days of summaries
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/summaries?startDate=2024-01-
09&endDate=2024-01-15&currency=INR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// Get last 30 days of summaries
const endDate = new Date();
const startDate = new Date();
startDate.setDate(startDate.getDate() - 30);

const summaries = await zenpays.ledger.getDailySummaries({
  startDate: startDate.toISOString().split('T')[0],
  endDate: endDate.toISOString().split('T')[0],
  currency: 'INR'
});

// Calculate period totals
const periodTotals = summaries.reduce((acc, day) => ({
  totalDeposits: acc.totalDeposits + day.totalDepositsGross,
  totalPayouts: acc.totalPayouts + day.totalPayouts,
  totalFees: acc.totalFees + day.totalFeesCollected,
  totalTransactions: acc.totalTransactions + day.totalTransactionCount,
}), { totalDeposits: 0, totalPayouts: 0, totalFees: 0, totalTransactions: 0 });

console.log('30-Day Period Summary:');
console.log(`Total Deposits: ${periodTotals.totalDeposits}`);
console.log(`Total Payouts: ${periodTotals.totalPayouts}`);
console.log(`Total Fees: ${periodTotals.totalFees}`);
console.log(`Total Transactions: ${periodTotals.totalTransactions}`);

// Calculate daily averages
const avgDailyDeposits = periodTotals.totalDeposits / summaries.length;
const avgDailyTransactions = periodTotals.totalTransactions / summaries.length;
console.log(`\nDaily Averages:`);
console.log(`Avg Daily Deposits: ${avgDailyDeposits.toFixed(2)}`);
console.log(`Avg Daily Transactions: ${avgDailyTransactions.toFixed(2)}`);

// Find best and worst days
const bestDay = summaries.reduce((best, day) =>
  day.totalDepositsGross > best.totalDepositsGross ? day : best
);
const worstDay = summaries.reduce((worst, day) =>
  day.totalDepositsGross < worst.totalDepositsGross ? day : worst
);

console.log(`\nBest Day: ${bestDay.summaryDate} (${bestDay.totalDepositsGross} deposits)`);
console.log(`Worst Day: ${worstDay.summaryDate} (${worstDay.totalDepositsGross} deposits)`);

// Trend analysis: week-over-week growth
const firstWeek = summaries.slice(0, 7);
const lastWeek = summaries.slice(-7);

const firstWeekTotal = firstWeek.reduce((sum, day) => sum + day.totalDepositsGross, 0);
const lastWeekTotal = lastWeek.reduce((sum, day) => sum + day.totalDepositsGross, 0);
const weeklyGrowth = ((lastWeekTotal - firstWeekTotal) / firstWeekTotal * 100).toFixed(2);

console.log(`\nWeek-over-week growth: ${weeklyGrowth}%`);
```

REST API / LEDGER

Get Ledger Reconciliation

Retrieve reconciliation data combining the daily summary with current account balances. This endpoint is designed for daily reconciliation workflows and financial auditing.

`GET` **`/merchant/api/v1/ledger/reconciliation`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Required	Default	Description
`date`	`string`	Yes	`-`	Date in `YYYY-MM-DD` format
`currency`	`string`	No	`-`	Filter by currency code (e.g., `INR`, `USD`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "summary": {
      "summaryId": "sum_20240115_mer_xyz789",
      "summaryDate": "2024-01-15",
      "merchantId": "mer_xyz789",
      "currency": "INR",
      "openingAvailableBalance": 100000,
      "closingAvailableBalance": 125000,
      "openingPendingBalance": 5000,
      "closingPendingBalance": 7500,
      "openingBlockedBalance": 2000,
      "closingBlockedBalance": 3000,
      "totalDepositsGross": 35000,
      "totalDepositsNet": 33250,
      "depositCount": 15,
      "totalDepositFees": 1750,
      "totalPayouts": 8000,
      "payoutCount": 3,
      "totalPayoutFees": 250,
      "totalRefunds": 1500,
      "refundCount": 2,
      "totalChargebacks": 500,
      "chargebackCount": 1,
      "totalSettlements": 0,
      "settlementCount": 0,
      "totalFeesCollected": 2000,
      "totalTransactionCount": 21,
      "grossTransactionValue": 45000,
      "netMovement": 25000,
      "generatedAt": "2024-01-16T00:05:00.000Z",
      "isFinalized": true
    },
    "balances": [
      {
        "accountCategory": "MERCHANT_AVAILABLE",
        "currency": "INR",
        "balance": 125000
      },
      {
        "accountCategory": "MERCHANT_PENDING",
        "currency": "INR",
        "balance": 7500
      },
      {
        "accountCategory": "MERCHANT_BLOCKED",
        "currency": "INR",
        "balance": 3000
      },
      {
        "accountCategory": "RESERVE_PAYOUT",
        "currency": "INR",
        "balance": 5000
      },
      {
        "accountCategory": "RESERVE_RISK",
        "currency": "INR",
        "balance": 1250
```

```
      },
      {
        "accountCategory": "RESERVE_CHARGEBACK",
        "currency": "INR",
        "balance": 500
      }
    ]
  },
  "message": "Reconciliation data retrieved successfully"
}
```


Reconciliation Verification

To verify reconciliation, ensure the following conditions are met:

1. **Balance Continuity**: Today's opening balance should equal yesterday's closing balance
2. **Net Movement Calculation**:

   ```
   netMovement = totalDepositsNet - totalPayouts - totalRefunds - totalChargebacks
   ```

3. **Closing Balance Verification**:

   ```
   closingAvailableBalance = openingAvailableBalance + netMovement
   ```

4. **Current Balance Match**: The `balances` array should match the summary's closing balances

Discrepancy Indicators

Check	Formula	Action if Failed
Balance continuity	`today.openingBalance == yesterday.closingBalance`	Investigate overnight adjustments
Movement accuracy	`netMovement == deposits - payouts - refunds - chargebacks`	Check for missing transactions
Real-time match	`balances.MERCHANT_AVAILABLE == summary.closingAvailableBalance`	May indicate pending transactions

Examples

CURL

```
# Get reconciliation data for specific date
curl -X GET "https://api.zenpays.com/merchant/api/v1/ledger/reconciliation?date=2024-01-15&currency=INR" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
// Get reconciliation data for yesterday (finalized)
const yesterday = new Date();
yesterday.setDate(yesterday.getDate() - 1);
const dateStr = yesterday.toISOString().split('T')[0];

const { summary, balances } = await zenpays.ledger.getReconciliation({
  date: dateStr,
  currency: 'INR'
});

console.log(`Reconciliation for ${summary.summaryDate}`);
console.log(`Status: ${summary.isFinalized ? 'Finalized' : 'Pending'}`);

// Verify net movement calculation
const calculatedNetMovement =
  summary.totalDepositsNet -
  summary.totalPayouts -
  summary.totalRefunds -
  summary.totalChargebacks;

const movementMatch = calculatedNetMovement === summary.netMovement;
console.log(`\nNet Movement Verification:`);
console.log(`Reported: ${summary.netMovement}`);
console.log(`Calculated: ${calculatedNetMovement}`);
console.log(`Match: ${movementMatch ? 'YES' : 'DISCREPANCY!'}`);

// Verify closing balance
const expectedClosing = summary.openingAvailableBalance + summary.netMovement;
const closingMatch = expectedClosing === summary.closingAvailableBalance;
console.log(`\nClosing Balance Verification:`);
console.log(`Expected: ${expectedClosing}`);
console.log(`Reported: ${summary.closingAvailableBalance}`);
console.log(`Match: ${closingMatch ? 'YES' : 'DISCREPANCY!'}`);

// Compare with current balances
const availableBalance = balances.find(b => b.accountCategory === 'MERCHANT_AVAILABLE');
const pendingBalance = balances.find(b => b.accountCategory === 'MERCHANT_PENDING');
const blockedBalance = balances.find(b => b.accountCategory === 'MERCHANT_BLOCKED');

console.log(`\nCurrent vs Closing Balance Comparison:`);
console.log(`Available: Current ${availableBalance?.balance} vs Closing
${summary.closingAvailableBalance}`);
console.log(`Pending: Current ${pendingBalance?.balance} vs Closing
${summary.closingPendingBalance}`);
console.log(`Blocked: Current ${blockedBalance?.balance} vs Closing
${summary.closingBlockedBalance}`);

// Check for reserves
const reserves = balances.filter(b => b.accountCategory.startsWith('RESERVE_'));
const totalReserves = reserves.reduce((sum, r) => sum + r.balance, 0);
console.log(`\nTotal Reserves: ${totalReserves}`);
reserves.forEach(r => {
  console.log(`  ${r.accountCategory}: ${r.balance}`);
});

// Generate reconciliation report
const reconciliationReport = {
  date: summary.summaryDate,
  status: summary.isFinalized ? 'FINALIZED' : 'PENDING',
  checks: {
    netMovement: movementMatch ? 'PASS' : 'FAIL',
    closingBalance: closingMatch ? 'PASS' : 'FAIL',
```

```
  },
  summary: {
    deposits: summary.totalDepositsGross,
    fees: summary.totalFeesCollected,
    payouts: summary.totalPayouts,
    refunds: summary.totalRefunds,
    chargebacks: summary.totalChargebacks,
    netMovement: summary.netMovement,
  },
  balances: {
    available: availableBalance?.balance || 0,
    pending: pendingBalance?.balance || 0,
    blocked: blockedBalance?.balance || 0,
    reserves: totalReserves,
  }
};

console.log('\n=== Reconciliation Report ===');
console.log(JSON.stringify(reconciliationReport, null, 2));
```

Reports

REST API / REPORTS

Generate Report

Generate a new report for transactions, payouts, settlements, or other data.

`POST` **`/merchant/api/v1/reports/generate`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`type`	`string`	Yes	Report type: `transactions`, `payouts`, `settlements`, `refunds`, `customers`
`format`	`string`	No	Output format: `csv`, `xlsx`, `pdf` (default: `csv`)
`startDate`	`string`	Yes	Start date (ISO 8601)
`endDate`	`string`	Yes	End date (ISO 8601)
`filters`	`object`	No	Additional filters based on report type
`columns`	`array`	No	Specific columns to include

Response

Success (201 Created)

```
{
  "success": true,
  "data": {
    "id": "rep_550e8400-e29b-41d4-a716-446655440000",
    "merchantId": "mer_xxxxx",
    "type": "transactions",
    "format": "csv",
    "status": "processing",
    "startDate": "2024-01-01T00:00:00.000Z",
    "endDate": "2024-01-31T23:59:59.999Z",
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```

Report Status

Status	Description
`pending`	Report queued for processing
`processing`	Report is being generated
`completed`	Report ready for download
`failed`	Report generation failed

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/reports/generate \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "type": "transactions",
    "format": "csv",
    "startDate": "2024-01-01T00:00:00.000Z",
    "endDate": "2024-01-31T23:59:59.999Z"
  }'
```

SDK

```
const report = await zenpays.reports.generate({
  type: 'transactions',
  format: 'csv',
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-01-31T23:59:59.999Z',
});

console.log(report.id); // rep_550e8400...
```

Related Endpoints

- List Reports
- Get Report
- Download Report

REST API / REPORTS

List Reports

Retrieve a paginated list of generated reports.

`GET` **`/merchant/api/v1/reports`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Query Parameters

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number
`limit`	`integer`	`20`	Items per page (max 100)
`type`	`string`	`-`	Filter by report type
`status`	`string`	`-`	Filter by status
`startDate`	`string`	`-`	Created after (ISO 8601)
`endDate`	`string`	`-`	Created before (ISO 8601)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "id": "rep_550e8400-e29b-41d4-a716-446655440000",
      "merchantId": "mer_xxxxx",
      "type": "transactions",
      "format": "csv",
      "status": "completed",
      "startDate": "2024-01-01T00:00:00.000Z",
      "endDate": "2024-01-31T23:59:59.999Z",
      "downloadUrl": "https://api.zenpays.com/merchant/api/v1/reports/rep_550e8400/download",
      "fileSize": 125000,
      "rowCount": 1500,
      "createdAt": "2024-01-15T10:30:00.000Z",
      "completedAt": "2024-01-15T10:32:00.000Z"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 45,
    "totalPages": 3
  }
}
```


Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/reports?status=completed&limit=10" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const { data, pagination } = await zenpays.reports.list({
  status: 'completed',
  limit: 10,
});

console.log(`Found ${pagination.total} reports`);
```

Related Endpoints

- Generate Report
- Get Report
- Download Report

REST API / REPORTS

Get Report

Retrieve details of a specific report by ID.

`GET` **`/merchant/api/v1/reports/:id`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Path Parameters

Parameter	Type	Description
`id`	`string`	Report ID

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "id": "rep_550e8400-e29b-41d4-a716-446655440000",
    "merchantId": "mer_xxxxx",
    "type": "transactions",
    "format": "csv",
    "status": "completed",
    "startDate": "2024-01-01T00:00:00.000Z",
    "endDate": "2024-01-31T23:59:59.999Z",
    "filters": {
      "status": "succeeded"
    },
    "downloadUrl": "https://api.zenpays.com/merchant/api/v1/reports/rep_550e8400/download",
    "fileSize": 125000,
    "rowCount": 1500,
    "createdAt": "2024-01-15T10:30:00.000Z",
    "completedAt": "2024-01-15T10:32:00.000Z"
  }
}
```

Error Responses

Code	HTTP	Message
`REPORT_NOT_FOUND`	404	Report does not exist

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/reports/rep_550e8400" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
const report = await zenpays.reports.get('rep_550e8400');

if (report.status === 'completed') {
  console.log('Report ready:', report.downloadUrl);
}
```

Related Endpoints

- Generate Report
- List Reports
- Download Report

REST API / REPORTS

Download Report

Download a completed report file.

`GET` **`/merchant/api/v1/reports/:id/download`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Path Parameters

Parameter	Type	Description
`id`	`string`	Report ID

Response

Success (200 OK)

Returns the report file with appropriate content-type header:

- `text/csv` for CSV reports
- `application/vnd.openxmlformats-officedocument.spreadsheetml.sheet` for XLSX
- `application/pdf` for PDF reports

Error Responses

Code	HTTP	Message
`REPORT_NOT_FOUND`	404	Report does not exist
`REPORT_NOT_READY`	400	Report is still processing

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/reports/rep_550e8400/download" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -o report.csv
```

SDK

```
const fileBuffer = await zenpays.reports.download('rep_550e8400');

// Save to file
fs.writeFileSync('report.csv', fileBuffer);
```

Related Endpoints

- Generate Report
- Get Report
- Delete Report

REST API / REPORTS

Delete Report

Delete a report and its associated file.

`DELETE` **`/merchant/api/v1/reports/:id`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation

Path Parameters

Parameter	Type	Description
`id`	`string`	Report ID

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "id": "rep_550e8400-e29b-41d4-a716-446655440000",
    "deleted": true
  }
}
```

Error Responses

Code	HTTP	Message
`REPORT_NOT_FOUND`	404	Report does not exist

Examples

CURL

```
curl -X DELETE "https://api.zenpays.com/merchant/api/v1/reports/rep_550e8400" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt"
```

SDK

```
await zenpays.reports.delete('rep_550e8400');
console.log('Report deleted');
```

Related Endpoints

- Generate Report
- List Reports

Ramp

REST API / RAMP

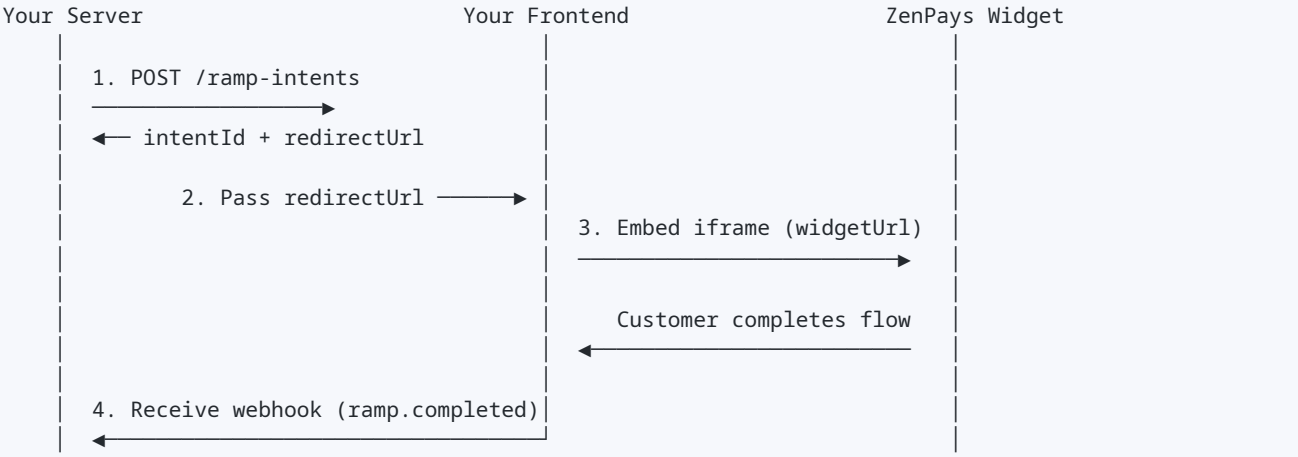
Ramp Overview

ZenPays Ramp enables your customers to buy crypto with fiat (**On-Ramp**) and sell crypto for fiat (**Off-Ramp**). There are two ways to integrate: the **Widget** (recommended) or the **Direct API**.

Integration Paths

Widget Integration (Recommended)

Embed the ZenPays hosted widget in your app via iframe. The widget handles the entire flow — KYC verification, payment method selection, provider routing, and transaction tracking — so you don't have to.

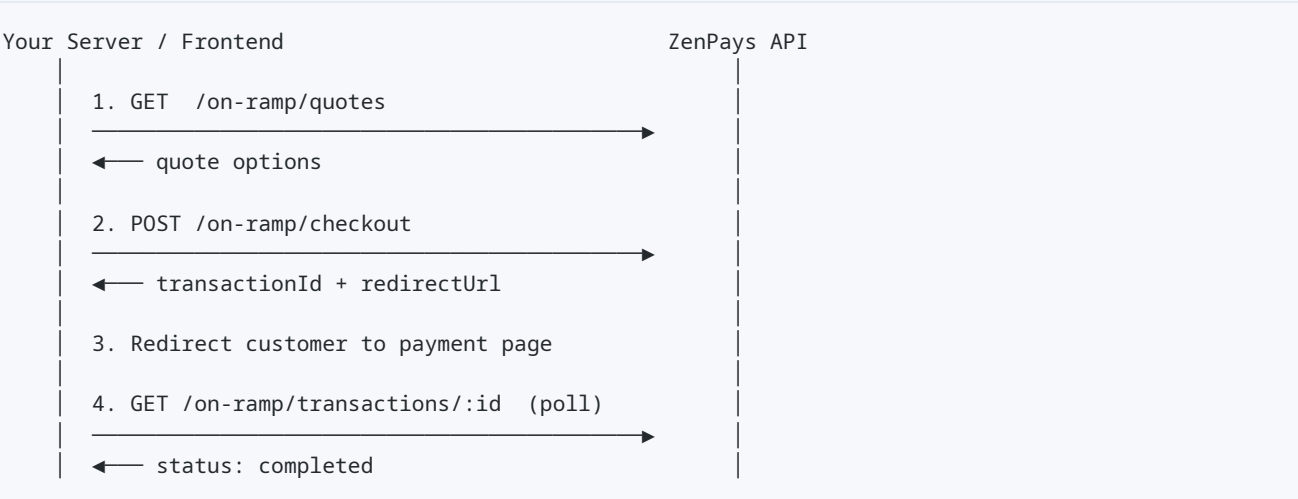


What you build: Create an intent on your server, embed the widget URL in an iframe. That's it.

What ZenPays handles: KYC, payment UI, provider selection, error handling, status tracking.

API Integration (Advanced)

Call the ramp APIs directly to build your own custom UI. You get full control over the user experience but must handle quotes, checkout creation, status polling, and error states yourself.



What you build: Your own quote display, checkout UI, payment flow, status polling, error handling.

What ZenPays handles: Provider routing, payment processing, crypto delivery.

Which Should I Choose?

	Widget	API
Integration effort	~10 lines of code	Significant frontend work
Time to go live	Hours	Days to weeks
KYC handling	Built-in	You must build or skip
Payment UI	Hosted by ZenPays	You build it
Provider selection	Automatic	You call quotes + choose
Error handling	Built-in	You handle all states
Customization	Limited (colors, defaults)	Full control
Best for	Most merchants	Exchanges, advanced UIs

> **Recommendation**
> **Start with the Widget.** It gets you to production fastest with the least maintenance. You can always migrate to the API later if you need deeper customization.

Supported Directions

Direction	Description	Use Case
On-Ramp (Buy)	Customer pays fiat, receives crypto	"Buy Bitcoin with USD"
Off-Ramp (Sell)	Customer sends crypto, receives fiat	"Sell USDT for INR"

Both directions are supported by the Widget and API integration paths.

Key Concepts

Ramp Intents

A **Ramp Intent** is a server-side session that tracks a ramp transaction from creation to completion. Intents are required for widget integration and optional (but recommended) for API integration.

- Created on your server with your API key (authenticated)
- Returns a `redirectUrl` with the widget embedded
- Expires after 1 hour
- Triggers `ramp.completed` or `ramp.failed` webhooks

Configuration Endpoints

Before building your integration, use the configuration endpoints to discover:

- **Supported Assets** — Which crypto and fiat currencies are available
- **Payment Methods** — Which payment methods work for a given currency
- **Defaults** — Default currencies and amounts per country
- **Ramp Config** — Current integration mode and feature flags

These endpoints are public (no authentication required) and useful for both Widget and API integrations.

Next Steps

- **Widget path:** Start with Generate Widget URL or Create Ramp Intent
- **API path:** Start with Buy Quotes (on-ramp) or Sell Quotes (off-ramp)
- **Examples:** See On-Ramp Flow and Off-Ramp Flow for complete walkthroughs

Widget Integration (Recommended)

REST API / RAMP / WIDGET INTEGRATION (RECOMMENDED)

Generate Widget URL

Generate a signed widget URL to embed the ramp experience in an iframe. Server-side signing ensures wallet addresses are tamper-proof. Works for both buy (on-ramp) and sell (off-ramp) modes.

`POST` **`/payment/api/v1/on-ramp/widget-url`**

`POST` **`/payment/api/v1/off-ramp/widget-url`**

The on-ramp endpoint defaults to `buy` mode; the off-ramp endpoint defaults to `sell` mode.

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`x-request-id`	No	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Body Parameters

Parameter	Type	Description
`mode`	`string`	`buy` , `sell` , or `buy,sell` (default depends on endpoint)
`defaultFiat`	`string`	Default fiat currency (e.g. `usd`)
`defaultCrypto`	`string`	Default crypto (e.g. `btc_bitcoin`)
`defaultAmount`	`number`	Default amount
`wallets`	`string`	TokenID:address format (e.g. `btc:bc1q...`)
`networkWallets`	`string`	NetworkID:address format (e.g. `ethereum:0x...`)
`walletAddressTags`	`string`	TokenID:tag format for memo coins
`successRedirectUrl`	`string`	URL to redirect on success
`failureRedirectUrl`	`string`	URL to redirect on failure
`onlyFiats`	`string`	Restrict fiat options (comma-separated)
`onlyCryptos`	`string`	Restrict crypto options (comma-separated)

Parameter	Type	Description
`onlyCryptoNetworks`	`string`	Restrict networks (comma-separated)
`email`	`string`	Pre-fill user email
`uuid`	`string`	Unique user identifier
`partnerContext`	`string`	Custom tracking context

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "widgetUrl": "https://buy.onramper.com/?
apiKey=...&mode=buy&defaultFiat=usd&defaultCrypto=BTC&wallets=btc:bc1q...&signature=abc123",
    "embedMode": "iframe"
  },
  "message": "Widget URL generated"
}
```

Response Fields

Parameter	Type	Description
`widgetUrl`	`string`	Signed URL to embed in an iframe
`embedMode`	`string`	Always `iframe`

Error Responses

Code	HTTP	Message
`RAMP_WIDGET_URL_FAILED`	500	Failed to generate buy widget URL
`OFFRAMP_WIDGET_URL_FAILED`	500	Failed to generate sell widget URL

Embedding the Widget

```
<iframe
  src="{widgetUrl}"
  height="660px"
  width="482px"
  title="Buy Crypto"
  allow="accelerometer; autoplay; camera; gyroscope; payment"
  sandbox="allow-scripts allow-same-origin allow-popups allow-forms allow-top-navigation"
  style="border: none; border-radius: 12px;"
/>
```

Examples

CURL

```
# Buy widget
curl -X POST https://api.zenpays.com/payment/api/v1/on-ramp/widget-url \
  -H "Content-Type: application/json" \
  -d '{
    "mode": "buy",
    "defaultFiat": "usd",
    "defaultCrypto": "BTC",
    "wallets": "btc:bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh",
    "successRedirectUrl": "https://yourapp.com/success"
  }'

# Sell widget
curl -X POST https://api.zenpays.com/payment/api/v1/off-ramp/widget-url \
  -H "Content-Type: application/json" \
  -d '{
    "defaultFiat": "usd",
    "defaultCrypto": "BTC"
  }'
```

JAVASCRIPT

```
// Generate buy widget URL
const response = await fetch('https://api.zenpays.com/payment/api/v1/on-ramp/widget-url', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    mode: 'buy',
    defaultFiat: 'usd',
    defaultCrypto: 'BTC',
    wallets: 'btc:bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh',
    successRedirectUrl: 'https://yourapp.com/success',
  }),
});

const { data } = await response.json();

// Embed in iframe
const iframe = document.createElement('iframe');
iframe.src = data.widgetUrl;
iframe.style.cssText = 'border:none; border-radius:12px; width:482px; height:660px;';
iframe.allow = 'accelerometer; autoplay; camera; gyroscope; payment';
document.getElementById('widget-container').appendChild(iframe);
```

PYTHON

```python
import requests

# Generate buy widget URL
response = requests.post(
    "https://api.zenpays.com/payment/api/v1/on-ramp/widget-url",
    json={
        "mode": "buy",
        "defaultFiat": "usd",
        "defaultCrypto": "BTC",
        "wallets": "btc:bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh",
        "successRedirectUrl": "https://yourapp.com/success",
    },
)

data = response.json()["data"]
print(f"Widget URL: {data['widgetUrl']}")
print(f"Embed mode: {data['embedMode']}")
```

Integration Flow (Widget Mode)

```
1. POST /on-ramp/widget-url     → Get signed widget URL
2. Embed widgetUrl in iframe     → User completes purchase inside widget
3. User redirected to success/failure URL when done
```

Next Steps

- Config -- Check integration mode (widget vs API)
- Wallet Addresses -- Understand wallet resolution
- Buy Checkout -- API mode alternative

Ramp Intents

REST API / RAMP / RAMP INTENTS

Create Ramp Intent

Create a standalone ramp intent for on-ramp (buy crypto) or off-ramp (sell crypto). A ramp intent represents a merchant-initiated ramp session that tracks the full lifecycle of a transaction — from creation through to completion or expiry.

`POST` **`/payment/api/v1/ramp-intents`**

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`Authorization`	Yes	`Bearer {api_key}`
`x-merchant-id`	Yes	Your merchant ID
`x-request-id`	Yes	Unique request ID for tracing

Authenticated Endpoint

This endpoint requires API key authentication. Include your API key in the `Authorization` header and your merchant ID in `x-merchant-id`.

Body Parameters

Parameter	Type	Required	Description
`userId`	`string`	Yes	Unique user identifier from your system. Used as the KYC record key.
`kycVerified`	`boolean`	Yes	Whether you have already verified this user's identity. When `true`, ZenPays KYC verification is skipped in the widget.
`type`	`string`	Yes	`buy` (on-ramp) or `sell` (off-ramp)
`wallets`	`array`	Yes	Array of `{ network, address }` objects (min 1)
`fiatCurrency`	`string`	Yes	Fiat currency code (e.g. `USD`, `EUR`)
`cryptoCurrency`	`string`	No	Crypto identifier (e.g. `btc_bitcoin`, `usdt_tron`)
`amount`	`number`	No	Default amount (must be positive)

Parameter	Type	Required	Description
`successUrl`	`string`	No	URL to redirect on success
`cancelUrl`	`string`	No	URL to redirect on cancel
`customerEmail`	`string`	No	Customer's email address
`metadata`	`object`	No	Custom key-value pairs for your reference

KYC Behavior

When `kycVerified: true`, the checkout widget skips ZenPays KYC verification entirely — use this when your platform has already verified the user's identity.

When `kycVerified: false` (default), the widget checks the user's KYC status. If unverified, the customer is prompted to complete identity verification before proceeding.

Wallet Object

Parameter	Type	Required	Description
`network`	`string`	Yes	Blockchain network (e.g. `ethereum`, `tron`, `solana`)
`address`	`string`	Yes	Wallet address on the network

Sample Request Body

```
{
  "userId": "usr_abc123",
  "kycVerified": false,
  "type": "buy",
  "wallets": [
    {
      "network": "ethereum",
      "address": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28"
    }
  ],
  "fiatCurrency": "USD",
  "cryptoCurrency": "eth_ethereum",
  "amount": 100,
  "successUrl": "https://yourapp.com/success",
  "cancelUrl": "https://yourapp.com/cancel",
  "customerEmail": "customer@example.com",
  "metadata": {
    "orderId": "order_123"
  }
}
```

Response

Success (201 Created)

```
{
  "success": true,
  "data": {
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "redirectUrl":
"https://checkout.zenpays.com/payments/ramping/widget/ri_1710345678000_a1b2c3d4e5f6g7h8",
    "expiresAt": "2026-03-15T10:00:00.000Z"
  },
  "message": "Ramp intent created"
}
```

Response Fields

Parameter	Type	Description
`intentId`	`string`	Unique intent identifier (format: `ri_{timestamp}_{hex}`)
`redirectUrl`	`string`	URL to redirect the customer to the ramp widget
`expiresAt`	`string`	ISO 8601 expiry timestamp (1 hour from creation)

Intent Expiry

Ramp intents automatically expire **1 hour** after creation. Once expired, the intent cannot be used and a new one must be created.

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request body or missing required fields
`RAMP_INTENT_CREATION_FAILED`	500	Failed to create the ramp intent

Examples

CURL

```
curl -X POST https://api.zenpays.com/payment/api/v1/ramp-intents \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer zp_test_your_api_key" \
  -H "x-merchant-id: merch_abc123" \
  -H "x-request-id: req_$(date +%s)" \
  -d '{
    "userId": "usr_abc123",
    "kycVerified": false,
    "type": "buy",
    "wallets": [
      { "network": "ethereum", "address": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28" }
    ],
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "customerEmail": "customer@example.com",
    "successUrl": "https://yourapp.com/success",
    "cancelUrl": "https://yourapp.com/cancel"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/payment/api/v1/ramp-intents', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer zp_test_your_api_key',
    'x-merchant-id': 'merch_abc123',
    'x-request-id': `req_${Date.now()}`,
  },
  body: JSON.stringify({
    userId: 'usr_abc123',
    kycVerified: false,
    type: 'buy',
    wallets: [
      { network: 'ethereum', address: '0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28' },
    ],
    fiatCurrency: 'USD',
    cryptoCurrency: 'eth_ethereum',
    amount: 100,
    customerEmail: 'customer@example.com',
    successUrl: 'https://yourapp.com/success',
    cancelUrl: 'https://yourapp.com/cancel',
  }),
});

const { data } = await response.json();

console.log(`Intent ID: ${data.intentId}`);
console.log(`Redirect to: ${data.redirectUrl}`);
console.log(`Expires at: ${data.expiresAt}`);

// Redirect the customer to the ramp widget
window.location.href = data.redirectUrl;
```

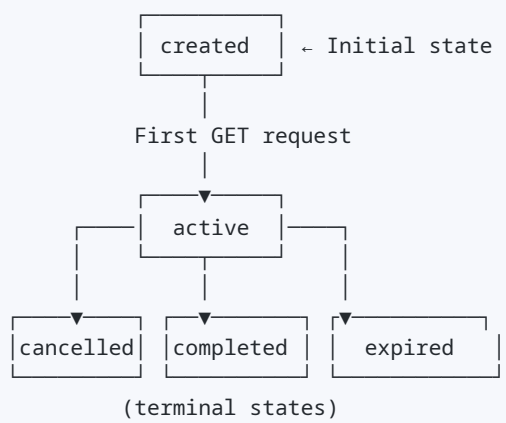
PYTHON

```python
import requests
import time

response = requests.post(
    "https://api.zenpays.com/payment/api/v1/ramp-intents",
    headers={
        "Authorization": "Bearer zp_test_your_api_key",
        "x-merchant-id": "merch_abc123",
        "x-request-id": f"req_{int(time.time())}",
    },
    json={
        "userId": "usr_abc123",
        "kycVerified": False,
        "type": "buy",
        "wallets": [
            {"network": "ethereum", "address": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28"}
        ],
        "fiatCurrency": "USD",
        "cryptoCurrency": "eth_ethereum",
        "amount": 100,
        "customerEmail": "customer@example.com",
        "successUrl": "https://yourapp.com/success",
        "cancelUrl": "https://yourapp.com/cancel",
    },
)

data = response.json()["data"]
print(f"Intent ID: {data['intentId']}")
print(f"Redirect to: {data['redirectUrl']}")
print(f"Expires at: {data['expiresAt']}")
```

Intent Status Lifecycle



Webhook Events

When a ramp intent reaches a terminal state, ZenPay delivers a webhook to your configured endpoint. Configure webhooks in the merchant dashboard under **Developer Tools → Webhooks**.

Events

Event	When	Description
`ramp.completed`	Buy: crypto sent to wallet. Sell: fiat payout submitted.	The ramp flow finished successfully.
`ramp.failed`	Crypto send or payout submission failed.	The ramp flow encountered an error.

Buy Ramp Completed (`ramp.completed`)

Delivered when crypto has been sent to the customer's wallet address.


```
{
  "event_type": "ramp.completed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "buy",
    "status": "completed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "chain": "ethereum",
    "destinationAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28",
    "fireblocksTxId": "fb_tx_9a8b7c6d5e4f",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```


Sell Ramp Completed (`ramp.completed`)

Delivered when the fiat payout has been submitted to the customer's bank account.


```
{
  "event_type": "ramp.completed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "sell",
    "status": "completed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "usdt_tron",
    "amount": 20,
    "payoutId": "po_xyz789",
    "payoutAmount": 19.50,
    "payoutCurrency": "USD",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```


Ramp Failed (`ramp.failed`)

Delivered when the crypto send (buy) or payout submission (sell) fails.

```
{
  "event_type": "ramp.failed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "buy",
    "status": "failed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "reason": "Crypto send failed",
    "failedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

Webhook Payload Fields

Parameter	Type	Description
`merchant_id`	`string`	Your merchant ID
`intentId`	`string`	Ramp intent ID (`ri_...`)
`type`	`string`	`"buy"` or `"sell"`
`status`	`string`	`"completed"` or `"failed"`
`fiatCurrency`	`string`	ISO currency code (e.g. `USD` , `EUR`)
`cryptoCurrency`	`string`	Crypto identifier (e.g. `eth_ethereum` , `usdt_tron`)
`amount`	`number`	Original intent amount
`completedAt`	`string`	ISO 8601 completion timestamp
`failedAt`	`string`	ISO 8601 failure timestamp
`reason`	`string`	Human-readable failure reason
`chain`	`string`	Blockchain network (e.g. `ethereum` , `tron`)
`destinationAddress`	`string`	Wallet address crypto was sent to
`cryptoTxId`	`string`	Crypto transaction reference
`payoutId`	`string`	Payout reference ID
`payoutAmount`	`number`	Fiat amount paid out (after fees)
`payoutCurrency`	`string`	Fiat currency of payout

Webhook Headers

Header	Description
`X-ZenPay-Signature`	HMAC-SHA256 signature of the request body
`X-Zenpay-Event`	Event type (e.g. `ramp.completed`)
`X-Zenpay-Event-Id`	Unique delivery ID for deduplication
`X-Zenpay-Timestamp`	ISO 8601 timestamp of delivery
`X-Zenpay-Attempt`	Delivery attempt number (1, 2, or 3)

See the Webhooks guide for signature verification, retry behavior, and best practices.

Next Steps

- Get Ramp Intent — Retrieve intent details and widget URL
- List Ramp Intents — Browse intent history
- On-Ramp Flow — Complete buy integration guide
- Off-Ramp Flow — Complete sell integration guide

REST API / RAMP / RAMP INTENTS

List Ramp Intents

Retrieve a paginated list of ramp intents for your merchant account. Supports filtering by status, type, and text search across intent IDs and customer emails.

`GET` **`/payment/api/v1/ramp-intents`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`x-merchant-id`	Yes	Your merchant ID
`x-request-id`	Yes	Unique request ID for tracing

Authenticated Endpoint

This endpoint requires API key authentication. Only intents belonging to the authenticated merchant are returned.

Query Parameters

Parameter	Type	Default	Description
`page`	`number`	`1`	Page number
`limit`	`number`	`20`	Results per page (max 100)
`status`	`string`	–	Filter by status: `created`, `active`, `completed`, `expired`, `cancelled`
`type`	`string`	–	Filter by type: `buy` or `sell`
`search`	`string`	–	Search by intent ID or customer email (partial match)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "data": [
      {
        "id": "a1b2c3d4-uuid",
        "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
        "type": "buy",
        "fiatCurrency": "USD",
        "cryptoCurrency": "eth_ethereum",
        "amount": 100,
        "status": "completed",
        "customerEmail": "customer@example.com",
        "wallets": [
          { "network": "ethereum", "address": "0x742d35Cc..." }
        ],
        "expiresAt": "2026-03-15T10:00:00.000Z",
        "completedAt": "2026-03-15T09:45:00.000Z",
        "createdAt": "2026-03-15T09:00:00.000Z",
        "updatedAt": "2026-03-15T09:45:00.000Z"
      }
    ],
    "meta": {
      "total": 42,
      "page": 1,
      "limit": 20,
      "totalPages": 3
    }
  },
  "message": "Ramp intents retrieved"
}
```


Response Fields

Intent Object

Parameter	Type	Description
`intentId`	`string`	Unique intent identifier
`type`	`string`	`buy` or `sell`
`fiatCurrency`	`string`	Fiat currency code
`cryptoCurrency`	`string | null`	Crypto identifier
`amount`	`number | null`	Amount
`status`	`string`	Current status
`customerEmail`	`string | null`	Customer email
`wallets`	`array`	Wallet addresses
`depositCharges`	`object | null`	Fee breakdown for deposit phase (sell intents)

Parameter	Type	Description
`payoutCharges`	`object | null`	Fee breakdown for payout phase (sell intents)
`expiresAt`	`string`	Expiry timestamp
`completedAt`	`string | null`	Completion timestamp
`createdAt`	`string`	Creation timestamp
`updatedAt`	`string`	Last update timestamp

Meta Object

Parameter	Type	Description
`total`	`number`	Total matching intents
`page`	`number`	Current page
`limit`	`number`	Results per page
`totalPages`	`number`	Total pages

Examples

CURL

```
# List all intents (page 1)
curl https://api.zenpays.com/payment/api/v1/ramp-intents \
  -H "Authorization: Bearer zp_test_your_api_key" \
  -H "x-merchant-id: merch_abc123" \
  -H "x-request-id: req_list_001"

# Filter by status and type
curl "https://api.zenpays.com/payment/api/v1/ramp-intents?
status=completed&type=sell&page=1&limit=10" \
  -H "Authorization: Bearer zp_test_your_api_key" \
  -H "x-merchant-id: merch_abc123" \
  -H "x-request-id: req_list_002"

# Search by email
curl "https://api.zenpays.com/payment/api/v1/ramp-intents?search=customer@example.com" \
  -H "Authorization: Bearer zp_test_your_api_key" \
  -H "x-merchant-id: merch_abc123" \
  -H "x-request-id: req_list_003"
```

JAVASCRIPT

```javascript
const params = new URLSearchParams({
  status: 'completed',
  type: 'sell',
  page: '1',
  limit: '10',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/ramp-intents?${params}`,
  {
    headers: {
      'Authorization': 'Bearer zp_test_your_api_key',
      'x-merchant-id': 'merch_abc123',
      'x-request-id': `req_${Date.now()}`,
    },
  }
);

const { data } = await response.json();
const { data: intents, meta } = data;

console.log(`Page ${meta.page} of ${meta.totalPages} (${meta.total} total)`);
intents.forEach((intent) => {
  console.log(`${intent.intentId} | ${intent.type} | ${intent.status}`);
});
```

PYTHON

```python
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/ramp-intents",
    headers={
        "Authorization": "Bearer zp_test_your_api_key",
        "x-merchant-id": "merch_abc123",
        "x-request-id": "req_list_001",
    },
    params={
        "status": "completed",
        "type": "sell",
        "page": 1,
        "limit": 10,
    },
)

result = response.json()["data"]
intents = result["data"]
meta = result["meta"]

print(f"Page {meta['page']} of {meta['totalPages']} ({meta['total']} total)")
for intent in intents:
    print(f"{intent['intentId']} | {intent['type']} | {intent['status']}")
```

Next Steps

- Get Ramp Intent — Retrieve full details for a specific intent
- Create Ramp Intent — Create a new ramp intent
- Update Ramp Intent Status — Manually update intent status

REST API / RAMP / RAMP INTENTS

Get Ramp Intent

Retrieve the full details of a ramp intent, including the generated widget URL for embedding. This endpoint also manages automatic status transitions — the intent moves from `created` to `active` on the first GET request, and auto-expires if past its `expiresAt` timestamp.

GET `/payment/api/v1/ramp-intents/:intentId`

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers. It is designed to be called from your frontend or checkout page.

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	The ramp intent ID (e.g. `ri_1710345678000_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "intent": {
      "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
      "type": "buy",
      "fiatCurrency": "USD",
      "cryptoCurrency": "eth_ethereum",
      "amount": 100,
      "status": "active",
      "kycStatus": "approved",
      "cancelReason": null,
      "wallets": [
        { "network": "ethereum", "address": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28" }
      ],
      "userId": "usr_abc123",
      "kycVerified": true,
      "successUrl": "https://yourapp.com/success",
      "cancelUrl": "https://yourapp.com/cancel",
      "expiresAt": "2026-03-15T10:00:00.000Z",
      "metadata": {}
    },
    "widgetUrl": "https://widget.onramper.com/...signed_url..."
  },
  "message": "Ramp intent retrieved"
}
```


Response Fields

Intent Object

Parameter	Type	Description
`intentId`	`string`	Unique intent identifier
`type`	`string`	`buy` or `sell`
`fiatCurrency`	`string`	Fiat currency code
`cryptoCurrency`	`string | null`	Crypto identifier
`amount`	`number | null`	Default amount
`status`	`string`	Lifecycle state: `created`, `active`, `completed`, `expired`, or `cancelled`
`kycStatus`	`string`	KYC sub-state: `pending`, `in_review`, `approved`, `declined`, or `expired`. When KYC is declined, `status` flips to `cancelled` and `cancelReason='kyc_rejected'`.
`cancelReason`	`string | null`	Reason the intent was cancelled. `kyc_rejected` when KYC was declined; `kyc_expired` for KYC session expiry. `null` otherwise.

Parameter	Type	Description
`successUrl`	`string | null`	Merchant success redirect URL
`cancelUrl`	`string | null`	Merchant cancel redirect URL
`expiresAt`	`string`	Expiry timestamp
`metadata`	`object`	Custom metadata
`wallets`	`array`	Array of `{ network, address }` wallet objects
`userId`	`string`	Merchant's user identifier (KYC record key)
`kycVerified`	`boolean`	Whether KYC was pre-verified by merchant. Convenience flag — equivalent to `kycStatus === 'approved'`.
`customerEmail`	`string | null`	Customer email if provided

Root Fields

Parameter	Type	Description
`widgetUrl`	`string | null`	Signed widget URL for iframe embedding. `null` if expired.
`kycRequired`	`boolean | undefined`	Whether ZenPays KYC verification is required. Present when `kycVerified` is `false` and the user has not yet been verified.
`kycVerificationUrl`	`string | null`	URL to redirect user for identity verification. `null` if no verification session has been initiated yet.

KYC Gating

The response shape depends on the user's KYC status:

Scenario 1: KYC pre-verified by merchant (`kycVerified: true` on the intent)

Widget URL is returned immediately. No KYC prompts.

Scenario 2: KYC verified via ZenPays (`kycVerified: false` , user already verified)

Widget URL is returned. `kycRequired` is absent or `false` .

Scenario 3: KYC required (`kycVerified: false` , user not verified)

`widgetUrl` is `null` , `kycRequired` is `true` . If a verification session exists, `kycVerificationUrl` contains the URL. If not, call `initiateKyc` (or the equivalent REST endpoint) to create a session.

```
{
  "success": true,
  "data": {
    "intent": {
      "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
      "type": "sell",
      "fiatCurrency": "INR",
      "status": "active",
      "wallets": [{ "network": "ethereum", "address": "0x742d..." }],
      "userId": "usr_abc123",
      "kycVerified": false,
      "expiresAt": "2026-03-15T10:00:00.000Z"
    },
    "widgetUrl": null,
    "kycRequired": true,
    "kycVerificationUrl": "https://verify.zenpays.com/session/abc123"
  },
  "message": "Ramp intent retrieved"
}
```


Automatic Behaviors

Condition	Action
Intent status is `created`	Auto-transitions to `active` on first GET
`expiresAt` has passed	Auto-transitions to `expired`, returns `widgetUrl: null`
Widget URL not yet cached	Generates and caches the widget URL
Intent in `awaiting_payout` phase	Skips widget URL generation (sell flow Phase 2)

Error Responses

Code	HTTP	Message
`NOT_FOUND`	404	Intent with the given ID does not exist

Examples

CURL

```
curl https://api.zenpays.com/payment/api/v1/ramp-intents/ri_1710345678000_a1b2c3d4e5f6g7h8
```

JAVASCRIPT

```javascript
const intentId = 'ri_1710345678000_a1b2c3d4e5f6g7h8';

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/ramp-intents/${intentId}`
);

const { data } = await response.json();
const { intent, widgetUrl } = data;

if (intent.status === 'expired') {
  console.log('Intent has expired — create a new one');
} else if (widgetUrl) {
  // Embed in an iframe
  const iframe = document.createElement('iframe');
  iframe.src = widgetUrl;
  iframe.style.width = '100%';
  iframe.style.height = '600px';
  iframe.style.border = 'none';
  document.getElementById('widget-container').appendChild(iframe);
}
```

PYTHON

```python
import requests

intent_id = "ri_1710345678000_a1b2c3d4e5f6g7h8"

response = requests.get(
    f"https://api.zenpays.com/payment/api/v1/ramp-intents/{intent_id}"
)

data = response.json()["data"]
intent = data["intent"]
widget_url = data["widgetUrl"]

print(f"Status: {intent['status']}")
print(f"Type: {intent['type']}")

if intent["status"] == "expired":
    print("Intent has expired — create a new one")
elif widget_url:
    print(f"Widget URL: {widget_url}")
```

Next Steps

- Create Ramp Intent — Create a new ramp intent
- Update Ramp Intent Status — Manually transition status
- Widget URL — Alternative widget URL generation
- On-Ramp Flow — Complete buy integration guide

REST API / RAMP / RAMP INTENTS

Get KYC Status

Poll the current KYC verification status for a ramp intent. The checkout widget calls this endpoint every 5 seconds while a customer is verifying — use the same endpoint if you embed verification yourself.

`GET` **`/payment/api/v1/ramp-intents/:intentId/kyc-status`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers. It is designed to be called from your frontend / checkout page.

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	The ramp intent ID (e.g. `ri_1710345678000_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "verified": false,
    "kycStatus": "in_review",
    "intentStatus": "active",
    "cancelReason": null
  },
  "message": "KYC status retrieved"
}
```

Response Fields

Parameter	Type	Description
`verified`	`boolean`	Convenience flag — `true` when `kycStatus === 'approved'`.
`kycStatus`	`string`	`pending`, `in_review`, `approved`, `declined`, or `expired`.
`intentStatus`	`string`	Current ramp intent lifecycle state (`created`, `active`, `completed`, `expired`, `cancelled`).
`cancelReason`	`string | null`	Set when the intent moved to `cancelled` because of KYC. `kyc_rejected` for declines, `kyc_expired` for expiry.

Understanding KYC states

`kycStatus`	What it means	What to do
`pending`	Customer hasn't completed the Didit flow yet	Keep polling. Optionally show a "verification in progress" UI.
`in_review`	Identity passed but AML or face-match was flagged for manual review	Keep polling. Surface a friendly "under review" message — this can take a few minutes.
`approved`	KYC and AML both cleared	Proceed with the checkout. Stop polling.
`declined`	Hard rejection (identity mismatch, document issue, AML rejected)	Stop polling. `intentStatus` is now `cancelled` with `cancelReason='kyc_rejected'`. Show a terminal "verification could not be completed" message.
`expired`	Verification session expired before the customer finished	Stop polling. Same handling as `declined`. Optionally offer to start a new intent.

Polling cadence

The checkout widget polls every 5 seconds with a hard ceiling of ~5 minutes (60 attempts) before showing a "still confirming your verification" message. If you implement your own polling, mirror that pattern so you don't leave customers on a spinner indefinitely.

Recommended pattern:

1. Poll every 5 seconds.
2. Stop on any of: `kycStatus === 'approved'`, `'declined'`, or `'expired'`.
3. After ~60 attempts with no terminal state, surface a non-blocking "we're still confirming your verification" message with a manual refresh.

Error Responses

Code	HTTP	Message
`NOT_FOUND`	404	Intent with the given ID does not exist

Examples

CURL

```
curl https://api.zenpays.com/payment/api/v1/ramp-intents/ri_1710345678000_a1b2c3d4e5f6g7h8/kyc-status
```

JAVASCRIPT

```
const intentId = 'ri_1710345678000_a1b2c3d4e5f6g7h8';

async function pollKycStatus() {
  const res = await fetch(
    `https://api.zenpays.com/payment/api/v1/ramp-intents/${intentId}/kyc-status`
  );
  const { data } = await res.json();

  switch (data.kycStatus) {
    case 'approved':
      console.log('KYC verified — proceed with checkout');
      return 'done';
    case 'in_review':
      console.log('Identity OK, under review — keep waiting');
      return 'continue';
    case 'declined':
    case 'expired':
      console.log(`Terminal: ${data.cancelReason}`);
      return 'stop';
    default:
      return 'continue';
  }
}

const intervalId = setInterval(async () => {
  const next = await pollKycStatus();
  if (next !== 'continue') clearInterval(intervalId);
}, 5000);
```

PYTHON

```python
import time, requests

intent_id = "ri_1710345678000_a1b2c3d4e5f6g7h8"

for _ in range(60):  # 5 minutes worth of polls
    res = requests.get(
        f"https://api.zenpays.com/payment/api/v1/ramp-intents/{intent_id}/kyc-status"
    )
    data = res.json()["data"]

    if data["kycStatus"] == "approved":
        print("KYC verified — proceed")
        break
    if data["kycStatus"] in ("declined", "expired"):
        print(f"Terminal: {data['cancelReason']}")
        break
    time.sleep(5)
else:
    print("Still confirming — show a manual refresh UI")
```

Next Steps

- Get Ramp Intent — Fetch the full intent including the widget URL
- KYC Webhooks — Get notified when KYC state changes without polling
- On-Ramp Flow — End-to-end buy integration

REST API / RAMP / RAMP INTENTS

Update Ramp Intent Status

Manually update the status of a ramp intent. This is useful for cancelling intents from your frontend or marking them as completed after verifying a transaction.

`PATCH` **`/payment/api/v1/ramp-intents/:intentId/status`**

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`x-request-id`	No	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Path Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	The ramp intent ID

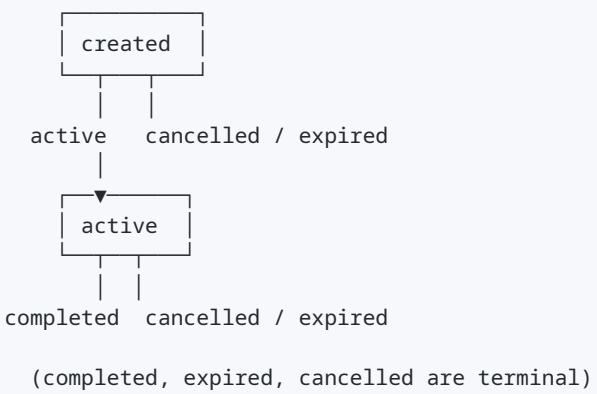
Body Parameters

Parameter	Type	Required	Description
`status`	`string`	Yes	Target status: `active`, `completed`, or `cancelled`

Status Transitions

Only certain transitions are allowed. Attempting an invalid transition returns a `400` error.

Current Status	Allowed Transitions
`created`	`active`, `cancelled`, `expired`
`active`	`completed`, `cancelled`, `expired`
`completed`	— (terminal)
`expired`	— (terminal)
`cancelled`	— (terminal)



Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "status": "cancelled"
  },
  "message": "Ramp intent status updated"
}
```

Response Fields

Parameter	Type	Description
`intentId`	`string`	The intent ID
`status`	`string`	The updated status

Error Responses

Code	HTTP	Message
`BAD_REQUEST`	400	Invalid status transition (e.g. `completed` → `active`)
`NOT_FOUND`	404	Intent with the given ID does not exist

Examples

CURL

```
# Cancel an intent
curl -X PATCH https://api.zenpays.com/payment/api/v1/ramp-
intents/ri_1710345678000_a1b2c3d4e5f6g7h8/status \
  -H "Content-Type: application/json" \
  -d '{ "status": "cancelled" }'

# Mark as completed
curl -X PATCH https://api.zenpays.com/payment/api/v1/ramp-
intents/ri_1710345678000_a1b2c3d4e5f6g7h8/status \
  -H "Content-Type: application/json" \
  -d '{ "status": "completed" }'
```

JAVASCRIPT

```
const intentId = 'ri_1710345678000_a1b2c3d4e5f6g7h8';

// Cancel an intent (e.g. user clicks "Cancel" on your checkout page)
const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/ramp-intents/${intentId}/status`,
  {
    method: 'PATCH',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ status: 'cancelled' }),
  }
);

const { data } = await response.json();
console.log(`Intent ${data.intentId} is now ${data.status}`);
```

PYTHON

```
import requests

intent_id = "ri_1710345678000_a1b2c3d4e5f6g7h8"

response = requests.patch(
    f"https://api.zenpays.com/payment/api/v1/ramp-intents/{intent_id}/status",
    json={"status": "cancelled"},
)

data = response.json()["data"]
print(f"Intent {data['intentId']} is now {data['status']}")
```

Next Steps

- Get Ramp Intent — Check current intent state
- List Ramp Intents — Browse all intents
- Create Ramp Intent — Create a new intent

REST API / RAMP

Get Transaction Status

Poll the status of an on-ramp or off-ramp transaction. Use the `transactionId` returned from a checkout response.

GET `/payment/api/v1/on-ramp/transactions/:transactionId`

GET `/payment/api/v1/off-ramp/transactions/:transactionId`

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Path Parameters

Parameter	Type	Required	Description
`transactionId`	`string`	Yes	Transaction ID from checkout response

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "transactionId": "txn_abc123",
    "status": "completed",
    "transactionType": "buy",
    "inAmount": 100,
    "outAmount": 0.00112,
    "sourceCurrency": "usd",
    "targetCurrency": "btc_bitcoin",
    "onramp": "moonpay",
    "paymentMethod": "creditcard",
    "walletAddress": "bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh",
    "transactionHash": null,
    "statusDate": "2026-03-11T04:50:00.000Z"
  },
  "message": "Transaction retrieved"
}
```

Response Fields

Parameter	Type	Description
`transactionId`	`string`	Unique transaction identifier
`status`	`string`	Current status (see table below)
`transactionType`	`string`	`buy` or `sell`
`inAmount`	`number`	Amount in source currency
`outAmount`	`number`	Amount in destination currency
`sourceCurrency`	`string`	Source currency ID
`targetCurrency`	`string`	Destination currency ID
`onramp`	`string`	Provider that processed the transaction
`paymentMethod`	`string`	Payment method used
`walletAddress`	`string`	Crypto wallet address
`transactionHash`	`string`	Blockchain transaction hash (when available)
`statusDate`	`string`	ISO 8601 timestamp of last status update

Transaction Statuses

Status	Description
`pending`	Transaction created, awaiting user action
`processing`	Payment is being processed
`completed`	Transaction completed successfully
`failed`	Transaction failed
`expired`	Transaction expired
`refunded`	Transaction was refunded

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid transaction ID
`RAMP_TRANSACTION_FAILED`	500	Failed to get on-ramp transaction
`OFFRAMP_STATUS_FAILED`	500	Failed to get off-ramp transaction

Examples

CURL

```
# On-ramp transaction
curl "https://api.zenpays.com/payment/api/v1/on-ramp/transactions/txn_abc123"

# Off-ramp transaction
curl "https://api.zenpays.com/payment/api/v1/off-ramp/transactions/txn_def456"
```

JAVASCRIPT

```
// Poll until completed or failed
async function pollTransaction(transactionId, type = 'on-ramp') {
  const maxAttempts = 60;
  const delayMs = 5000;

  for (let i = 0; i < maxAttempts; i++) {
    const response = await fetch(
      `https://api.zenpays.com/payment/api/v1/${type}/transactions/${transactionId}`
    );
    const { data } = await response.json();

    if (data.status === 'completed') {
      console.log(`Completed: ${data.inAmount} ${data.sourceCurrency} → ${data.outAmount}
${data.targetCurrency}`);
      console.log(`Provider: ${data.onramp}, Hash: ${data.transactionHash}`);
      return data;
    }

    if (data.status === 'failed' || data.status === 'expired') {
      throw new Error(`Transaction ${data.status}`);
    }

    await new Promise(resolve => setTimeout(resolve, delayMs));
  }

  throw new Error('Polling timeout');
}

await pollTransaction('txn_abc123');
```

PYTHON

```python
import requests
import time

def poll_transaction(transaction_id, ramp_type="on-ramp", max_attempts=60, delay=5):
    for _ in range(max_attempts):
        response = requests.get(
            f"https://api.zenpays.com/payment/api/v1/{ramp_type}/transactions/{transaction_id}"
        )
        data = response.json()["data"]

        if data["status"] == "completed":
            print(f"Completed: {data['inAmount']} {data['sourceCurrency']} → {data['outAmount']}
{data['targetCurrency']}")
            print(f"Provider: {data['onramp']}, Hash: {data['transactionHash']}")
            return data

        if data["status"] in ("failed", "expired"):
            raise Exception(f"Transaction {data['status']}")

        time.sleep(delay)

    raise Exception("Polling timeout")

poll_transaction("txn_abc123")
```

Next Steps

- Buy Checkout -- Create a buy checkout
- Sell Checkout -- Create a sell checkout

API Integration (Advanced)

On-Ramp (Buy)

REST API / RAMP / API INTEGRATION (ADVANCED) / ON-RAMP (BUY)

Get Buy Quotes

Get on-ramp quotes showing how much crypto you receive for a given fiat amount. Returns multiple quotes from different sources so you can compare rates and fees.

`GET` **`/payment/api/v1/on-ramp/quotes`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

Public Endpoint

This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Required	Description
`source`	`string`	Yes	Fiat currency (e.g. `usd`, `eur`)
`destination`	`string`	Yes	Crypto ID (e.g. `btc_bitcoin`, `eth_ethereum`)
`amount`	`string`	Yes	Fiat amount (e.g. `100`)
`paymentMethod`	`string`	No	Filter by payment method (e.g. `creditcard`)
`country`	`string`	No	ISO country code
`walletAddress`	`string`	No	Destination wallet address
`uuid`	`string`	No	Unique user identifier

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "onramp": "moonpay",
      "paymentMethod": "creditcard",
      "inputAmount": 100,
      "outputAmount": 0.00112,
      "rate": 89285.71,
      "networkFee": 0.5,
      "transactionFee": 3.99,
      "totalFee": 4.49,
      "recommended": true,
      "labels": ["best_rate"],
      "minAmount": 20,
      "maxAmount": 10000,
      "quoteId": "01H985NH79FW951SKERQ45JMYXmoonpay",
      "availablePaymentMethods": [
        { "paymentTypeId": "creditcard", "name": "Credit Card", "icon":
"https://cdn.onramper.com/icons/payments/creditcard.svg" },
        { "paymentTypeId": "banktransfer", "name": "Bank Transfer", "icon":
"https://cdn.onramper.com/icons/payments/banktransfer.svg" }
      ]
    },
    {
      "onramp": "transak",
      "paymentMethod": "creditcard",
      "inputAmount": 100,
      "outputAmount": 0.00110,
      "rate": 90909.09,
      "networkFee": 1.0,
      "transactionFee": 4.50,
      "totalFee": 5.50,
      "recommended": false,
      "labels": [],
      "minAmount": 30,
      "maxAmount": 5000,
      "quoteId": "01H985NH79FW951SKERQ45JMYXtransak",
      "availablePaymentMethods": [
        { "paymentTypeId": "creditcard", "name": "Credit Card", "icon":
"https://cdn.onramper.com/icons/payments/creditcard.svg" }
      ]
    }
  ],
  "message": "Quotes retrieved"
}
```


Response Fields

Parameter	Type	Description
`onramp`	`string`	Reference identifier -- pass this to the checkout endpoint
`paymentMethod`	`string`	Payment method for this quote
`inputAmount`	`number`	Fiat amount you pay

Parameter	Type	Description
`outputAmount`	`number`	Crypto amount you receive
`rate`	`number`	Exchange rate
`networkFee`	`number`	Blockchain network fee
`transactionFee`	`number`	Processing fee
`totalFee`	`number`	Combined fees
`recommended`	`boolean`	Whether this is the recommended quote
`labels`	`string[]`	Tags like `best_rate`, `low_kyc`
`minAmount`	`number`	Minimum allowed amount
`maxAmount`	`number`	Maximum allowed amount
`estimatedTime`	`string`	Estimated completion time
`errors`	`string[]`	Any issues with this quote
`quoteId`	`string`	Unique quote identifier for reference
`availablePaymentMethods`	`array`	Payment methods available for this quote. Each entry has `paymentTypeId`, `name`, and `icon`

Tip

The `onramp` field is a reference identifier. When creating a checkout, pass the `onramp` value from your selected quote.

Error Responses

Code	HTTP	Message
`RAMP_QUOTES_FAILED`	500	Failed to get buy quotes

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/on-ramp/quotes?
source=usd&destination=btc_bitcoin&amount=100&paymentMethod=creditcard"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  source: 'usd',
  destination: 'btc_bitcoin',
  amount: '100',
  paymentMethod: 'creditcard',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/on-ramp/quotes?${params}`
);
const { data: quotes } = await response.json();

// Find the recommended quote
const best = quotes.find(q => q.recommended) || quotes[0];
console.log(`Best rate: ${best.outputAmount} BTC for $${best.inputAmount}`);
console.log(`Fees: $${best.totalFee}`);

// Use best.onramp when creating checkout
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/on-ramp/quotes",
    params={
        "source": "usd",
        "destination": "btc_bitcoin",
        "amount": "100",
        "paymentMethod": "creditcard",
    },
)
quotes = response.json()["data"]

# Find recommended quote
best = next((q for q in quotes if q["recommended"]), quotes[0])
print(f"Best rate: {best['outputAmount']} BTC for ${best['inputAmount']}")
print(f"Fees: ${best['totalFee']}")

# Use best["onramp"] when creating checkout
```

Next Steps

- Buy Checkout -- Create a checkout with the selected quote
- Buy Rates -- Get rates for multiple assets at once
- Buy Prices -- Get current crypto prices

REST API / RAMP / API INTEGRATION (ADVANCED) / ON-RAMP (BUY)

Get Buy Rates

Get exchange rates for multiple crypto destinations at once. Returns the best available rate per destination -- simpler than quotes when you just need pricing.

`GET` **`/payment/api/v1/on-ramp/rates`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Required	Description
`source`	`string`	Yes	Fiat currency (e.g. `usd`)
`destinations`	`string`	Yes	Comma-separated crypto IDs (e.g. `btc_bitcoin,eth_ethereum`)
`paymentMethod`	`string`	No	Filter by payment method
`country`	`string`	No	ISO country code
`amount`	`string`	No	Reference amount (default: `100`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "source": "usd",
      "destination": "btc_bitcoin",
      "rate": 89285.71,
      "paymentMethod": "creditcard",
      "networkFee": 0.5,
      "transactionFee": 3.99,
      "minAmount": 20,
      "maxAmount": 10000,
      "timestamp": "2026-03-11T04:44:00.000Z"
    },
    {
      "source": "usd",
      "destination": "eth_ethereum",
      "rate": 3200.50,
      "paymentMethod": "creditcard",
      "networkFee": 0.3,
      "transactionFee": 3.99,
      "minAmount": 20,
      "maxAmount": 10000,
      "timestamp": "2026-03-11T04:44:00.000Z"
    }
  ],
  "message": "Rates retrieved"
}
```


Response Fields

Parameter	Type	Description
`source`	`string`	Fiat currency
`destination`	`string`	Crypto asset ID
`rate`	`number`	Exchange rate
`paymentMethod`	`string`	Best payment method for this rate
`networkFee`	`number`	Blockchain network fee
`transactionFee`	`number`	Processing fee
`minAmount`	`number`	Minimum allowed amount
`maxAmount`	`number`	Maximum allowed amount
`timestamp`	`string`	ISO 8601 timestamp

Error Responses

Code	HTTP	Message
`RAMP_RATES_FAILED`	500	Failed to get buy rates

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/on-ramp/rates?
source=usd&destinations=btc_bitcoin,eth_ethereum,usdt_tron"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  source: 'usd',
  destinations: 'btc_bitcoin,eth_ethereum,usdt_tron',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/on-ramp/rates?${params}`
);
const { data: rates } = await response.json();

rates.forEach(r => {
  console.log(`${r.destination}: ${r.rate} (fee: $${r.networkFee + r.transactionFee})`);
});
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/on-ramp/rates",
    params={
        "source": "usd",
        "destinations": "btc_bitcoin,eth_ethereum,usdt_tron",
    },
)
rates = response.json()["data"]

for r in rates:
    print(f"{r['destination']}: {r['rate']} (fee: ${r['networkFee'] + r['transactionFee']})")
```

Next Steps

- Buy Quotes -- Get detailed quotes with multiple options per pair
- Buy Prices -- Get simple price-per-unit

REST API / RAMP / API INTEGRATION (ADVANCED) / ON-RAMP (BUY)

Get Buy Prices

Get the current price of crypto assets in fiat. Returns the price per 1 unit of each crypto asset.

`GET` **`/payment/api/v1/on-ramp/prices`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Required	Description
`assets`	`string`	Yes	Comma-separated crypto IDs (e.g. `btc_bitcoin,eth_ethereum`)
`fiat`	`string`	Yes	Fiat currency (e.g. `usd`)
`country`	`string`	No	ISO country code

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "asset": "btc_bitcoin",
      "fiat": "usd",
      "price": 89285.71,
      "timestamp": "2026-03-11T04:44:00.000Z"
    },
    {
      "asset": "eth_ethereum",
      "fiat": "usd",
      "price": 3200.50,
      "timestamp": "2026-03-11T04:44:00.000Z"
    }
  ],
  "message": "Prices retrieved"
}
```

Response Fields

Parameter	Type	Description
`asset`	`string`	Crypto asset ID
`fiat`	`string`	Fiat currency
`price`	`number`	Price of 1 unit of crypto in fiat
`timestamp`	`string`	ISO 8601 timestamp

Error Responses

Code	HTTP	Message
`RAMP_PRICES_FAILED`	500	Failed to get prices

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/on-ramp/prices?
assets=btc_bitcoin,eth_ethereum&fiat=usd"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  assets: 'btc_bitcoin,eth_ethereum',
  fiat: 'usd',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/on-ramp/prices?${params}`
);
const { data: prices } = await response.json();

prices.forEach(p => {
  console.log(`${p.asset}: $${p.price.toLocaleString()}`);
});
// btc_bitcoin: $89,285.71
// eth_ethereum: $3,200.50
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/on-ramp/prices",
    params={"assets": "btc_bitcoin,eth_ethereum", "fiat": "usd"},
)
prices = response.json()["data"]

for p in prices:
    print(f"{p['asset']}: ${p['price']:,.2f}")
```

Next Steps

- Buy Quotes -- Get detailed quotes with fees
- Buy Rates -- Get rates with fee breakdown

REST API / RAMP / API INTEGRATION (ADVANCED) / ON-RAMP (BUY)

Create Buy Checkout

Create a checkout intent to execute an on-ramp (fiat to crypto) transaction. Pass the `onramp` value from a quote response to select which quote to use.

`POST` **`/payment/api/v1/on-ramp/checkout`**

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`x-request-id`	No	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Body Parameters

Parameter	Type	Required	Description
`onramp`	`string`	Yes	Identifier from quote response (e.g. `moonpay`)
`source`	`string`	Yes	Fiat currency (e.g. `usd`)
`destination`	`string`	Yes	Crypto ID (e.g. `btc_bitcoin`)
`amount`	`number`	Yes	Fiat amount
`type`	`string`	Yes	`buy`
`paymentMethod`	`string`	Yes	Payment method (e.g. `creditcard`)
`walletAddress`	`string`	No	Crypto wallet address for delivery. See Wallet Addresses
`walletMemo`	`string`	No	Memo/tag for XRP, XLM, etc.
`network`	`string`	No	Blockchain network (e.g. `ethereum`)
`country`	`string`	No	ISO country code
`sessionId`	`string`	No	Checkout session ID for tracking

Parameter	Type	Required	Description
`walletAddresses`	`object`	No	Network-to-address mappings (e.g. `{ "ethereum": "0x..." }`)

Wallet Resolution

If `walletAddress` is not provided but `sessionId` is, ZenPays automatically resolves the wallet from your merchant settings. See Wallet Addresses for details.

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "transactionId": "txn_abc123",
    "redirectUrl": "https://checkout.example.com/...",
    "status": "pending"
  },
  "message": "Checkout intent created"
}
```

Response Fields

Parameter	Type	Description
`transactionId`	`string`	Unique transaction identifier
`redirectUrl`	`string`	URL to redirect the user to complete payment
`status`	`string`	Initial status (`pending`)

After receiving the response, redirect the user to `redirectUrl` to complete payment.

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request body
`RAMP_CHECKOUT_FAILED`	500	Failed to create checkout intent

Examples

CURL

```
curl -X POST https://api.zenpays.com/payment/api/v1/on-ramp/checkout \
  -H "Content-Type: application/json" \
  -d '{
    "onramp": "moonpay",
    "source": "usd",
    "destination": "btc_bitcoin",
    "amount": 100,
    "type": "buy",
    "paymentMethod": "creditcard",
    "walletAddress": "bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/payment/api/v1/on-ramp/checkout', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    onramp: 'moonpay',
    source: 'usd',
    destination: 'btc_bitcoin',
    amount: 100,
    type: 'buy',
    paymentMethod: 'creditcard',
    walletAddress: 'bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh',
  }),
});

const { data } = await response.json();

// Redirect user to complete payment
window.location.href = data.redirectUrl;
```

PYTHON

```
import requests

response = requests.post(
    "https://api.zenpays.com/payment/api/v1/on-ramp/checkout",
    json={
        "onramp": "moonpay",
        "source": "usd",
        "destination": "btc_bitcoin",
        "amount": 100,
        "type": "buy",
        "paymentMethod": "creditcard",
        "walletAddress": "bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh",
    },
)

data = response.json()["data"]
print(f"Transaction: {data['transactionId']}")
print(f"Redirect to: {data['redirectUrl']}")
```

Full Integration Flow

```
1. GET  /on-ramp/supported-assets   → Populate currency dropdowns
2. GET  /on-ramp/payment-methods/usd → Get available payment options
3. GET  /on-ramp/quotes?source=usd&destination=btc_bitcoin&amount=100
                                    → Show pricing options to user
4. POST /on-ramp/checkout           → Create checkout with selected quote
5. Redirect to redirectUrl           → User completes payment
6. GET  /on-ramp/transactions/{id}   → Poll until completed or failed
```

Intent-Based Buy Checkout (Recommended)

For merchants using ramp intents, use this endpoint to create a buy checkout tied to an existing intent.

Endpoint

```
POST /payment/api/v1/on-ramp/buy/checkout
```

Body Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Ramp intent ID
`cryptoCurrency`	`string`	Yes	Crypto to buy (e.g. `USDT`)
`chain`	`string`	Yes	Blockchain network (e.g. `tron`, `ethereum`)
`fiatCurrency`	`string`	Yes	Fiat currency (e.g. `USD`)
`fiatAmount`	`number`	Yes	Amount to pay in fiat
`cryptoAmount`	`number`	Yes	Expected crypto amount to receive
`rate`	`number`	Yes	Locked exchange rate
`totalFee`	`number`	Yes	Total fees
`quoteId`	`string`	No	Quote ID from quotes endpoint

Sample Request

```
{
  "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
  "cryptoCurrency": "USDT",
  "chain": "tron",
  "fiatCurrency": "USD",
  "fiatAmount": 100,
  "cryptoAmount": 98.5,
  "rate": 1.0,
  "totalFee": 1.5
}
```

Response (200 OK)


```
{
  "success": true,
  "data": {
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "paymentIntentId": "pi_abc123def456",
    "checkoutUrl": "https://checkout.zenpays.com/pay/pi_abc123def456",
    "fiatAmount": 100,
    "fiatCurrency": "USD",
    "cryptoAmount": 98.5,
    "cryptoCurrency": "USDT",
    "chain": "tron",
    "rate": 1.0,
    "expiresAt": "2026-03-15T10:00:00.000Z"
  },
  "message": "Buy checkout created"
}
```


Response Fields

Parameter	Type	Description
`intentId`	`string`	Ramp intent ID
`paymentIntentId`	`string`	Payment intent for the fiat charge
`checkoutUrl`	`string`	URL to redirect customer for payment
`fiatAmount`	`number`	Fiat amount to charge
`fiatCurrency`	`string`	Fiat currency
`cryptoAmount`	`number`	Crypto amount customer will receive
`cryptoCurrency`	`string`	Crypto currency
`chain`	`string`	Blockchain network
`rate`	`number`	Exchange rate applied
`expiresAt`	`string`	Checkout expiry (ISO 8601)

Next Steps

- Transaction Status -- Poll transaction progress
- Buy Quotes -- Get quotes before checkout
- Wallet Addresses -- Understand wallet resolution

Off-Ramp (Sell)

REST API / RAMP / API INTEGRATION (ADVANCED) / OFF-RAMP (SELL)

Get Sell Quotes

Get off-ramp quotes showing how much fiat you receive for a given crypto amount. Returns multiple quotes so you can compare rates and fees.

`GET` **`/payment/api/v1/off-ramp/quotes`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

Public Endpoint

This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Required	Description
`source`	`string`	Yes	Crypto ID (e.g. `btc_bitcoin`)
`destination`	`string`	Yes	Fiat currency (e.g. `usd`)
`amount`	`string`	Yes	Crypto amount (e.g. `0.01`)
`paymentMethod`	`string`	No	Payout method (e.g. `bank_transfer`)
`country`	`string`	No	ISO country code

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "onramp": "moonpay",
      "paymentMethod": "bank_transfer",
      "inputAmount": 0.01,
      "outputAmount": 870.50,
      "rate": 89285.71,
      "networkFee": 2.00,
      "transactionFee": 5.00,
      "totalFee": 7.00,
      "recommended": true,
      "labels": [],
      "minAmount": 0.001,
      "maxAmount": 1.0,
      "quoteId": "01H985NH79FW951SKERQ45JMYXmoonpay",
      "availablePaymentMethods": [
        { "paymentTypeId": "bank_transfer", "name": "Bank Transfer", "icon":
"https://cdn.onramper.com/icons/payments/banktransfer.svg" },
        { "paymentTypeId": "creditcard", "name": "Credit Card", "icon":
"https://cdn.onramper.com/icons/payments/creditcard.svg" }
      ]
    }
  ],
  "message": "Sell quotes retrieved"
}
```


Response Fields

Parameter	Type	Description
`onramp`	`string`	Reference identifier -- pass this to the sell checkout endpoint
`paymentMethod`	`string`	Payout method for this quote
`inputAmount`	`number`	Crypto amount you sell
`outputAmount`	`number`	Fiat amount you receive
`rate`	`number`	Exchange rate
`networkFee`	`number`	Blockchain network fee
`transactionFee`	`number`	Processing fee
`totalFee`	`number`	Combined fees
`recommended`	`boolean`	Whether this is the recommended quote
`labels`	`string[]`	Tags
`minAmount`	`number`	Minimum allowed amount
`maxAmount`	`number`	Maximum allowed amount

Parameter	Type	Description
`estimatedTime`	`string`	Estimated completion time
`errors`	`string[]`	Any issues with this quote
`quoteId`	`string`	Unique quote identifier for reference
`availablePaymentMethods`	`array`	Payment methods available for this quote. Each entry has `paymentTypeId`, `name`, and `icon`

Error Responses

Code	HTTP	Message
`OFFRAMP_QUOTES_FAILED`	500	Failed to get sell quotes

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/off-ramp/quotes?
source=btc_bitcoin&destination=usd&amount=0.01"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  source: 'btc_bitcoin',
  destination: 'usd',
  amount: '0.01',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/off-ramp/quotes?${params}`
);
const { data: quotes } = await response.json();

const best = quotes.find(q => q.recommended) || quotes[0];
console.log(`You receive: $${best.outputAmount} for ${best.inputAmount} BTC`);
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/off-ramp/quotes",
    params={
        "source": "btc_bitcoin",
        "destination": "usd",
        "amount": "0.01",
    },
)
quotes = response.json()["data"]

best = next((q for q in quotes if q["recommended"]), quotes[0])
print(f"You receive: ${best['outputAmount']} for {best['inputAmount']} BTC")
```

Next Steps

- Sell Checkout -- Create a sell checkout with the selected quote
- Sell Rates -- Get rates for multiple fiat destinations

REST API / RAMP / API INTEGRATION (ADVANCED) / OFF-RAMP (SELL)

Get Sell Rates

Get off-ramp exchange rates for multiple fiat destinations at once. Returns the best available rate per destination.

`GET` **`/payment/api/v1/off-ramp/rates`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Required	Description
`source`	`string`	Yes	Crypto ID (e.g. `btc_bitcoin`)
`destinations`	`string`	Yes	Comma-separated fiat currencies (e.g. `usd,eur,gbp`)
`paymentMethod`	`string`	No	Filter by payout method
`country`	`string`	No	ISO country code
`amount`	`string`	No	Reference amount (default: `100`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "source": "btc_bitcoin",
      "destination": "usd",
      "rate": 89285.71,
      "paymentMethod": "bank_transfer",
      "networkFee": 2.00,
      "transactionFee": 5.00,
      "minAmount": 0.001,
      "maxAmount": 1.0,
      "timestamp": "2026-03-11T04:44:00.000Z"
    },
    {
      "source": "btc_bitcoin",
      "destination": "eur",
      "rate": 82150.30,
      "paymentMethod": "bank_transfer",
      "networkFee": 2.00,
      "transactionFee": 4.50,
      "minAmount": 0.001,
      "maxAmount": 1.0,
      "timestamp": "2026-03-11T04:44:00.000Z"
    }
  ],
  "message": "Sell rates retrieved"
}
```


Response Fields

Parameter	Type	Description
`source`	`string`	Crypto asset ID
`destination`	`string`	Fiat currency
`rate`	`number`	Exchange rate
`paymentMethod`	`string`	Best payout method for this rate
`networkFee`	`number`	Blockchain network fee
`transactionFee`	`number`	Processing fee
`minAmount`	`number`	Minimum allowed amount
`maxAmount`	`number`	Maximum allowed amount
`timestamp`	`string`	ISO 8601 timestamp

Error Responses

Code	HTTP	Message
`OFFRAMP_RATES_FAILED`	500	Failed to get sell rates

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/off-ramp/rates?
source=btc_bitcoin&destinations=usd,eur,gbp"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  source: 'btc_bitcoin',
  destinations: 'usd,eur,gbp',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/off-ramp/rates?${params}`
);
const { data: rates } = await response.json();

rates.forEach(r => {
  console.log(`BTC → ${r.destination.toUpperCase()}: ${r.rate}`);
});
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/off-ramp/rates",
    params={
        "source": "btc_bitcoin",
        "destinations": "usd,eur,gbp",
    },
)
rates = response.json()["data"]

for r in rates:
    print(f"BTC → {r['destination'].upper()}: {r['rate']}")
```

Next Steps

- Sell Quotes -- Get detailed quotes with multiple options
- Sell Checkout -- Create a sell checkout

REST API / RAMP / API INTEGRATION (ADVANCED) / OFF-RAMP (SELL)

Create Sell Checkout

Create a checkout intent to execute an off-ramp (crypto to fiat) transaction. Pass the `onramp` value from a sell quote response.

`POST` **`/payment/api/v1/off-ramp/checkout`**

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`x-request-id`	No	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Body Parameters

Parameter	Type	Required	Description
`onramp`	`string`	Yes	Identifier from sell quote response
`source`	`string`	Yes	Crypto ID (e.g. `btc_bitcoin`)
`destination`	`string`	Yes	Fiat currency (e.g. `usd`)
`amount`	`number`	Yes	Crypto amount to sell
`type`	`string`	Yes	`sell`
`paymentMethod`	`string`	Yes	Payout method (e.g. `bank_transfer`)
`walletAddress`	`string`	No	Source wallet address. See Wallet Addresses
`walletMemo`	`string`	No	Memo/tag for XRP, XLM, etc.
`network`	`string`	No	Blockchain network
`country`	`string`	No	ISO country code
`sessionId`	`string`	No	Checkout session ID for tracking
`walletAddresses`	`object`	No	Network-to-address mappings

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "transactionId": "txn_def456",
    "redirectUrl": "https://checkout.example.com/...",
    "status": "pending"
  },
  "message": "Sell checkout intent created"
}
```

Response Fields

Parameter	Type	Description
`transactionId`	`string`	Unique transaction identifier
`redirectUrl`	`string`	URL to redirect the user to complete the sell
`status`	`string`	Initial status (`pending`)

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request body
`OFFRAMP_CHECKOUT_FAILED`	500	Failed to create sell checkout

Examples

CURL

```
curl -X POST https://api.zenpays.com/payment/api/v1/off-ramp/checkout \
  -H "Content-Type: application/json" \
  -d '{
    "onramp": "moonpay",
    "source": "btc_bitcoin",
    "destination": "usd",
    "amount": 0.01,
    "type": "sell",
    "paymentMethod": "bank_transfer",
    "walletAddress": "bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh"
  }'
```

JAVASCRIPT

```javascript
const response = await fetch('https://api.zenpays.com/payment/api/v1/off-ramp/checkout', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    onramp: 'moonpay',
    source: 'btc_bitcoin',
    destination: 'usd',
    amount: 0.01,
    type: 'sell',
    paymentMethod: 'bank_transfer',
    walletAddress: 'bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh',
  }),
});

const { data } = await response.json();
window.location.href = data.redirectUrl;
```

PYTHON

```python
import requests

response = requests.post(
    "https://api.zenpays.com/payment/api/v1/off-ramp/checkout",
    json={
        "onramp": "moonpay",
        "source": "btc_bitcoin",
        "destination": "usd",
        "amount": 0.01,
        "type": "sell",
        "paymentMethod": "bank_transfer",
        "walletAddress": "bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh",
    },
)

data = response.json()["data"]
print(f"Transaction: {data['transactionId']}")
print(f"Redirect to: {data['redirectUrl']}")
```

Next Steps

- Transaction Status -- Poll transaction progress
- Sell Quotes -- Get quotes before checkout
- Wallet Addresses -- Understand wallet resolution

REST API / RAMP / API INTEGRATION (ADVANCED) / OFF-RAMP (SELL)

Create Sell Deposit (Phase 1)

Phase 1 of the two-phase off-ramp (sell) flow. Generates a crypto deposit address where the customer sends their crypto. The system monitors the address for incoming deposits and confirms receipt automatically.

This endpoint is **idempotent** — calling it again with the same `intentId` returns the existing deposit instead of creating a new one.

`POST` **`/payment/api/v1/off-ramp/sell/checkout`**

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`x-request-id`	No	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers. It is designed to be called from your checkout or widget page.

Body Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Ramp intent ID (from Create Ramp Intent)
`cryptoCurrency`	`string`	Yes	Crypto to deposit (e.g. `USDT`)
`chain`	`string`	Yes	Blockchain network (e.g. `tron`, `ethereum`)
`fiatCurrency`	`string`	Yes	Target fiat currency (e.g. `USD`)
`cryptoAmount`	`number`	Yes	Amount of crypto to send
`fiatAmount`	`number`	Yes	Expected fiat payout (from quote)
`rate`	`number`	Yes	Locked exchange rate (from quote)
`totalFee`	`number`	Yes	Total fees (from quote)
`quoteId`	`string`	No	Reference to the quote used
`onramp`	`string`	No	Quote provider identifier

Use Quote Values

Pass the `rate`, `fiatAmount`, and `totalFee` directly from a Sell Quote response to ensure consistent pricing.

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "depositAddress": "TLfMkVBKQSy7gzP7x9URJp6ZLXyWGZbwDs",
    "memo": null,
    "currency": "USDT",
    "chain": "tron",
    "expectedCryptoAmount": 20,
    "fiatAmount": 19.78,
    "fiatCurrency": "USD",
    "rate": 0.999,
    "cryptoDepositId": "d4e5f6a7-b8c9-1234-5678-abcdef012345",
    "expiresAt": "2026-03-15T10:00:00.000Z",
  },
  "message": "Sell checkout created"
}
```


Response Fields

Parameter	Type	Description
`intentId`	`string`	The ramp intent ID
`depositAddress`	`string`	Crypto address to send funds to
`memo`	`string | null`	Memo/tag (required for XRP, XLM, etc.)
`currency`	`string`	Crypto currency (e.g. `USDT`)
`chain`	`string`	Blockchain network
`expectedCryptoAmount`	`number`	Amount the customer should send
`fiatAmount`	`number`	Expected fiat payout
`fiatCurrency`	`string`	Fiat currency code
`rate`	`number`	Locked exchange rate
`cryptoDepositId`	`string`	Deposit tracking ID (needed for Phase 2)
`expiresAt`	`string`	Deposit expiry timestamp

Save the cryptoDepositId

You will need `cryptoDepositId` when submitting the payout in Phase 2. Store it alongside the `intentId`.

Error Responses

Code	HTTP	Message
`BAD_REQUEST`	400	Invalid or inactive intent
`SELL_CHECKOUT_FAILED`	500	Failed to generate deposit address

Examples

CURL

```
curl -X POST https://api.zenpays.com/payment/api/v1/off-ramp/sell/checkout \
  -H "Content-Type: application/json" \
  -d '{
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "cryptoCurrency": "USDT",
    "chain": "tron",
    "fiatCurrency": "USD",
    "cryptoAmount": 20,
    "fiatAmount": 19.78,
    "rate": 0.999,
    "totalFee": 0.62
  }'
```

JAVASCRIPT

```
const response = await fetch(
  'https://api.zenpays.com/payment/api/v1/off-ramp/sell/checkout',
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
      cryptoCurrency: 'USDT',
      chain: 'tron',
      fiatCurrency: 'USD',
      cryptoAmount: 20,
      fiatAmount: 19.78,
      rate: 0.999,
      totalFee: 0.62,
    }),
  }
);

const { data } = await response.json();

console.log(`Send ${data.expectedCryptoAmount} ${data.currency} to:`);
console.log(`Address: ${data.depositAddress}`);
if (data.memo) console.log(`Memo: ${data.memo}`);
console.log(`Chain: ${data.chain}`);
console.log(`Deposit ID: ${data.cryptoDepositId}`);

// Display to customer with QR code, countdown timer, etc.
```

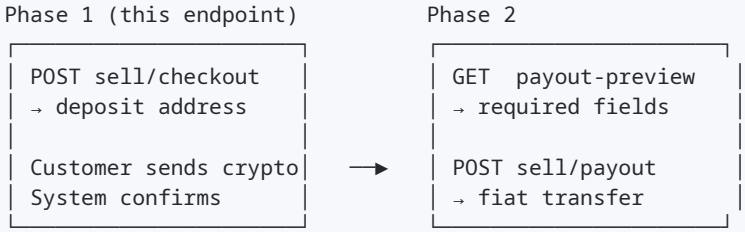
PYTHON

```python
import requests

response = requests.post(
    "https://api.zenpays.com/payment/api/v1/off-ramp/sell/checkout",
    json={
        "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
        "cryptoCurrency": "USDT",
        "chain": "tron",
        "fiatCurrency": "USD",
        "cryptoAmount": 20,
        "fiatAmount": 19.78,
        "rate": 0.999,
        "totalFee": 0.62,
    },
)

data = response.json()["data"]
print(f"Send {data['expectedCryptoAmount']} {data['currency']} to:")
print(f"Address: {data['depositAddress']}")
print(f"Chain: {data['chain']}")
print(f"Deposit ID: {data['cryptoDepositId']}")
```

Two-Phase Sell Flow



Next Steps

- Get Payout Preview — Discover required beneficiary fields (Phase 2a)
- Submit Sell Payout — Execute the fiat payout (Phase 2b)
- Off-Ramp Flow — Complete sell integration guide

REST API / RAMP / API INTEGRATION (ADVANCED) / OFF-RAMP (SELL)

Get Payout Preview (Phase 2a)

Before submitting a sell payout, call this endpoint to discover which beneficiary fields are required for the destination currency and country. The response contains dynamic form field definitions that your frontend should render for the customer to fill in.

`GET` **`/payment/api/v1/off-ramp/sell/payout-preview`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Required	Default	Description
`intentId`	`string`	Yes	–	Ramp intent ID
`fiatCurrency`	`string`	Yes	–	Target fiat currency (e.g. `USD`)
`country`	`string`	No	`US`	ISO country code for the payout destination
`payoutType`	`string`	No	`bank_transfer`	Payout method

> **Prerequisite**
> The crypto deposit for this intent must be in `confirmed` or `matched` status. If the deposit hasn't been confirmed yet, this endpoint returns a `400` error.

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "selectedTsp": "zenpay_routing",
    "tspDisplayName": "ZenPays",
    "requiredFields": [
      {
        "fieldName": "receiverFirstName",
        "label": "First Name",
        "type": "text",
        "required": true,
        "placeholder": "Enter first name"
      },
      {
        "fieldName": "receiverLastName",
        "label": "Last Name",
        "type": "text",
        "required": true,
        "placeholder": "Enter last name"
      },
      {
        "fieldName": "receiverAccountNumber",
        "label": "Account Number",
        "type": "text",
        "required": true,
        "placeholder": "Enter account number"
      },
      {
        "fieldName": "receiverCountry",
        "label": "Country",
        "type": "text",
        "required": true,
        "placeholder": "ISO country code (e.g. US)"
      },
      {
        "fieldName": "receiverBankName",
        "label": "Bank Name",
        "type": "text",
        "required": true,
        "placeholder": "Enter bank name"
      }
    ],
    "optionalFields": [
      {
        "fieldName": "receiverEmail",
        "label": "Email",
        "type": "email",
        "required": false,
        "placeholder": "Enter email address"
      },
      {
        "fieldName": "receiverPhone",
        "label": "Phone",
        "type": "tel",
        "required": false,
        "placeholder": "Enter phone number"
      }
    ],
    "oneOfGroups": [
```

```
      ["receiverBankCode", "receiverBankName"]
    ],
    "fiatAmount": 19.78,
    "fiatCurrency": "USD",
    "fees": {
      "platform": 0.62,
      "network": 0,
      "total": 0.62
    }
  },
  "message": "Payout preview retrieved"
}
```


Response Fields

Parameter	Type	Description
`selectedTsp`	`string`	Selected payout provider
`tspDisplayName`	`string`	Human-readable provider name
`requiredFields`	`array`	Fields that must be provided
`optionalFields`	`array`	Fields that can optionally be provided
`oneOfGroups`	`array | null`	Groups where at least one field must be filled
`fiatAmount`	`number`	Calculated fiat payout amount
`fiatCurrency`	`string`	Fiat currency
`fees`	`object`	Fee breakdown

Field Definition Object

Parameter	Type	Description
`fieldName`	`string`	Key to use in `beneficiaryDetails` when submitting payout
`label`	`string`	Human-readable label for the form field
`type`	`string`	Input type: `text`, `email`, `tel`, `number`, or `select`
`required`	`boolean`	Whether the field is mandatory
`placeholder`	`string`	Placeholder text
`pattern`	`string | null`	Regex validation pattern
`validationMessage`	`string | null`	Error message for failed validation
`options`	`array | null`	Options for `select` type fields
`minLength`	`number | null`	Minimum character length
`maxLength`	`number | null`	Maximum character length

Fees Object

Parameter	Type	Description
`platform`	`number`	Platform fee
`network`	`number`	Network/gas fee
`total`	`number`	Total fees

Dynamic Form Rendering

Use the `requiredFields` and `optionalFields` arrays to dynamically build your payout form. Each field includes validation constraints (`pattern` , `minLength` , `maxLength`) so you can validate client-side before submitting.

For `oneOfGroups` , at least one field from each group must be provided. For example, `["receiverBankCode", "receiverBankName"]` means you need either the SWIFT/BIC code or the bank name.

Error Responses

Code	HTTP	Message
`BAD_REQUEST`	400	Deposit not yet confirmed, or invalid intent
`NOT_FOUND`	404	Intent not found
`SELL_PAYOUT_PREVIEW_FAILED`	500	Failed to generate preview

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout-preview?
intentId=ri_1710345678000_a1b2c3d4e5f6g7h8&fiatCurrency=USD&country=US"
```

JAVASCRIPT

```javascript
const params = new URLSearchParams({
  intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
  fiatCurrency: 'USD',
  country: 'US',
});

const response = await fetch(
  `https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout-preview?${params}`
);

const { data } = await response.json();

console.log(`Provider: ${data.tspDisplayName}`);
console.log(`Payout: ${data.fiatAmount} ${data.fiatCurrency}`);
console.log(`Fees: ${data.fees.total}`);

// Dynamically render form fields
data.requiredFields.forEach((field) => {
  const input = document.createElement('input');
  input.name = field.fieldName;
  input.placeholder = field.placeholder;
  input.type = field.type;
  input.required = field.required;
  if (field.pattern) input.pattern = field.pattern;
  document.getElementById('payout-form').appendChild(input);
});
```

PYTHON

```python
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout-preview",
    params={
        "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
        "fiatCurrency": "USD",
        "country": "US",
    },
)

data = response.json()["data"]
print(f"Provider: {data['tspDisplayName']}")
print(f"Payout: {data['fiatAmount']} {data['fiatCurrency']}")
print(f"Fees: {data['fees']['total']}")

print("\nRequired fields:")
for field in data["requiredFields"]:
    print(f"  - {field['label']} ({field['fieldName']}): {field['type']}")
```

Next Steps

- Submit Sell Payout — Submit beneficiary details and trigger payout (Phase 2b)
- Create Sell Deposit — Generate deposit address (Phase 1)
- Off-Ramp Flow — Complete sell integration guide

REST API / RAMP / API INTEGRATION (ADVANCED) / OFF-RAMP (SELL)

Submit Sell Payout (Phase 2b)

Phase 2 of the two-phase off-ramp (sell) flow. Submit the customer's beneficiary details to trigger the fiat payout. The system obtains an FX quote (if cross-currency), maps the beneficiary fields to the payout provider, and initiates the bank transfer.

On success, the ramp intent automatically transitions to `completed`.

`POST` **`/payment/api/v1/off-ramp/sell/payout`**

Request

Headers

Header	Required	Description
`Content-Type`	Yes	`application/json`
`x-request-id`	No	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Body Parameters

Parameter	Type	Required	Description
`intentId`	`string`	Yes	Ramp intent ID
`cryptoDepositId`	`string`	Yes	Deposit ID from Phase 1 response
`fiatCurrency`	`string`	Yes	Target fiat currency (e.g. `USD`)
`payoutType`	`string`	No	Payout method (default: `bank_transfer`)
`country`	`string`	No	ISO country code (default: `US`)
`selectedTsp`	`string`	No	Payout provider from Payout Preview response (e.g. `zenpay_routing`). When omitted, the system selects the best provider automatically.
`beneficiaryDetails`	`object`	Yes	Dynamic fields from Payout Preview

beneficiaryDetails Object

The fields in this object are **dynamic** — they come from the `requiredFields` and `optionalFields` returned by the Payout Preview endpoint. Always call payout-preview first to discover which fields are needed for the target currency and country.

Common fields include:

Parameter	Type	Description
`receiverFirstName`	`string`	Beneficiary first name
`receiverLastName`	`string`	Beneficiary last name
`receiverAccountNumber`	`string`	Bank account number
`receiverBankName`	`string`	Bank name
`receiverBankCode`	`string`	SWIFT/BIC code
`receiverCountry`	`string`	ISO country code
`receiverAddressLine1`	`string`	Street address
`receiverCity`	`string`	City
`receiverState`	`string`	State/province
`receiverPinCode`	`string`	Postal/ZIP code
`receiverEmail`	`string`	Email address
`receiverPhone`	`string`	Phone number
`remittancePurpose`	`string`	Purpose of remittance (e.g. `PAYP001 - Family Support`)
`sourceOfFund`	`string`	Source of funds (e.g. `PAYF001 - Salary`)
`relationship`	`string`	Relationship to sender (e.g. `PAYR001 - Self`)

> **Field Requirements Vary**
> The required fields change based on the destination currency, country, and payout type. **Always use the payout-preview response** to determine which fields to collect. Do not hard-code field assumptions.

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "payoutId": "po_abc123def456",
    "status": "processing",
    "amount": 19.78,
    "currency": "USD",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8"
  },
  "message": "Sell payout submitted"
}
```

Response Fields

Parameter	Type	Description
`payoutId`	`string`	Payout tracking ID
`status`	`string`	Payout status (`processing`)
`amount`	`number`	Fiat payout amount
`currency`	`string`	Fiat currency
`intentId`	`string`	The ramp intent ID

Automatic Behaviors

Condition	Action
Received crypto matches expected (±5%)	Uses locked fiat amount from quote
Received crypto differs >5% from expected	Recalculates fiat amount proportionally with fee scaling
Same-currency payout (e.g. USD → USD)	Skips FX quotation step
Cross-currency payout	Obtains live FX quote from provider
Payout succeeds	Intent auto-transitions to `completed`
Payout fails	Intent receives `payout_failed` metadata

Error Responses

Code	HTTP	Message
`BAD_REQUEST`	400	Deposit not confirmed, invalid amount, or FX quotation failed
`NOT_FOUND`	404	Intent or deposit not found
`SELL_PAYOUT_FAILED`	500	Payout submission failed

Examples

CURL

```
curl -X POST https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout \
  -H "Content-Type: application/json" \
  -d '{
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "cryptoDepositId": "d4e5f6a7-b8c9-1234-5678-abcdef012345",
    "fiatCurrency": "USD",
    "country": "US",
    "selectedTsp": "zenpay_routing",
    "beneficiaryDetails": {
      "receiverFirstName": "John",
      "receiverLastName": "Doe",
      "receiverCountry": "US",
      "receiverAccountNumber": "123456789",
      "receiverBankName": "Bank of America",
      "receiverBankCode": "BOFAUS3N",
      "receiverAddressLine1": "123 Main St",
      "receiverCity": "New York",
      "receiverState": "NY",
      "receiverPinCode": "10001",
      "remittancePurpose": "PAYP001 - Family Support",
      "sourceOfFund": "PAYF001 - Salary",
      "relationship": "PAYR001 - Self"
    }
  }'
```

JAVASCRIPT

```javascript
const response = await fetch(
  'https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout',
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      intentId: 'ri_1710345678000_a1b2c3d4e5f6g7h8',
      cryptoDepositId: 'd4e5f6a7-b8c9-1234-5678-abcdef012345',
      fiatCurrency: 'USD',
      country: 'US',
      selectedTsp: 'zenpay_routing',
      beneficiaryDetails: {
        receiverFirstName: 'John',
        receiverLastName: 'Doe',
        receiverCountry: 'US',
        receiverAccountNumber: '123456789',
        receiverBankName: 'Bank of America',
        receiverBankCode: 'BOFAUS3N',
        receiverAddressLine1: '123 Main St',
        receiverCity: 'New York',
        receiverState: 'NY',
        receiverPinCode: '10001',
        remittancePurpose: 'PAYP001 - Family Support',
        sourceOfFund: 'PAYF001 - Salary',
        relationship: 'PAYR001 - Self',
      },
    }),
  }
);

const { data } = await response.json();

console.log(`Payout ID: ${data.payoutId}`);
console.log(`Status: ${data.status}`);
console.log(`Amount: ${data.amount} ${data.currency}`);

// The intent is now completed — redirect user
window.location.href = '/success';
```

PYTHON

```python
import requests

response = requests.post(
    "https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout",
    json={
        "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
        "cryptoDepositId": "d4e5f6a7-b8c9-1234-5678-abcdef012345",
        "fiatCurrency": "USD",
        "country": "US",
        "selectedTsp": "zenpay_routing",
        "beneficiaryDetails": {
            "receiverFirstName": "John",
            "receiverLastName": "Doe",
            "receiverCountry": "US",
            "receiverAccountNumber": "123456789",
            "receiverBankName": "Bank of America",
            "receiverBankCode": "BOFAUS3N",
            "receiverAddressLine1": "123 Main St",
            "receiverCity": "New York",
            "receiverState": "NY",
            "receiverPinCode": "10001",
            "remittancePurpose": "PAYP001 - Family Support",
            "sourceOfFund": "PAYF001 - Salary",
            "relationship": "PAYR001 - Self",
        },
    },
)

data = response.json()["data"]
print(f"Payout ID: {data['payoutId']}")
print(f"Status: {data['status']}")
print(f"Amount: {data['amount']} {data['currency']}")
```

Complete Two-Phase Flow

```
Phase 1: Deposit                    Phase 2: Payout
─────────────────                   ──────────────────────

1. POST sell/checkout               4. GET  payout-preview
   → Get deposit address               → Get required fields

2. Customer sends crypto            5. Render beneficiary form

3. Wait for confirmation            6. POST sell/payout
   (deposit → confirmed)               → Trigger fiat transfer
                                        → Intent → completed
```

Next Steps

- Get Payout Preview — Discover required fields (Phase 2a)
- Create Sell Deposit — Generate deposit address (Phase 1)
- Transaction Status — Poll transaction progress
- Off-Ramp Flow — Complete sell integration guide

Configuration

REST API / RAMP / CONFIGURATION

Get Ramp Config

Returns the current ramp configuration including integration mode and supported directions.

`GET` **`/payment/api/v1/on-ramp/config`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "integrationMode": "api",
    "supportsBuy": true,
    "supportsSell": true,
    "widgetBaseUrl": "https://buy.onramper.com"
  },
  "message": "Config retrieved"
}
```

Response Fields

Parameter	Type	Description
`integrationMode`	`string`	`widget`, `api`, or `both`
`supportsBuy`	`boolean`	Whether on-ramp (fiat to crypto) is enabled
`supportsSell`	`boolean`	Whether off-ramp (crypto to fiat) is enabled
`widgetBaseUrl`	`string`	Base URL for the embeddable widget

Error Responses

Code	HTTP	Message
`RAMP_CONFIG_FAILED`	500	Failed to retrieve configuration

Examples

CURL

```
curl https://api.zenpays.com/payment/api/v1/on-ramp/config
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/payment/api/v1/on-ramp/config');
const { data } = await response.json();

console.log(data.integrationMode); // "api"
console.log(data.supportsBuy);     // true
```

PYTHON

```
import requests

response = requests.get("https://api.zenpays.com/payment/api/v1/on-ramp/config")
data = response.json()["data"]

print(data["integrationMode"])  # "api"
print(data["supportsBuy"])      # True
```

Next Steps

- Supported Assets -- Get available currencies
- Buy Quotes -- Get buy quotes
- Widget URL -- Generate a widget URL

REST API / RAMP / CONFIGURATION

Get Supported Assets

Returns the list of supported crypto and fiat currencies for on-ramp or off-ramp.

`GET` **`/payment/api/v1/on-ramp/supported-assets`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Description
`type`	`string`	`buy` or `sell` (default: `buy`)
`country`	`string`	ISO country code (e.g. `US`, `GB`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "message": {
      "crypto": [
        {
          "id": "btc_bitcoin",
          "code": "BTC",
          "name": "Bitcoin",
          "network": "bitcoin"
        },
        {
          "id": "eth_ethereum",
          "code": "ETH",
          "name": "Ethereum",
          "network": "ethereum"
        }
      ],
      "fiat": [
        {
          "id": "usd",
          "code": "USD",
          "name": "US Dollar",
          "symbol": "$"
        },
        {
          "id": "eur",
          "code": "EUR",
          "name": "Euro",
          "symbol": "\u20ac"
        }
      ]
    }
  },
  "message": "Supported assets retrieved"
}
```


Error Responses

Code	HTTP	Message
`RAMP_SUPPORTED_FAILED`	500	Failed to get supported assets

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/on-ramp/supported-assets?type=buy&country=US"
```

JAVASCRIPT

```javascript
const response = await fetch(
  'https://api.zenpays.com/payment/api/v1/on-ramp/supported-assets?type=buy&country=US'
);
const { data } = await response.json();

console.log(data.message.crypto); // [{ id: "btc_bitcoin", ... }]
console.log(data.message.fiat);   // [{ id: "usd", ... }]
```

PYTHON

```python
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/on-ramp/supported-assets",
    params={"type": "buy", "country": "US"},
)
data = response.json()["data"]["message"]

print(data["crypto"])  # [{"id": "btc_bitcoin", ...}]
print(data["fiat"])    # [{"id": "usd", ...}]
```

Next Steps

- Payment Methods -- Get payment methods for a currency
- Buy Quotes -- Get quotes for a specific pair

REST API / RAMP / CONFIGURATION

Get Payment Methods

Returns payment methods available for a given source currency.

`GET` **`/payment/api/v1/on-ramp/payment-methods/:source`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

Public Endpoint

This endpoint does not require authentication headers.

Path Parameters

Parameter	Type	Required	Description
`source`	`string`	Yes	Source currency ID (e.g. `usd`, `eur`)

Query Parameters

Parameter	Type	Description
`type`	`string`	`buy` or `sell`
`destination`	`string`	Destination currency ID
`country`	`string`	ISO country code

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "id": "creditcard",
      "name": "Credit Card",
      "icon": "https://...",
      "currencyStatus": "available"
    },
    {
      "id": "debitcard",
      "name": "Debit Card"
    },
    {
      "id": "banktransfer",
      "name": "Bank Transfer"
    }
  ],
  "message": "Payment methods retrieved"
}
```


Error Responses

Code	HTTP	Message
`RAMP_PAYMENT_METHODS_FAILED`	500	Failed to get payment methods

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/on-ramp/payment-methods/usd?type=buy&country=US"
```

JAVASCRIPT

```
const response = await fetch(
  'https://api.zenpays.com/payment/api/v1/on-ramp/payment-methods/usd?type=buy&country=US'
);
const { data } = await response.json();

console.log(data); // [{ id: "creditcard", name: "Credit Card" }, ...]
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/on-ramp/payment-methods/usd",
    params={"type": "buy", "country": "US"},
)
methods = response.json()["data"]
```

Next Steps

- Buy Quotes -- Get quotes with a specific payment method
- Defaults -- Get recommended defaults per country

REST API / RAMP / CONFIGURATION

Get Defaults

Returns recommended default currencies, amounts, and payment methods per country.

`GET` **`/payment/api/v1/on-ramp/defaults`**

Request

Headers

Header	Description
`x-request-id`	Custom request ID for tracing

> **Public Endpoint**
> This endpoint does not require authentication headers.

Query Parameters

Parameter	Type	Description
`type`	`string`	`buy` or `sell` (default: `buy`)
`country`	`string`	ISO country code (e.g. `US`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "message": {
      "recommended": {
        "amount": 300,
        "paymentMethod": "debitcard",
        "source": "USD",
        "target": "BTC",
        "country": "us"
      },
      "defaults": {
        "us": {
          "amount": 300,
          "paymentMethod": "debitcard",
          "source": "USD",
          "target": "BTC"
        }
      }
    }
  },
  "message": "Defaults retrieved"
}
```

Error Responses

Code	HTTP	Message
`RAMP_DEFAULTS_FAILED`	500	Failed to get defaults

Examples

CURL

```
curl "https://api.zenpays.com/payment/api/v1/on-ramp/defaults?type=buy&country=US"
```

JAVASCRIPT

```
const response = await fetch(
  'https://api.zenpays.com/payment/api/v1/on-ramp/defaults?type=buy&country=US'
);
const { data } = await response.json();

const { amount, paymentMethod, source, target } = data.message.recommended;
console.log(`Default: ${amount} ${source} → ${target} via ${paymentMethod}`);
```

PYTHON

```
import requests

response = requests.get(
    "https://api.zenpays.com/payment/api/v1/on-ramp/defaults",
    params={"type": "buy", "country": "US"},
)
recommended = response.json()["data"]["message"]["recommended"]
print(f"Default: {recommended['amount']} {recommended['source']} → {recommended['target']}")
```

Next Steps

- Buy Quotes -- Get quotes using these defaults
- Supported Assets -- See all available currencies

REST API / RAMP / CONFIGURATION

Wallet Addresses

Crypto must be sent to (on-ramp) or withdrawn from (off-ramp) a wallet address. There are two ways to provide wallet addresses for ramp transactions.

Option 1: Pass Directly in the Request

Include `walletAddress` as a parameter in any checkout or quote request. This is the simplest approach.

```
{
  "walletAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD18"
}
```

Option 2: Use Merchant Default Wallets

Configure your wallet addresses in the merchant dashboard under **Settings > Financial Info > Crypto Wallets**. Then pass a `sessionId` in your request — ZenPays will automatically use your configured wallet for the target network.

Supported networks:

Network	Description
`ethereum`	Ethereum mainnet (ERC-20)
`polygon`	Polygon (MATIC)
`arbitrum`	Arbitrum One
`base`	Base
`optimism`	Optimism
`bnb`	BNB Chain
`avalanche`	Avalanche C-Chain

> **Tip**
> If you pass `walletAddress` directly, it always takes priority over merchant defaults.

Where It Applies

Endpoint	Wallet Behavior
`GET /on-ramp/quotes`	Optional -- improves quote accuracy when provided
`GET /on-ramp/rates`	Optional
`GET /on-ramp/prices`	Optional
`POST /on-ramp/checkout`	Required -- pass directly or use merchant defaults
`POST /on-ramp/widget-url`	Optional -- embedded in signed widget URL
`GET /off-ramp/quotes`	Optional
`GET /off-ramp/rates`	Optional
`POST /off-ramp/checkout`	Required -- pass directly or use merchant defaults
`POST /off-ramp/widget-url`	Optional

Examples

CURL

```
# Explicit wallet address
curl -X POST https://api.zenpays.com/payment/api/v1/on-ramp/checkout \
  -H "Content-Type: application/json" \
  -d '{
    "onramp": "moonpay",
    "source": "usd",
    "destination": "eth_ethereum",
    "amount": 100,
    "type": "buy",
    "paymentMethod": "creditcard",
    "walletAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD18"
  }'

# Using merchant default wallet
curl -X POST https://api.zenpays.com/payment/api/v1/on-ramp/checkout \
  -H "Content-Type: application/json" \
  -d '{
    "onramp": "moonpay",
    "source": "usd",
    "destination": "eth_ethereum",
    "amount": 100,
    "type": "buy",
    "paymentMethod": "creditcard",
    "sessionId": "cs_abc123",
    "network": "ethereum"
  }'
```

JAVASCRIPT

```
// Explicit wallet
const result = await fetch('https://api.zenpays.com/payment/api/v1/on-ramp/checkout', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    onramp: 'moonpay',
    source: 'usd',
    destination: 'eth_ethereum',
    amount: 100,
    type: 'buy',
    paymentMethod: 'creditcard',
    walletAddress: '0x742d35Cc6634C0532925a3b844Bc9e7595f2bD18',
  }),
});

// Using merchant default wallet
const result2 = await fetch('https://api.zenpays.com/payment/api/v1/on-ramp/checkout', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    onramp: 'moonpay',
    source: 'usd',
    destination: 'eth_ethereum',
    amount: 100,
    type: 'buy',
    paymentMethod: 'creditcard',
    sessionId: 'cs_abc123',
    network: 'ethereum',
  }),
});
```

PYTHON

```python
import requests

# Explicit wallet
response = requests.post(
    "https://api.zenpays.com/payment/api/v1/on-ramp/checkout",
    json={
        "onramp": "moonpay",
        "source": "usd",
        "destination": "eth_ethereum",
        "amount": 100,
        "type": "buy",
        "paymentMethod": "creditcard",
        "walletAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD18",
    },
)

# Using merchant default wallet
response2 = requests.post(
    "https://api.zenpays.com/payment/api/v1/on-ramp/checkout",
    json={
        "onramp": "moonpay",
        "source": "usd",
        "destination": "eth_ethereum",
        "amount": 100,
        "type": "buy",
        "paymentMethod": "creditcard",
        "sessionId": "cs_abc123",
        "network": "ethereum",
    },
)
```

Next Steps

- Buy Checkout -- Create a buy checkout intent
- Sell Checkout -- Create a sell checkout intent
- Widget URL -- Generate a signed widget URL

Vendors

REST API / VENDORS

Create Vendor

Create a new vendor record for managing referral partnerships and commission payouts.

`POST` **`/merchant/api/v1/vendors`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`name`	`string`	Yes	Vendor display name
`description`	`string`	No	Vendor description
`email`	`string`	Yes	Vendor email address (must be valid email)
`phone`	`string`	No	Vendor phone number
`website`	`string`	No	Vendor website URL
`status`	`string`	No	Initial status (`active` , `inactive` , `suspended` , `pending`)
`contactInfo`	`object`	Yes	Contact information object
`contactInfo.primaryContact`	`object`	Yes	Primary contact details
`contactInfo.primaryContact.firstName`	`string`	Yes	Contact first name
`contactInfo.primaryContact.lastName`	`string`	Yes	Contact last name
`contactInfo.primaryContact.email`	`string`	Yes	Contact email address
`contactInfo.primaryContact.phone`	`string`	Yes	Contact phone number

Parameter	Type	Required	Description
`contactInfo.primaryContact.designation`	`string`	Yes	Contact job title / designation
`contactInfo.address`	`object`	Yes	Business address
`contactInfo.address.street`	`string`	Yes	Street address
`contactInfo.address.city`	`string`	Yes	City
`contactInfo.address.state`	`string`	Yes	State or province
`contactInfo.address.postalCode`	`string`	Yes	Postal / ZIP code
`contactInfo.address.country`	`string`	Yes	Country
`businessInfo`	`object`	Yes	Business information
`businessInfo.registrationNumber`	`string`	Yes	Business registration number
`businessInfo.taxId`	`string`	Yes	Tax identification number
`businessInfo.businessType`	`string`	Yes	Type of business
`businessInfo.incorporationDate`	`string`	Yes	Date of incorporation (ISO 8601)
`businessInfo.monthlyVolume`	`number`	Yes	Expected monthly transaction volume
`commissionRate`	`number`	Yes	Commission rate (0-100)
`commissionType`	`string`	Yes	Commission model (`percentage` , `fixed` , `tiered`)
`commissionConfig`	`object`	No	Additional commission configuration (e.g., tier brackets)
`riskLevel`	`string`	No	Risk classification (`low` , `medium` , `high` , `critical`)
`payoutFrequency`	`string`	Yes	Payout schedule (`weekly` , `monthly` , `quarterly`)
`bankDetails`	`object`	Yes	Bank account for payouts
`bankDetails.accountNumber`	`string`	Yes	Bank account number
`bankDetails.routingNumber`	`string`	No	Routing number (US)
`bankDetails.ifscCode`	`string`	No	IFSC code (India)

Parameter	Type	Required	Description
`bankDetails.iban`	`string`	No	International Bank Account Number
`bankDetails.bankName`	`string`	Yes	Name of the bank
`bankDetails.swiftCode`	`string`	No	SWIFT / BIC code
`bankDetails.beneficiaryName`	`string`	Yes	Name on the bank account
`bankDetails.accountType`	`string`	Yes	Account type (`savings` , `current`)
`bankDetails.payoutType`	`string`	No	Preferred payout type
`environment`	`string`	No	Environment identifier
`metadata`	`object`	No	Custom key-value pairs

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "name": "Acme Referrals",
    "description": "Strategic referral partner for APAC region",
    "email": "partners@acmereferrals.com",
    "phone": "+14155551234",
    "website": "https://acmereferrals.com",
    "status": "active",
    "contactInfo": {
      "primaryContact": {
        "firstName": "Sarah",
        "lastName": "Chen",
        "email": "sarah.chen@acmereferrals.com",
        "phone": "+14155551235",
        "designation": "Partnerships Director"
      },
      "address": {
        "street": "100 Market Street",
        "city": "San Francisco",
        "state": "CA",
        "postalCode": "94105",
        "country": "US"
      }
    },
    "businessInfo": {
      "registrationNumber": "REG-2024-78901",
      "taxId": "87-6543210",
      "businessType": "LLC",
      "incorporationDate": "2020-03-15",
      "monthlyVolume": 500000
    },
    "commissionRate": 12.5,
    "commissionType": "percentage",
    "commissionConfig": null,
    "riskLevel": "low",
    "payoutFrequency": "monthly",
    "bankDetails": {
      "accountNumber": "****6789",
      "routingNumber": "****4321",
      "bankName": "First National Bank",
      "swiftCode": "FNBKUS33",
      "beneficiaryName": "Acme Referrals LLC",
      "accountType": "current",
      "payoutType": "wire"
    },
    "environment": "production",
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Vendor created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 145,
    "api_version": "v1",
```

```
    "endpoint": "POST /vendors",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`DUPLICATE_VENDOR`	409	Vendor with this email already exists

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/vendors \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Acme Referrals",
    "description": "Strategic referral partner for APAC region",
    "email": "partners@acmereferrals.com",
    "phone": "+14155551234",
    "website": "https://acmereferrals.com",
    "contactInfo": {
      "primaryContact": {
        "firstName": "Sarah",
        "lastName": "Chen",
        "email": "sarah.chen@acmereferrals.com",
        "phone": "+14155551235",
        "designation": "Partnerships Director"
      },
      "address": {
        "street": "100 Market Street",
        "city": "San Francisco",
        "state": "CA",
        "postalCode": "94105",
        "country": "US"
      }
    },
    "businessInfo": {
      "registrationNumber": "REG-2024-78901",
      "taxId": "87-6543210",
      "businessType": "LLC",
      "incorporationDate": "2020-03-15",
      "monthlyVolume": 500000
    },
    "commissionRate": 12.5,
    "commissionType": "percentage",
    "riskLevel": "low",
    "payoutFrequency": "monthly",
    "bankDetails": {
      "accountNumber": "9876543210",
      "routingNumber": "021000021",
      "bankName": "First National Bank",
      "swiftCode": "FNBKUS33",
      "beneficiaryName": "Acme Referrals LLC",
      "accountType": "current",
      "payoutType": "wire"
    }
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/vendors', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Acme Referrals',
    description: 'Strategic referral partner for APAC region',
    email: 'partners@acmereferrals.com',
    phone: '+14155551234',
    website: 'https://acmereferrals.com',
    contactInfo: {
      primaryContact: {
        firstName: 'Sarah',
        lastName: 'Chen',
        email: 'sarah.chen@acmereferrals.com',
        phone: '+14155551235',
        designation: 'Partnerships Director',
      },
      address: {
        street: '100 Market Street',
        city: 'San Francisco',
        state: 'CA',
        postalCode: '94105',
        country: 'US',
      },
    },
    businessInfo: {
      registrationNumber: 'REG-2024-78901',
      taxId: '87-6543210',
      businessType: 'LLC',
      incorporationDate: '2020-03-15',
      monthlyVolume: 500000,
    },
    commissionRate: 12.5,
    commissionType: 'percentage',
    riskLevel: 'low',
    payoutFrequency: 'monthly',
    bankDetails: {
      accountNumber: '9876543210',
      routingNumber: '021000021',
      bankName: 'First National Bank',
      swiftCode: 'FNBKUS33',
      beneficiaryName: 'Acme Referrals LLC',
      accountType: 'current',
      payoutType: 'wire',
    },
  }),
});

const result = await response.json();
console.log(result.data.vendorId); // vnd_a1b2c3d4e5f6g7h8
```

SDK

```
const vendor = await zenpays.vendors.create({
  name: 'Acme Referrals',
  description: 'Strategic referral partner for APAC region',
  email: 'partners@acmereferrals.com',
  phone: '+14155551234',
  website: 'https://acmereferrals.com',
  contactInfo: {
    primaryContact: {
      firstName: 'Sarah',
      lastName: 'Chen',
      email: 'sarah.chen@acmereferrals.com',
      phone: '+14155551235',
      designation: 'Partnerships Director',
    },
    address: {
      street: '100 Market Street',
      city: 'San Francisco',
      state: 'CA',
      postalCode: '94105',
      country: 'US',
    },
  },
  businessInfo: {
    registrationNumber: 'REG-2024-78901',
    taxId: '87-6543210',
    businessType: 'LLC',
    incorporationDate: '2020-03-15',
    monthlyVolume: 500000,
  },
  commissionRate: 12.5,
  commissionType: 'percentage',
  riskLevel: 'low',
  payoutFrequency: 'monthly',
  bankDetails: {
    accountNumber: '9876543210',
    routingNumber: '021000021',
    bankName: 'First National Bank',
    swiftCode: 'FNBKUS33',
    beneficiaryName: 'Acme Referrals LLC',
    accountType: 'current',
    payoutType: 'wire',
  },
});

console.log(vendor.vendorId); // vnd_a1b2c3d4e5f6g7h8
```

Related Endpoints

- List Vendors
- Get Vendor
- Update Vendor
- Delete Vendor

REST API / VENDORS

List Vendors

Retrieve a paginated list of vendors with support for filtering, searching, and sorting.

`GET` **`/merchant/api/v1/vendors`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Query Parameters

Parameter	Type	Description
`name`	`string`	Filter by vendor name (partial match)
`email`	`string`	Filter by vendor email
`vendorId`	`string`	Filter by vendor ID
`status`	`string`	Filter by status (`active` , `inactive` , `suspended` , `pending`)
`riskLevel`	`string`	Filter by risk level (`low` , `medium` , `high` , `critical`)
`environment`	`string`	Filter by environment
`commissionType`	`string`	Filter by commission type (`percentage` , `fixed` , `tiered`)
`minCommissionRate`	`number`	Minimum commission rate
`maxCommissionRate`	`number`	Maximum commission rate
`minTotalReferrals`	`number`	Minimum total referrals count
`minActiveReferrals`	`number`	Minimum active referrals count
`minCommissionEarned`	`number`	Minimum total commission earned
`payoutFrequency`	`string`	Filter by payout frequency (`weekly` , `monthly` , `quarterly`)
`kycStatus`	`string`	Filter by KYC verification status

Parameter	Type	Description
`createdAfter`	`string`	Filter vendors created after this date (ISO 8601)
`createdBefore`	`string`	Filter vendors created before this date (ISO 8601)
`lastPayoutAfter`	`string`	Filter by last payout date (after, ISO 8601)
`lastPayoutBefore`	`string`	Filter by last payout date (before, ISO 8601)
`vendorSortBy`	`string`	Field to sort by
`vendorSortOrder`	`string`	Sort direction (`asc` , `desc`)
`page`	`number`	Page number (default: `1`)
`limit`	`number`	Results per page (default: `20` , max: `100`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "vendorId": "vnd_a1b2c3d4e5f6g7h8",
      "name": "Acme Referrals",
      "email": "partners@acmereferrals.com",
      "phone": "+14155551234",
      "status": "active",
      "commissionRate": 12.5,
      "commissionType": "percentage",
      "riskLevel": "low",
      "payoutFrequency": "monthly",
      "totalReferrals": 45,
      "activeReferrals": 38,
      "totalCommissionEarned": 15230.50,
      "kycStatus": "verified",
      "createdAt": "2024-01-15T10:30:00.000Z",
      "updatedAt": "2024-03-20T14:15:00.000Z"
    },
    {
      "vendorId": "vnd_b2c3d4e5f6g7h8i9",
      "name": "Global Partners Inc",
      "email": "ops@globalpartners.io",
      "phone": "+442071234567",
      "status": "active",
      "commissionRate": 10.0,
      "commissionType": "tiered",
      "riskLevel": "medium",
      "payoutFrequency": "weekly",
      "totalReferrals": 120,
      "activeReferrals": 95,
      "totalCommissionEarned": 48750.00,
      "kycStatus": "verified",
      "createdAt": "2023-11-01T08:00:00.000Z",
      "updatedAt": "2024-03-18T09:45:00.000Z"
    }
  ],
  "message": "Vendors retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T10:30:00.000Z",
    "processing_time_ms": 85,
    "api_version": "v1",
    "endpoint": "GET /vendors",
    "user_type": "merchant",
    "pagination": {
      "page": 1,
      "limit": 20,
      "totalItems": 2,
      "totalPages": 1
    }
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/vendors?
status=active&riskLevel=low&page=1&limit=20" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  status: 'active',
  riskLevel: 'low',
  page: '1',
  limit: '20',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/vendors?${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data); // Array of vendors
console.log(result.meta.pagination.totalItems); // Total count
```

SDK

```
const vendors = await zenpays.vendors.list({
  status: 'active',
  riskLevel: 'low',
  page: 1,
  limit: 20,
});

console.log(vendors.data); // Array of vendors
console.log(vendors.meta.pagination.totalItems); // Total count
```

Related Endpoints

- Create Vendor

- Get Vendor
- Vendor Stats

REST API / VENDORS

Get Vendor

Retrieve detailed information about a specific vendor by their ID.

`GET` **`/merchant/api/v1/vendors/:id`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "name": "Acme Referrals",
    "description": "Strategic referral partner for APAC region",
    "email": "partners@acmereferrals.com",
    "phone": "+14155551234",
    "website": "https://acmereferrals.com",
    "status": "active",
    "contactInfo": {
      "primaryContact": {
        "firstName": "Sarah",
        "lastName": "Chen",
        "email": "sarah.chen@acmereferrals.com",
        "phone": "+14155551235",
        "designation": "Partnerships Director"
      },
      "address": {
        "street": "100 Market Street",
        "city": "San Francisco",
        "state": "CA",
        "postalCode": "94105",
        "country": "US"
      }
    },
    "businessInfo": {
      "registrationNumber": "REG-2024-78901",
      "taxId": "87-6543210",
      "businessType": "LLC",
      "incorporationDate": "2020-03-15",
      "monthlyVolume": 500000
    },
    "commissionRate": 12.5,
    "commissionType": "percentage",
    "commissionConfig": null,
    "riskLevel": "low",
    "payoutFrequency": "monthly",
    "bankDetails": {
      "accountNumber": "****6789",
      "routingNumber": "****4321",
      "bankName": "First National Bank",
      "swiftCode": "FNBKUS33",
      "beneficiaryName": "Acme Referrals LLC",
      "accountType": "current",
      "payoutType": "wire"
    },
    "totalReferrals": 45,
    "activeReferrals": 38,
    "totalCommissionEarned": 15230.50,
    "kycStatus": "verified",
    "environment": "production",
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-03-20T14:15:00.000Z"
  },
  "message": "Vendor retrieved successfully",
  "error": null,
  "meta": {
```

```
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T10:30:00.000Z",
    "processing_time_ms": 65,
    "api_version": "v1",
    "endpoint": "GET /vendors/:id",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.name); // Acme Referrals
console.log(result.data.commissionRate); // 12.5
```

SDK

```
const vendor = await zenpays.vendors.retrieve('vnd_a1b2c3d4e5f6g7h8');

console.log(vendor.name); // Acme Referrals
console.log(vendor.commissionRate); // 12.5
```

Related Endpoints

- List Vendors
- Update Vendor

- Vendor Commissions
- Vendor Analytics

REST API / VENDORS

Update Vendor

Update an existing vendor's information. All fields are optional; only provided fields will be updated.

`PUT` **`/merchant/api/v1/vendors/:id`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Description
`name`	`string`	Vendor display name
`description`	`string`	Vendor description
`email`	`string`	Vendor email address (must be valid email)
`phone`	`string`	Vendor phone number
`website`	`string`	Vendor website URL
`status`	`string`	Status (`active`, `inactive`, `suspended`, `pending`)
`contactInfo`	`object`	Contact information object
`contactInfo.primaryContact`	`object`	Primary contact details
`contactInfo.primaryContact.firstName`	`string`	Contact first name
`contactInfo.primaryContact.lastName`	`string`	Contact last name

Parameter	Type	Description
`contactInfo.primaryContact.email`	`string`	Contact email address
`contactInfo.primaryContact.phone`	`string`	Contact phone number
`contactInfo.primaryContact.designation`	`string`	Contact job title / designation
`contactInfo.address`	`object`	Business address
`contactInfo.address.street`	`string`	Street address
`contactInfo.address.city`	`string`	City
`contactInfo.address.state`	`string`	State or province
`contactInfo.address.postalCode`	`string`	Postal / ZIP code
`contactInfo.address.country`	`string`	Country
`businessInfo`	`object`	Business information
`businessInfo.registrationNumber`	`string`	Business registration number
`businessInfo.taxId`	`string`	Tax identification number
`businessInfo.businessType`	`string`	Type of business
`businessInfo.incorporationDate`	`string`	Date of incorporation (ISO 8601)
`businessInfo.monthlyVolume`	`number`	Expected monthly transaction volume
`commissionRate`	`number`	Commission rate (0-100)
`commissionType`	`string`	Commission model (`percentage` , `fixed` , `tiered`)
`commissionConfig`	`object`	Additional commission configuration
`riskLevel`	`string`	Risk classification (`low` , `medium` , `high` , `critical`)
`payoutFrequency`	`string`	Payout schedule (`weekly` , `monthly` , `quarterly`)
`bankDetails`	`object`	Bank account for payouts
`bankDetails.accountNumber`	`string`	Bank account number
`bankDetails.routingNumber`	`string`	Routing number (US)
`bankDetails.ifscCode`	`string`	IFSC code (India)
`bankDetails.iban`	`string`	International Bank Account Number
`bankDetails.bankName`	`string`	Name of the bank
`bankDetails.swiftCode`	`string`	SWIFT / BIC code
`bankDetails.beneficiaryName`	`string`	Name on the bank account
`bankDetails.accountType`	`string`	Account type (`savings` , `current`)
`bankDetails.payoutType`	`string`	Preferred payout type
`environment`	`string`	Environment identifier

Parameter	Type	Description
`metadata`	`object`	Custom key-value pairs

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "name": "Acme Referrals",
    "description": "Updated partner description",
    "email": "partners@acmereferrals.com",
    "phone": "+14155559999",
    "website": "https://acmereferrals.com",
    "status": "active",
    "contactInfo": {
      "primaryContact": {
        "firstName": "Sarah",
        "lastName": "Chen",
        "email": "sarah.chen@acmereferrals.com",
        "phone": "+14155551235",
        "designation": "VP of Partnerships"
      },
      "address": {
        "street": "100 Market Street",
        "city": "San Francisco",
        "state": "CA",
        "postalCode": "94105",
        "country": "US"
      }
    },
    "businessInfo": {
      "registrationNumber": "REG-2024-78901",
      "taxId": "87-6543210",
      "businessType": "LLC",
      "incorporationDate": "2020-03-15",
      "monthlyVolume": 750000
    },
    "commissionRate": 15.0,
    "commissionType": "percentage",
    "commissionConfig": null,
    "riskLevel": "low",
    "payoutFrequency": "monthly",
    "bankDetails": {
      "accountNumber": "****6789",
      "routingNumber": "****4321",
      "bankName": "First National Bank",
      "swiftCode": "FNBKUS33",
      "beneficiaryName": "Acme Referrals LLC",
      "accountType": "current",
      "payoutType": "wire"
    },
    "environment": "production",
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-03-21T11:00:00.000Z"
  },
  "message": "Vendor updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T11:00:00.000Z",
    "processing_time_ms": 110,
    "api_version": "v1",
```

```
    "endpoint": "PUT /vendors/:id",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist
`DUPLICATE_VENDOR`	409	Another vendor with this email already exists

Examples

CURL

```
curl -X PUT https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T11:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "commissionRate": 15.0,
    "phone": "+14155559999",
    "businessInfo": {
      "monthlyVolume": 750000
    },
    "contactInfo": {
      "primaryContact": {
        "designation": "VP of Partnerships"
      }
    }
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8', {
  method: 'PUT',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    commissionRate: 15.0,
    phone: '+14155559999',
    businessInfo: {
      monthlyVolume: 750000,
    },
    contactInfo: {
      primaryContact: {
        designation: 'VP of Partnerships',
      },
    },
  }),
});

const result = await response.json();
console.log(result.data.commissionRate); // 15.0
```

SDK

```
const vendor = await zenpays.vendors.update('vnd_a1b2c3d4e5f6g7h8', {
  commissionRate: 15.0,
  phone: '+14155559999',
  businessInfo: {
    monthlyVolume: 750000,
  },
  contactInfo: {
    primaryContact: {
      designation: 'VP of Partnerships',
    },
  },
});

console.log(vendor.commissionRate); // 15.0
```

Related Endpoints

- Get Vendor
- Update Vendor Status
- Update Vendor KYC
- Delete Vendor

REST API / VENDORS

Delete Vendor

Soft delete a vendor. The vendor record is marked as deleted but retained in the database for auditing purposes.

`DELETE` **`/merchant/api/v1/vendors/:id`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "deleted": true,
    "deletedAt": "2024-03-21T12:00:00.000Z"
  },
  "message": "Vendor deleted successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T12:00:00.000Z",
    "processing_time_ms": 95,
    "api_version": "v1",
    "endpoint": "DELETE /vendors/:id",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist
`VENDOR_HAS_ACTIVE_REFERRALS`	409	Vendor cannot be deleted while active referrals exist

Examples

CURL

```
curl -X DELETE https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T12:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8', {
  method: 'DELETE',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.deleted); // true
```

SDK

```
const result = await zenpays.vendors.delete('vnd_a1b2c3d4e5f6g7h8');

console.log(result.deleted); // true
```

Related Endpoints

- Get Vendor
- Update Vendor Status
- List Vendors

REST API / VENDORS

Vendor Stats

Retrieve aggregate statistics across all vendors, including counts by status, total commissions, and referral summaries.

`GET` **`/merchant/api/v1/vendors/stats`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "totalVendors": 48,
    "activeVendors": 35,
    "inactiveVendors": 5,
    "suspendedVendors": 3,
    "pendingVendors": 5,
    "totalReferrals": 1240,
    "activeReferrals": 980,
    "totalCommissionsPaid": 245830.75,
    "totalCommissionsPending": 18420.50,
    "averageCommissionRate": 11.3,
    "topPerformingVendors": [
      {
        "vendorId": "vnd_b2c3d4e5f6g7h8i9",
        "name": "Global Partners Inc",
        "totalReferrals": 120,
        "totalCommissionEarned": 48750.00
      },
      {
        "vendorId": "vnd_a1b2c3d4e5f6g7h8",
        "name": "Acme Referrals",
        "totalReferrals": 45,
        "totalCommissionEarned": 15230.50
      }
    ],
    "riskDistribution": {
      "low": 30,
      "medium": 12,
      "high": 4,
      "critical": 2
    },
    "kycDistribution": {
      "verified": 38,
      "pending": 7,
      "rejected": 3
    }
  },
  "message": "Vendor stats retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T10:30:00.000Z",
    "processing_time_ms": 210,
    "api_version": "v1",
    "endpoint": "GET /vendors/stats",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/vendors/stats \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/vendors/stats', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.totalVendors); // 48
console.log(result.data.totalCommissionsPaid); // 245830.75
```

SDK

```
const stats = await zenpays.vendors.stats();

console.log(stats.totalVendors); // 48
console.log(stats.totalCommissionsPaid); // 245830.75
console.log(stats.riskDistribution); // { low: 30, medium: 12, high: 4, critical: 2 }
```

Related Endpoints

- List Vendors
- Vendor Analytics
- Vendor Commissions

REST API / VENDORS

Update Vendor Status

Update the status of a specific vendor. Use this endpoint to activate, deactivate, or suspend a vendor.

`PATCH` **`/merchant/api/v1/vendors/:id/status`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Required	Description
`status`	`string`	Yes	New status (`active`, `inactive`, `suspended`, `pending`)
`reason`	`string`	No	Reason for the status change
`adminComment`	`string`	No	Internal admin comment

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "previousStatus": "active",
    "status": "suspended",
    "reason": "Compliance review required",
    "adminComment": "Flagged during quarterly audit",
    "updatedAt": "2024-03-21T14:00:00.000Z"
  },
  "message": "Vendor status updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T14:00:00.000Z",
    "processing_time_ms": 90,
    "api_version": "v1",
    "endpoint": "PATCH /vendors/:id/status",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid status value
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist
`INVALID_STATUS_TRANSITION`	409	Status transition is not allowed

Examples

CURL

```
curl -X PATCH https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/status \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T14:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "status": "suspended",
    "reason": "Compliance review required",
    "adminComment": "Flagged during quarterly audit"
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/status', {
  method: 'PATCH',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    status: 'suspended',
    reason: 'Compliance review required',
    adminComment: 'Flagged during quarterly audit',
  }),
});

const result = await response.json();
console.log(result.data.previousStatus); // active
console.log(result.data.status); // suspended
```

SDK

```
const result = await zenpays.vendors.updateStatus('vnd_a1b2c3d4e5f6g7h8', {
  status: 'suspended',
  reason: 'Compliance review required',
  adminComment: 'Flagged during quarterly audit',
});

console.log(result.previousStatus); // active
console.log(result.status); // suspended
```

Related Endpoints

- Get Vendor
- Update Vendor
- Update Vendor KYC

REST API / VENDORS

Update Vendor KYC

Update the KYC (Know Your Customer) verification status of a vendor.

`PATCH` **`/merchant/api/v1/vendors/:id/kyc`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Required	Description
`verificationStatus`	`string`	Yes	KYC status (`pending`, `verified`, `rejected`)
`verifiedBy`	`string`	No	Identifier of the person who performed verification
`rejectionReason`	`string`	No	Reason for rejection (required when status is `rejected`)
`adminNotes`	`string`	No	Internal notes about the KYC review

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "previousKycStatus": "pending",
    "kycStatus": "verified",
    "verifiedBy": "admin_c3d4e5f6g7h8i9j0",
    "rejectionReason": null,
    "adminNotes": "All documents verified successfully",
    "verifiedAt": "2024-03-21T15:00:00.000Z",
    "updatedAt": "2024-03-21T15:00:00.000Z"
  },
  "message": "Vendor KYC status updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T15:00:00.000Z",
    "processing_time_ms": 105,
    "api_version": "v1",
    "endpoint": "PATCH /vendors/:id/kyc",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid verification status or missing rejection reason
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist

Examples

CURL

```
curl -X PATCH https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/kyc \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T15:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "verificationStatus": "verified",
    "verifiedBy": "admin_c3d4e5f6g7h8i9j0",
    "adminNotes": "All documents verified successfully"
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/kyc', {
  method: 'PATCH',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    verificationStatus: 'verified',
    verifiedBy: 'admin_c3d4e5f6g7h8i9j0',
    adminNotes: 'All documents verified successfully',
  }),
});

const result = await response.json();
console.log(result.data.kycStatus); // verified
```

SDK

```
const result = await zenpays.vendors.updateKyc('vnd_a1b2c3d4e5f6g7h8', {
  verificationStatus: 'verified',
  verifiedBy: 'admin_c3d4e5f6g7h8i9j0',
  adminNotes: 'All documents verified successfully',
});

console.log(result.kycStatus); // verified
```

Related Endpoints

- Get Vendor
- Update Vendor Status
- Update Vendor

REST API / VENDORS

Vendor Commissions

Retrieve a paginated list of commission records for a specific vendor, with filtering and sorting support.

`GET` **`/merchant/api/v1/vendors/:id/commissions`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Query Parameters

Parameter	Type	Description
`status`	`string`	Filter by commission status (`pending`, `calculated`, `paid`, `cancelled`, `disputed`)
`commissionType`	`string`	Filter by commission type (`percentage`, `fixed`, `tiered`)
`calculatedAfter`	`string`	Commissions calculated after this date (ISO 8601)
`calculatedBefore`	`string`	Commissions calculated before this date (ISO 8601)
`paidAfter`	`string`	Commissions paid after this date (ISO 8601)
`paidBefore`	`string`	Commissions paid before this date (ISO 8601)
`minCommissionAmount`	`number`	Minimum commission amount
`currency`	`string`	Filter by currency code (e.g., `USD`, `EUR`)

Parameter	Type	Description
`environment`	`string`	Filter by environment
`page`	`number`	Page number (default: `1`)
`limit`	`number`	Results per page (default: `20`)
`sortBy`	`string`	Field to sort by
`sortOrder`	`string`	Sort direction (`asc`, `desc`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "commissionId": "comm_d4e5f6g7h8i9j0k1",
      "vendorId": "vnd_a1b2c3d4e5f6g7h8",
      "referralId": "ref_e5f6g7h8i9j0k1l2",
      "transactionId": "txn_f6g7h8i9j0k1l2m3",
      "amount": 125.00,
      "currency": "USD",
      "commissionRate": 12.5,
      "commissionType": "percentage",
      "baseAmount": 1000.00,
      "status": "paid",
      "calculatedAt": "2024-03-15T10:00:00.000Z",
      "paidAt": "2024-03-20T14:00:00.000Z",
      "environment": "production"
    },
    {
      "commissionId": "comm_g7h8i9j0k1l2m3n4",
      "vendorId": "vnd_a1b2c3d4e5f6g7h8",
      "referralId": "ref_h8i9j0k1l2m3n4o5",
      "transactionId": "txn_i9j0k1l2m3n4o5p6",
      "amount": 75.50,
      "currency": "USD",
      "commissionRate": 12.5,
      "commissionType": "percentage",
      "baseAmount": 604.00,
      "status": "pending",
      "calculatedAt": "2024-03-21T08:00:00.000Z",
      "paidAt": null,
      "environment": "production"
    }
  ],
  "message": "Vendor commissions retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T10:30:00.000Z",
    "processing_time_ms": 95,
    "api_version": "v1",
    "endpoint": "GET /vendors/:id/commissions",
    "user_type": "merchant",
    "pagination": {
      "page": 1,
      "limit": 20,
      "totalItems": 2,
      "totalPages": 1
    }
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/commissions?
status=pending&currency=USD&page=1&limit=20" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  status: 'pending',
  currency: 'USD',
  page: '1',
  limit: '20',
});

const response = await
fetch(`https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/commissions?
${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data); // Array of commission records
console.log(result.meta.pagination.totalItems); // Total count
```

SDK

```
const commissions = await zenpays.vendors.listCommissions('vnd_a1b2c3d4e5f6g7h8', {
  status: 'pending',
  currency: 'USD',
  page: 1,
  limit: 20,
});

console.log(commissions.data); // Array of commission records
console.log(commissions.meta.pagination.totalItems); // Total count
```

Related Endpoints

- Get Vendor
- Vendor Referrals
- Vendor Payout
- Vendor Analytics

REST API / VENDORS

Vendor Referrals

Retrieve the list of referrals associated with a specific vendor.

`GET` **`/merchant/api/v1/vendors/:id/referrals`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "referralId": "ref_e5f6g7h8i9j0k1l2",
      "vendorId": "vnd_a1b2c3d4e5f6g7h8",
      "merchantId": "mer_k1l2m3n4o5p6q7r8",
      "referralCode": "ACME-2024-001",
      "referralSource": "partner_portal",
      "status": "active",
      "commissionRate": 12.5,
      "totalTransactions": 85,
      "totalRevenue": 42500.00,
      "totalCommission": 5312.50,
      "environment": "production",
      "notes": "Referred via partner event",
      "createdAt": "2024-01-20T09:00:00.000Z",
      "updatedAt": "2024-03-21T10:00:00.000Z"
    },
    {
      "referralId": "ref_h8i9j0k1l2m3n4o5",
      "vendorId": "vnd_a1b2c3d4e5f6g7h8",
      "merchantId": "mer_l2m3n4o5p6q7r8s9",
      "referralCode": "ACME-2024-002",
      "referralSource": "website",
      "status": "active",
      "commissionRate": 12.5,
      "totalTransactions": 32,
      "totalRevenue": 18200.00,
      "totalCommission": 2275.00,
      "environment": "production",
      "notes": null,
      "createdAt": "2024-02-10T14:30:00.000Z",
      "updatedAt": "2024-03-19T16:45:00.000Z"
    }
  ],
  "message": "Vendor referrals retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T10:30:00.000Z",
    "processing_time_ms": 78,
    "api_version": "v1",
    "endpoint": "GET /vendors/:id/referrals",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/referrals \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/referrals', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data); // Array of referrals
console.log(result.data[0].referralCode); // ACME-2024-001
```

SDK

```
const referrals = await zenpays.vendors.listReferrals('vnd_a1b2c3d4e5f6g7h8');

console.log(referrals.data); // Array of referrals
console.log(referrals.data[0].referralCode); // ACME-2024-001
```

Related Endpoints

- Create Vendor Referral
- Vendor Commissions
- Get Vendor

REST API / VENDORS

Create Vendor Referral

Create a new referral record linking a merchant to a vendor for commission tracking.

`POST` **`/merchant/api/v1/vendors/:id/referrals`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Required	Description
`merchantId`	`string`	Yes	ID of the referred merchant
`referralCode`	`string`	No	Custom referral tracking code
`referralSource`	`string`	No	Source of the referral (e.g., `website`, `partner_portal`, `event`)
`commissionRate`	`number`	No	Override commission rate for this referral (0-100)
`environment`	`string`	No	Environment identifier
`notes`	`string`	No	Additional notes about the referral

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "referralId": "ref_j0k1l2m3n4o5p6q7",
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "merchantId": "mer_m3n4o5p6q7r8s9t0",
    "referralCode": "ACME-2024-003",
    "referralSource": "partner_portal",
    "status": "active",
    "commissionRate": 15.0,
    "totalTransactions": 0,
    "totalRevenue": 0,
    "totalCommission": 0,
    "environment": "production",
    "notes": "Enterprise client referred at FinTech Summit 2024",
    "createdAt": "2024-03-21T11:00:00.000Z",
    "updatedAt": "2024-03-21T11:00:00.000Z"
  },
  "message": "Vendor referral created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T11:00:00.000Z",
    "processing_time_ms": 130,
    "api_version": "v1",
    "endpoint": "POST /vendors/:id/referrals",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist
`MERCHANT_NOT_FOUND`	404	Merchant with specified ID does not exist
`DUPLICATE_REFERRAL`	409	Referral for this merchant already exists under this vendor

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/referrals \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T11:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "merchantId": "mer_m3n4o5p6q7r8s9t0",
    "referralCode": "ACME-2024-003",
    "referralSource": "partner_portal",
    "commissionRate": 15.0,
    "environment": "production",
    "notes": "Enterprise client referred at FinTech Summit 2024"
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/referrals', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    merchantId: 'mer_m3n4o5p6q7r8s9t0',
    referralCode: 'ACME-2024-003',
    referralSource: 'partner_portal',
    commissionRate: 15.0,
    environment: 'production',
    notes: 'Enterprise client referred at FinTech Summit 2024',
  }),
});

const result = await response.json();
console.log(result.data.referralId); // ref_j0k1l2m3n4o5p6q7
```

SDK

```
const referral = await zenpays.vendors.createReferral('vnd_a1b2c3d4e5f6g7h8', {
  merchantId: 'mer_m3n4o5p6q7r8s9t0',
  referralCode: 'ACME-2024-003',
  referralSource: 'partner_portal',
  commissionRate: 15.0,
  environment: 'production',
  notes: 'Enterprise client referred at FinTech Summit 2024',
});

console.log(referral.referralId); // ref_j0k1l2m3n4o5p6q7
```

Related Endpoints

- Vendor Referrals

- Vendor Commissions
- Get Vendor

REST API / VENDORS

Vendor Payout

Initiate a commission payout to a vendor. This triggers a transfer of earned commissions to the vendor's registered bank account.

`POST` **`/merchant/api/v1/vendors/:id/payouts`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Required	Description
`amount`	`number`	No	Payout amount (min: `0`). If omitted, pays all pending commissions
`currency`	`string`	Yes	Currency code (e.g., `USD`, `EUR`, `INR`)
`paymentMethod`	`string`	No	Payout method override
`adminNotes`	`string`	No	Internal notes about this payout
`environment`	`string`	Yes	Environment identifier

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "payoutId": "pyt_n4o5p6q7r8s9t0u1",
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "amount": 5312.50,
    "currency": "USD",
    "status": "processing",
    "paymentMethod": "wire",
    "commissionsIncluded": 12,
    "bankDetails": {
      "bankName": "First National Bank",
      "beneficiaryName": "Acme Referrals LLC",
      "accountNumber": "****6789"
    },
    "adminNotes": "Q1 2024 commission payout",
    "environment": "production",
    "initiatedAt": "2024-03-21T16:00:00.000Z",
    "estimatedArrival": "2024-03-24T16:00:00.000Z"
  },
  "message": "Vendor payout initiated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T16:00:00.000Z",
    "processing_time_ms": 250,
    "api_version": "v1",
    "endpoint": "POST /vendors/:id/payouts",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`INSUFFICIENT_BALANCE`	400	Payout amount exceeds pending commissions
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist
`PAYOUT_IN_PROGRESS`	409	A payout is already being processed for this vendor
`KYC_NOT_VERIFIED`	422	Vendor KYC must be verified before payouts

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/payouts \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T16:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "amount": 5312.50,
    "currency": "USD",
    "paymentMethod": "wire",
    "adminNotes": "Q1 2024 commission payout",
    "environment": "production"
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/payouts', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    amount: 5312.50,
    currency: 'USD',
    paymentMethod: 'wire',
    adminNotes: 'Q1 2024 commission payout',
    environment: 'production',
  }),
});

const result = await response.json();
console.log(result.data.payoutId); // pyt_n4o5p6q7r8s9t0u1
console.log(result.data.status); // processing
```

SDK

```
const payout = await zenpays.vendors.createPayout('vnd_a1b2c3d4e5f6g7h8', {
  amount: 5312.50,
  currency: 'USD',
  paymentMethod: 'wire',
  adminNotes: 'Q1 2024 commission payout',
  environment: 'production',
});

console.log(payout.payoutId); // pyt_n4o5p6q7r8s9t0u1
console.log(payout.status); // processing
```

Related Endpoints

- Vendor Commissions
- Get Vendor

- Vendor Analytics

REST API / VENDORS

Vendor Analytics

Retrieve analytics and performance metrics for a specific vendor over a given time period.

`GET` **`/merchant/api/v1/vendors/:id/analytics`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`string`	Yes	Vendor ID (e.g., `vnd_a1b2c3d4e5f6g7h8`)

Query Parameters

Parameter	Type	Description
`startDate`	`string`	Start date for analytics period (ISO 8601)
`endDate`	`string`	End date for analytics period (ISO 8601)
`period`	`string`	Aggregation period (`daily`, `weekly`, `monthly`, `quarterly`, `yearly`)
`environment`	`string`	Filter by environment
`includeInactiveReferrals`	`boolean`	Include inactive referrals in calculations (default: `false`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "vendorId": "vnd_a1b2c3d4e5f6g7h8",
    "period": {
      "startDate": "2024-01-01T00:00:00.000Z",
      "endDate": "2024-03-31T23:59:59.999Z",
      "granularity": "monthly"
    },
    "summary": {
      "totalReferrals": 45,
      "activeReferrals": 38,
      "newReferrals": 12,
      "totalTransactions": 1240,
      "totalRevenue": 620000.00,
      "totalCommission": 77500.00,
      "averageTransactionValue": 500.00,
      "conversionRate": 68.5
    },
    "timeline": [
      {
        "period": "2024-01",
        "referrals": 4,
        "transactions": 380,
        "revenue": 190000.00,
        "commission": 23750.00
      },
      {
        "period": "2024-02",
        "referrals": 3,
        "transactions": 420,
        "revenue": 210000.00,
        "commission": 26250.00
      },
      {
        "period": "2024-03",
        "referrals": 5,
        "transactions": 440,
        "revenue": 220000.00,
        "commission": 27500.00
      }
    ],
    "topReferrals": [
      {
        "referralId": "ref_e5f6g7h8i9j0k1l2",
        "merchantId": "mer_k1l2m3n4o5p6q7r8",
        "referralCode": "ACME-2024-001",
        "transactions": 85,
        "revenue": 42500.00,
        "commission": 5312.50
      },
      {
        "referralId": "ref_h8i9j0k1l2m3n4o5",
        "merchantId": "mer_l2m3n4o5p6q7r8s9",
        "referralCode": "ACME-2024-002",
        "transactions": 32,
        "revenue": 18200.00,
        "commission": 2275.00
      }
```

```
    ],
    "payoutSummary": {
      "totalPaid": 65000.00,
      "totalPending": 12500.00,
      "lastPayoutDate": "2024-03-15T14:00:00.000Z",
      "nextScheduledPayout": "2024-04-01T00:00:00.000Z"
    }
  },
  "message": "Vendor analytics retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-03-21T10:30:00.000Z",
    "processing_time_ms": 320,
    "api_version": "v1",
    "endpoint": "GET /vendors/:id/analytics",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters or date range
`UNAUTHORIZED`	401	Invalid API key
`VENDOR_NOT_FOUND`	404	Vendor with specified ID does not exist

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/analytics?
startDate=2024-01-01T00:00:00.000Z&endDate=2024-03-31T23:59:59.999Z&period=monthly" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-03-21T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-03-31T23:59:59.999Z',
  period: 'monthly',
});

const response = await
fetch(`https://api.zenpays.com/merchant/api/v1/vendors/vnd_a1b2c3d4e5f6g7h8/analytics?${params}`,
{
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.summary.totalRevenue); // 620000.00
console.log(result.data.timeline); // Array of period data
```

SDK

```
const analytics = await zenpays.vendors.analytics('vnd_a1b2c3d4e5f6g7h8', {
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-03-31T23:59:59.999Z',
  period: 'monthly',
});

console.log(analytics.summary.totalRevenue); // 620000.00
console.log(analytics.timeline); // Array of period data
console.log(analytics.payoutSummary.totalPending); // 12500.00
```

Related Endpoints

- Get Vendor
- Vendor Commissions
- Vendor Stats
- Vendor Payout

Chargebacks

REST API / CHARGEBACKS

Create Chargeback

Create a new chargeback record against a specific transaction.

`POST` **`/merchant/api/v1/chargebacks`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Body Parameters

Parameter	Type	Required	Description
`transactionId`	`string`	Yes	ID of the original transaction being disputed
`amount`	`number`	Yes	Chargeback amount
`currency`	`string`	Yes	ISO 4217 currency code
`reason`	`string`	Yes	Reason for the chargeback
`source`	`string`	Yes	Source of the chargeback (e.g. card_network, customer, bank)
`reasonCode`	`string`	No	Standard reason code from the card network
`reasonDescription`	`string`	No	Detailed description of the reason
`externalChargebackId`	`string`	No	External chargeback identifier from the payment network
`disputeDeadline`	`string`	No	Deadline to respond to the dispute (ISO 8601)
`evidenceDeadline`	`string`	No	Deadline to submit evidence (ISO 8601)

Parameter	Type	Required	Description
`priority`	`string`	No	Priority level: `low`, `medium`, `high`, `urgent`

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "chargebackId": "cb_a1b2c3d4e5f6g7h8",
    "transactionId": "txn_f8e7d6c5b4a39281",
    "amount": 150.00,
    "currency": "USD",
    "reason": "fraudulent",
    "source": "card_network",
    "reasonCode": "10.4",
    "reasonDescription": "Other Fraud - Card-Absent Environment",
    "externalChargebackId": "ext_cb_12345",
    "status": "initiated",
    "outcome": "pending",
    "priority": "high",
    "disputeDeadline": "2024-02-15T23:59:59.000Z",
    "evidenceDeadline": "2024-02-10T23:59:59.000Z",
    "evidence": [],
    "merchantResponse": null,
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Chargeback created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 45,
    "api_version": "v1",
    "endpoint": "POST /chargebacks",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`TRANSACTION_NOT_FOUND`	404	Referenced transaction does not exist
`DUPLICATE_CHARGEBACK`	409	A chargeback already exists for this transaction

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/chargebacks \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx" \
  -d '{
    "transactionId": "txn_f8e7d6c5b4a39281",
    "amount": 150.00,
    "currency": "USD",
    "reason": "fraudulent",
    "source": "card_network",
    "reasonCode": "10.4",
    "priority": "high",
    "disputeDeadline": "2024-02-15T23:59:59.000Z",
    "evidenceDeadline": "2024-02-10T23:59:59.000Z"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/chargebacks', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
  body: JSON.stringify({
    transactionId: 'txn_f8e7d6c5b4a39281',
    amount: 150.00,
    currency: 'USD',
    reason: 'fraudulent',
    source: 'card_network',
    reasonCode: '10.4',
    priority: 'high',
    disputeDeadline: '2024-02-15T23:59:59.000Z',
    evidenceDeadline: '2024-02-10T23:59:59.000Z',
  }),
});

const result = await response.json();
console.log(result.data.chargebackId); // cb_a1b2c3d4e5f6g7h8
```

SDK

```
const chargeback = await zenpays.chargebacks.create({
  transactionId: 'txn_f8e7d6c5b4a39281',
  amount: 150.00,
  currency: 'USD',
  reason: 'fraudulent',
  source: 'card_network',
  reasonCode: '10.4',
  priority: 'high',
  disputeDeadline: '2024-02-15T23:59:59.000Z',
  evidenceDeadline: '2024-02-10T23:59:59.000Z',
});

console.log(chargeback.chargebackId); // cb_a1b2c3d4e5f6g7h8
```

Related Endpoints

- List Chargebacks
- Get Chargeback
- Submit Evidence
- Add Response

REST API / CHARGEBACKS

List Chargebacks

Retrieve a paginated list of chargebacks with optional filters for status, priority, date range, and more.

`GET` **`/merchant/api/v1/chargebacks`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Query Parameters

Parameter	Type	Description
`customerId`	`string`	Filter by customer ID
`transactionId`	`string`	Filter by transaction ID
`status`	`string`	Filter by status: `initiated`, `pending_review`, `evidence_requested`, `evidence_submitted`, `under_review`, `resolved_won`, `resolved_lost`, `cancelled`, `expired`
`reason`	`string`	Filter by chargeback reason
`outcome`	`string`	Filter by outcome: `pending`, `won`, `lost`, `cancelled`, `expired`
`priority`	`string`	Filter by priority: `low`, `medium`, `high`, `urgent`
`currency`	`string`	Filter by ISO 4217 currency code
`startDate`	`string`	Start date filter (ISO 8601)
`endDate`	`string`	End date filter (ISO 8601)
`minAmount`	`number`	Minimum chargeback amount
`maxAmount`	`number`	Maximum chargeback amount

Parameter	Type	Description
`urgentDeadlinesOnly`	`boolean`	Return only chargebacks with upcoming deadlines
`search`	`string`	Free-text search across chargeback fields
`page`	`number`	Page number (default: `1`)
`limit`	`number`	Items per page (default: `20`)
`sortBy`	`string`	Field to sort by (e.g. `createdAt`, `amount`, `priority`)
`sortOrder`	`string`	Sort direction: `asc`, `desc`

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "chargebacks": [
      {
        "chargebackId": "cb_a1b2c3d4e5f6g7h8",
        "transactionId": "txn_f8e7d6c5b4a39281",
        "amount": 150.00,
        "currency": "USD",
        "reason": "fraudulent",
        "source": "card_network",
        "status": "evidence_requested",
        "outcome": "pending",
        "priority": "high",
        "disputeDeadline": "2024-02-15T23:59:59.000Z",
        "evidenceDeadline": "2024-02-10T23:59:59.000Z",
        "createdAt": "2024-01-15T10:30:00.000Z",
        "updatedAt": "2024-01-18T14:22:00.000Z"
      },
      {
        "chargebackId": "cb_b2c3d4e5f6g7h8i9",
        "transactionId": "txn_e7d6c5b4a3928100",
        "amount": 75.50,
        "currency": "USD",
        "reason": "product_not_received",
        "source": "customer",
        "status": "resolved_won",
        "outcome": "won",
        "priority": "medium",
        "disputeDeadline": "2024-02-20T23:59:59.000Z",
        "evidenceDeadline": "2024-02-15T23:59:59.000Z",
        "createdAt": "2024-01-10T08:15:00.000Z",
        "updatedAt": "2024-01-25T16:45:00.000Z"
      }
    ],
    "pagination": {
      "page": 1,
      "limit": 20,
      "totalItems": 42,
      "totalPages": 3
    }
  },
  "message": "Chargebacks retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 38,
    "api_version": "v1",
    "endpoint": "GET /chargebacks",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/chargebacks?
status=evidence_requested&priority=high&page=1&limit=20" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  status: 'evidence_requested',
  priority: 'high',
  page: '1',
  limit: '20',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/chargebacks?${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
});

const result = await response.json();
console.log(result.data.chargebacks); // [{ chargebackId: "cb_a1b2c3d4e5f6g7h8", ... }]
console.log(result.data.pagination.totalItems); // 42
```

SDK

```
const result = await zenpays.chargebacks.list({
  status: 'evidence_requested',
  priority: 'high',
  page: 1,
  limit: 20,
});

console.log(result.chargebacks); // [{ chargebackId: "cb_a1b2c3d4e5f6g7h8", ... }]
console.log(result.pagination.totalItems); // 42
```

Related Endpoints

- Create Chargeback
- Get Chargeback
- Chargeback Stats
- Chargeback Analytics

REST API / CHARGEBACKS

Get Chargeback

Retrieve full details of a specific chargeback including evidence and merchant response.

`GET` **`/merchant/api/v1/chargebacks/:chargebackId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Path Parameters

Parameter	Type	Required	Description
`chargebackId`	`string`	Yes	Unique chargeback identifier

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "chargebackId": "cb_a1b2c3d4e5f6g7h8",
    "transactionId": "txn_f8e7d6c5b4a39281",
    "amount": 150.00,
    "currency": "USD",
    "reason": "fraudulent",
    "source": "card_network",
    "reasonCode": "10.4",
    "reasonDescription": "Other Fraud - Card-Absent Environment",
    "externalChargebackId": "ext_cb_12345",
    "status": "evidence_requested",
    "outcome": "pending",
    "priority": "high",
    "disputeDeadline": "2024-02-15T23:59:59.000Z",
    "evidenceDeadline": "2024-02-10T23:59:59.000Z",
    "evidence": [
      {
        "type": "receipt",
        "documentUrl": "https://files.zenpays.com/evidence/receipt_001.pdf",
        "fileName": "receipt_001.pdf",
        "mimeType": "application/pdf",
        "fileSize": 204800,
        "description": "Original transaction receipt",
        "submittedAt": "2024-01-18T14:22:00.000Z"
      }
    ],
    "merchantResponse": "Customer received the product on Jan 12. Delivery confirmation attached.",
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-18T14:22:00.000Z"
  },
  "message": "Chargeback retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 32,
    "api_version": "v1",
    "endpoint": "GET /chargebacks/:chargebackId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`CHARGEBACK_NOT_FOUND`	404	Chargeback with the specified ID does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/chargebacks/cb_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/chargebacks/cb_a1b2c3d4e5f6g7h8', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
});

const result = await response.json();
console.log(result.data.chargebackId); // cb_a1b2c3d4e5f6g7h8
console.log(result.data.evidence); // [{ type: "receipt", ... }]
console.log(result.data.merchantResponse); // "Customer received the product..."
```

SDK

```
const chargeback = await zenpays.chargebacks.get('cb_a1b2c3d4e5f6g7h8');

console.log(chargeback.chargebackId); // cb_a1b2c3d4e5f6g7h8
console.log(chargeback.evidence); // [{ type: "receipt", ... }]
console.log(chargeback.merchantResponse); // "Customer received the product..."
```

Related Endpoints

- List Chargebacks
- Submit Evidence
- Add Response
- Create Chargeback

REST API / CHARGEBACKS

Chargeback Stats

Retrieve aggregated chargeback statistics including totals, win/loss rates, and urgent deadlines.

`GET` **`/merchant/api/v1/chargebacks/stats`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Query Parameters

Parameter	Type	Description
`startDate`	`string`	Start date filter (ISO 8601)
`endDate`	`string`	End date filter (ISO 8601)
`currency`	`string`	Filter by ISO 4217 currency code

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "totalChargebacks": 142,
    "activeDisputes": 18,
    "disputedAmount": 12450.75,
    "wonCount": 87,
    "lostCount": 34,
    "winRate": 71.9,
    "urgentDeadlines": 5
  },
  "message": "Chargeback stats retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 55,
    "api_version": "v1",
    "endpoint": "GET /chargebacks/stats",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/chargebacks/stats?startDate=2024-01-
01T00:00:00.000Z&endDate=2024-01-31T23:59:59.000Z&currency=USD" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-01-31T23:59:59.000Z',
  currency: 'USD',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/chargebacks/stats?
${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
});

const result = await response.json();
console.log(result.data.totalChargebacks); // 142
console.log(result.data.winRate); // 71.9
console.log(result.data.urgentDeadlines); // 5
```

SDK

```
const stats = await zenpays.chargebacks.stats({
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-01-31T23:59:59.000Z',
  currency: 'USD',
});

console.log(stats.totalChargebacks); // 142
console.log(stats.winRate); // 71.9
console.log(stats.urgentDeadlines); // 5
```

Related Endpoints

- List Chargebacks
- Chargeback Analytics
- Get Chargeback

REST API / CHARGEBACKS

Chargeback Analytics

Retrieve time-series chargeback analytics data with configurable granularity.

`GET` **`/merchant/api/v1/chargebacks/analytics`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Query Parameters

Parameter	Type	Description
`startDate`	`string`	Start date filter (ISO 8601)
`endDate`	`string`	End date filter (ISO 8601)
`granularity`	`string`	Time bucket size: `day`, `week`, `month` (default: `day`)
`currency`	`string`	Filter by ISO 4217 currency code

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "granularity": "day",
    "currency": "USD",
    "timeSeries": [
      {
        "date": "2024-01-13",
        "totalChargebacks": 3,
        "totalAmount": 450.00,
        "wonCount": 1,
        "lostCount": 1,
        "pendingCount": 1
      },
      {
        "date": "2024-01-14",
        "totalChargebacks": 5,
        "totalAmount": 820.50,
        "wonCount": 3,
        "lostCount": 0,
        "pendingCount": 2
      },
      {
        "date": "2024-01-15",
        "totalChargebacks": 2,
        "totalAmount": 175.25,
        "wonCount": 0,
        "lostCount": 1,
        "pendingCount": 1
      }
    ]
  },
  "message": "Chargeback analytics retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 68,
    "api_version": "v1",
    "endpoint": "GET /chargebacks/analytics",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/chargebacks/analytics?startDate=2024-01-01T00:00:00.000Z&endDate=2024-01-31T23:59:59.000Z&granularity=day&currency=USD" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-01-31T23:59:59.000Z',
  granularity: 'day',
  currency: 'USD',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/chargebacks/analytics?${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
});

const result = await response.json();
console.log(result.data.timeSeries); // [{ date: "2024-01-13", totalChargebacks: 3, ... }]
console.log(result.data.granularity); // "day"
```

SDK

```
const analytics = await zenpays.chargebacks.analytics({
  startDate: '2024-01-01T00:00:00.000Z',
  endDate: '2024-01-31T23:59:59.000Z',
  granularity: 'day',
  currency: 'USD',
});

console.log(analytics.timeSeries); // [{ date: "2024-01-13", totalChargebacks: 3, ... }]
console.log(analytics.granularity); // "day"
```

Related Endpoints

- Chargeback Stats
- List Chargebacks
- Get Chargeback

REST API / CHARGEBACKS

Submit Evidence

Submit supporting evidence for a chargeback dispute. At least one evidence item is required.

`POST` **`/merchant/api/v1/chargebacks/:chargebackId/evidence`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Path Parameters

Parameter	Type	Required	Description
`chargebackId`	`string`	Yes	Unique chargeback identifier

Body Parameters

Parameter	Type	Required	Description
`evidence`	`array`	Yes	Array of evidence items (minimum 1)
`evidence[].type`	`string`	Yes	Evidence type (e.g. `receipt`, `shipping_proof`, `communication_log`, `refund_policy`)
`evidence[].documentUrl`	`string`	Yes	URL of the uploaded evidence document
`evidence[].fileName`	`string`	No	Original file name
`evidence[].mimeType`	`string`	No	MIME type of the document
`evidence[].fileSize`	`number`	No	File size in bytes
`evidence[].description`	`string`	No	Description of the evidence
`notes`	`string`	No	Additional notes to accompany the evidence submission

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "chargebackId": "cb_a1b2c3d4e5f6g7h8",
    "status": "evidence_submitted",
    "evidence": [
      {
        "type": "receipt",
        "documentUrl": "https://files.zenpays.com/evidence/receipt_001.pdf",
        "fileName": "receipt_001.pdf",
        "mimeType": "application/pdf",
        "fileSize": 204800,
        "description": "Original transaction receipt",
        "submittedAt": "2024-01-18T14:22:00.000Z"
      },
      {
        "type": "shipping_proof",
        "documentUrl": "https://files.zenpays.com/evidence/tracking_001.pdf",
        "fileName": "tracking_001.pdf",
        "mimeType": "application/pdf",
        "fileSize": 153600,
        "description": "Shipping confirmation with tracking number",
        "submittedAt": "2024-01-18T14:22:00.000Z"
      }
    ],
    "notes": "Product was delivered and signed for on Jan 12.",
    "updatedAt": "2024-01-18T14:22:00.000Z"
  },
  "message": "Evidence submitted successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-18T14:22:00.000Z",
    "processing_time_ms": 62,
    "api_version": "v1",
    "endpoint": "POST /chargebacks/:chargebackId/evidence",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters or empty evidence array
`UNAUTHORIZED`	401	Invalid API key
`CHARGEBACK_NOT_FOUND`	404	Chargeback with the specified ID does not exist
`INVALID_STATE`	409	Chargeback is not in a state that accepts evidence
`DEADLINE_EXPIRED`	410	Evidence submission deadline has passed

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/chargebacks/cb_a1b2c3d4e5f6g7h8/evidence \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-18T14:22:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx" \
  -d '{
    "evidence": [
      {
        "type": "receipt",
        "documentUrl": "https://files.zenpays.com/evidence/receipt_001.pdf",
        "fileName": "receipt_001.pdf",
        "mimeType": "application/pdf",
        "fileSize": 204800,
        "description": "Original transaction receipt"
      },
      {
        "type": "shipping_proof",
        "documentUrl": "https://files.zenpays.com/evidence/tracking_001.pdf",
        "fileName": "tracking_001.pdf",
        "mimeType": "application/pdf",
        "fileSize": 153600,
        "description": "Shipping confirmation with tracking number"
      }
    ],
    "notes": "Product was delivered and signed for on Jan 12."
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/chargebacks/cb_a1b2c3d4e5f6g7h8/evidence', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
  body: JSON.stringify({
    evidence: [
      {
        type: 'receipt',
        documentUrl: 'https://files.zenpays.com/evidence/receipt_001.pdf',
        fileName: 'receipt_001.pdf',
        mimeType: 'application/pdf',
        fileSize: 204800,
        description: 'Original transaction receipt',
      },
      {
        type: 'shipping_proof',
        documentUrl: 'https://files.zenpays.com/evidence/tracking_001.pdf',
        fileName: 'tracking_001.pdf',
        mimeType: 'application/pdf',
        fileSize: 153600,
        description: 'Shipping confirmation with tracking number',
      },
    ],
    notes: 'Product was delivered and signed for on Jan 12.',
  }),
});

const result = await response.json();
console.log(result.data.status); // "evidence_submitted"
console.log(result.data.evidence.length); // 2
```

SDK

```
const result = await zenpays.chargebacks.submitEvidence('cb_a1b2c3d4e5f6g7h8', {
  evidence: [
    {
      type: 'receipt',
      documentUrl: 'https://files.zenpays.com/evidence/receipt_001.pdf',
      fileName: 'receipt_001.pdf',
      mimeType: 'application/pdf',
      fileSize: 204800,
      description: 'Original transaction receipt',
    },
    {
      type: 'shipping_proof',
      documentUrl: 'https://files.zenpays.com/evidence/tracking_001.pdf',
      fileName: 'tracking_001.pdf',
      mimeType: 'application/pdf',
      fileSize: 153600,
      description: 'Shipping confirmation with tracking number',
    },
  ],
  notes: 'Product was delivered and signed for on Jan 12.',
});

console.log(result.status); // "evidence_submitted"
console.log(result.evidence.length); // 2
```

Related Endpoints

- Get Chargeback
- Add Response
- Create Chargeback

REST API / CHARGEBACKS

Add Response

Add a merchant response to a chargeback dispute.

`POST` **`/merchant/api/v1/chargebacks/:chargebackId/response`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Path Parameters

Parameter	Type	Required	Description
`chargebackId`	`string`	Yes	Unique chargeback identifier

Body Parameters

Parameter	Type	Required	Description
`response`	`string`	Yes	Merchant response text (must be non-empty)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "chargebackId": "cb_a1b2c3d4e5f6g7h8",
    "status": "pending_review",
    "merchantResponse": "The customer received the product on January 12th. Delivery was confirmed
with a signature. The tracking number is 1Z999AA10123456784. We have attached the signed delivery
receipt and shipping confirmation as evidence.",
    "updatedAt": "2024-01-19T09:15:00.000Z"
  },
  "message": "Response added successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-19T09:15:00.000Z",
    "processing_time_ms": 40,
    "api_version": "v1",
    "endpoint": "POST /chargebacks/:chargebackId/response",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters or empty response
`UNAUTHORIZED`	401	Invalid API key
`CHARGEBACK_NOT_FOUND`	404	Chargeback with the specified ID does not exist
`INVALID_STATE`	409	Chargeback is not in a state that accepts responses

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/chargebacks/cb_a1b2c3d4e5f6g7h8/response \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-19T09:15:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx" \
  -d '{
    "response": "The customer received the product on January 12th. Delivery was confirmed with a
signature. The tracking number is 1Z999AA10123456784."
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/chargebacks/cb_a1b2c3d4e5f6g7h8/response', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
  body: JSON.stringify({
    response: 'The customer received the product on January 12th. Delivery was confirmed with a
signature. The tracking number is 1Z999AA10123456784.',
  }),
});

const result = await response.json();
console.log(result.data.chargebackId); // cb_a1b2c3d4e5f6g7h8
console.log(result.data.merchantResponse); // "The customer received the product..."
```

SDK

```
const result = await zenpays.chargebacks.addResponse('cb_a1b2c3d4e5f6g7h8', {
  response: 'The customer received the product on January 12th. Delivery was confirmed with a
signature. The tracking number is 1Z999AA10123456784.',
});

console.log(result.chargebackId); // cb_a1b2c3d4e5f6g7h8
console.log(result.merchantResponse); // "The customer received the product..."
```

Related Endpoints

- Get Chargeback
- Submit Evidence
- List Chargebacks

Bulk Chargebacks

REST API / CHARGEBACKS / BULK CHARGEBACKS

Bulk Upload Chargebacks

Upload a CSV or Excel file to create multiple chargebacks in a single batch operation.

`POST` **`/merchant/api/v1/chargebacks/bulk`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`multipart/form-data`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Body Parameters

Parameter	Type	Required	Description
`file`	`File`	Yes	CSV or Excel file containing chargeback records (max 10MB)
`webhookUrl`	`string`	No	URL to receive a webhook notification when processing completes

Accepted File Types

MIME Type	Extension
`text/csv`	`.csv`
`application/vnd.ms-excel`	`.xls`
`application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`	`.xlsx`

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "batchId": "batch_a1b2c3d4e5f6g7h8",
    "status": "processing",
    "totalRecords": 150,
    "processedRecords": 0,
    "failedRecords": 0,
    "createdAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Bulk upload accepted for processing",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 120,
    "api_version": "v1",
    "endpoint": "POST /chargebacks/bulk",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid file format or missing required file
`UNAUTHORIZED`	401	Invalid API key
`FILE_TOO_LARGE`	413	File exceeds the 10MB size limit
`UNSUPPORTED_MEDIA_TYPE`	415	File type is not CSV, XLS, or XLSX

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/chargebacks/bulk \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx" \
  -F "file=@chargebacks.csv" \
  -F "webhookUrl=https://example.com/webhooks/bulk-status"
```

JAVASCRIPT

```
const formData = new FormData();
formData.append('file', fileInput.files[0]);
formData.append('webhookUrl', 'https://example.com/webhooks/bulk-status');

const response = await fetch('https://api.zenpays.com/merchant/api/v1/chargebacks/bulk', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
  body: formData,
});

const result = await response.json();
console.log(result.data.batchId); // "batch_a1b2c3d4e5f6g7h8"
console.log(result.data.status); // "processing"
```

SDK

```
const result = await zenpays.chargebacks.bulkUpload({
  file: fs.createReadStream('chargebacks.csv'),
  webhookUrl: 'https://example.com/webhooks/bulk-status',
});

console.log(result.batchId); // "batch_a1b2c3d4e5f6g7h8"
console.log(result.status); // "processing"
```

Related Endpoints

- Bulk Upload Status
- Create Chargeback
- List Chargebacks

REST API / CHARGEBACKS / BULK CHARGEBACKS

Bulk Upload Status

Check the processing status of a bulk chargeback upload batch.

`GET` **`/merchant/api/v1/chargebacks/bulk/:batchId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`
`X-Merchant-Id`	Yes	Merchant identifier
`X-Request-Id`	Yes	Unique request identifier

Path Parameters

Parameter	Type	Required	Description
`batchId`	`string`	Yes	Unique batch identifier returned from the bulk upload

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "batchId": "batch_a1b2c3d4e5f6g7h8",
    "status": "completed",
    "totalRecords": 150,
    "processedRecords": 147,
    "failedRecords": 3,
    "createdAt": "2024-01-15T10:30:00.000Z",
    "completedAt": "2024-01-15T10:32:45.000Z"
  },
  "message": "Batch status retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:33:00.000Z",
    "processing_time_ms": 28,
    "api_version": "v1",
    "endpoint": "GET /chargebacks/bulk/:batchId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`BATCH_NOT_FOUND`	404	Batch with the specified ID does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/chargebacks/bulk/batch_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:33:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -H "X-Merchant-Id: merch_xxxxx" \
  -H "X-Request-Id: req_xxxxx"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/chargebacks/bulk/batch_a1b2c3d4e5f6g7h8', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
    'X-Merchant-Id': merchantId,
    'X-Request-Id': requestId,
  },
});

const result = await response.json();
console.log(result.data.batchId); // "batch_a1b2c3d4e5f6g7h8"
console.log(result.data.status); // "completed"
console.log(result.data.processedRecords); // 147
console.log(result.data.failedRecords); // 3
```

SDK

```
const batch = await zenpays.chargebacks.bulkUploadStatus('batch_a1b2c3d4e5f6g7h8');

console.log(batch.batchId); // "batch_a1b2c3d4e5f6g7h8"
console.log(batch.status); // "completed"
console.log(batch.processedRecords); // 147
console.log(batch.failedRecords); // 3
```

Related Endpoints

- Bulk Upload Chargebacks
- List Chargebacks
- Get Chargeback

Billing

Products

REST API / BILLING / PRODUCTS

Create Product

Create a new product for billing and invoicing.

`POST` **`/merchant/api/v1/products`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`name`	`string`	Yes	Product name (1-255 characters)
`description`	`string`	No	Product description
`price`	`number`	Yes	Product price (minimum 0)
`currency`	`string`	No	Currency code (default: `INR`). One of `USD`, `INR`, `EUR`, `GBP`
`sku`	`string`	No	Stock keeping unit (1-100 characters)
`status`	`string`	No	Product status (default: `ACTIVE`). One of `ACTIVE`, `INACTIVE`, `ARCHIVED`
`categoryId`	`number`	No	Category ID to assign the product to
`images`	`array`	No	Array of product images
`images[].url`	`string`	Yes	Image URL
`images[].alt`	`string`	No	Alt text for the image
`images[].isPrimary`	`boolean`	No	Whether this is the primary image
`pricingTiers`	`array`	No	Array of pricing tiers

Parameter	Type	Required	Description
`pricingTiers[].name`	`string`	Yes	Tier name
`pricingTiers[].price`	`number`	Yes	Tier price (minimum 0)
`pricingTiers[].description`	`string`	No	Tier description
`pricingTiers[].features`	`string[]`	No	List of features included in the tier
`metadata`	`object`	No	Custom key-value pairs

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "name": "Pro Plan",
    "description": "Professional subscription plan with advanced features",
    "price": 4999,
    "currency": "INR",
    "sku": "PRO-PLAN-001",
    "status": "ACTIVE",
    "categoryId": 1,
    "images": [
      {
        "url": "https://example.com/images/pro-plan.png",
        "alt": "Pro Plan",
        "isPrimary": true
      }
    ],
    "pricingTiers": [
      {
        "name": "Monthly",
        "price": 4999,
        "description": "Billed monthly",
        "features": ["Unlimited access", "Priority support"]
      }
    ],
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Product created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 45,
    "api_version": "v1",
    "endpoint": "POST /products",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`DUPLICATE_SKU`	409	Product with this SKU already exists

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/products \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Pro Plan",
    "description": "Professional subscription plan with advanced features",
    "price": 4999,
    "currency": "INR",
    "sku": "PRO-PLAN-001",
    "status": "ACTIVE",
    "categoryId": 1,
    "images": [
      {
        "url": "https://example.com/images/pro-plan.png",
        "alt": "Pro Plan",
        "isPrimary": true
      }
    ],
    "pricingTiers": [
      {
        "name": "Monthly",
        "price": 4999,
        "description": "Billed monthly",
        "features": ["Unlimited access", "Priority support"]
      }
    ]
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/products', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Pro Plan',
    description: 'Professional subscription plan with advanced features',
    price: 4999,
    currency: 'INR',
    sku: 'PRO-PLAN-001',
    status: 'ACTIVE',
    categoryId: 1,
    images: [
      {
        url: 'https://example.com/images/pro-plan.png',
        alt: 'Pro Plan',
        isPrimary: true,
      },
    ],
    pricingTiers: [
      {
        name: 'Monthly',
        price: 4999,
        description: 'Billed monthly',
        features: ['Unlimited access', 'Priority support'],
      },
    ],
  }),
});

const result = await response.json();
console.log(result.data.productId); // prod_a1b2c3d4e5f6g7h8
```

SDK

```
const product = await zenpays.products.create({
  name: 'Pro Plan',
  description: 'Professional subscription plan with advanced features',
  price: 4999,
  currency: 'INR',
  sku: 'PRO-PLAN-001',
  status: 'ACTIVE',
  categoryId: 1,
  images: [
    {
      url: 'https://example.com/images/pro-plan.png',
      alt: 'Pro Plan',
      isPrimary: true,
    },
  ],
  pricingTiers: [
    {
      name: 'Monthly',
      price: 4999,
      description: 'Billed monthly',
      features: ['Unlimited access', 'Priority support'],
    },
  ],
});

console.log(product.productId); // prod_a1b2c3d4e5f6g7h8
```

Related Endpoints

- List Products
- Get Product
- Update Product
- Delete Product

REST API / BILLING / PRODUCTS

List Products

Retrieve a paginated list of products with optional filtering and sorting.

`GET` **`/merchant/api/v1/products`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Query Parameters

Parameter	Type	Description
`status`	`string`	Filter by status: `ACTIVE`, `INACTIVE`, `ARCHIVED`
`currency`	`string`	Filter by currency code
`categoryId`	`number`	Filter by category ID
`page`	`number`	Page number (default: `1`)
`limit`	`number`	Items per page (default: `20`, max: `100`)
`sortBy`	`string`	Sort field (default: `createdAt`)
`sortOrder`	`string`	Sort direction: `ASC` or `DESC` (default: `DESC`)
`searchTerm`	`string`	Search by product name or SKU

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "items": [
      {
        "productId": "prod_a1b2c3d4e5f6g7h8",
        "name": "Pro Plan",
        "description": "Professional subscription plan with advanced features",
        "price": 4999,
        "currency": "INR",
        "sku": "PRO-PLAN-001",
        "status": "ACTIVE",
        "categoryId": 1,
        "images": [
          {
            "url": "https://example.com/images/pro-plan.png",
            "alt": "Pro Plan",
            "isPrimary": true
          }
        ],
        "pricingTiers": [],
        "metadata": {},
        "createdAt": "2024-01-15T10:30:00.000Z",
        "updatedAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "total": 1,
    "page": 1,
    "limit": 20,
    "totalPages": 1
  },
  "message": "Products retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 35,
    "api_version": "v1",
    "endpoint": "GET /products",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/products?status=ACTIVE&page=1&limit=20" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  status: 'ACTIVE',
  page: '1',
  limit: '20',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/products?${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.items); // Array of products
console.log(result.data.total); // Total count
```

SDK

```
const products = await zenpays.products.list({
  status: 'ACTIVE',
  page: 1,
  limit: 20,
});

console.log(products.items); // Array of products
console.log(products.total); // Total count
```

Related Endpoints

- Create Product
- Get Product
- Update Product
- Delete Product

REST API / BILLING / PRODUCTS

Get Product

Retrieve a single product by its ID.

`GET` **`/merchant/api/v1/products/:productId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`productId`	`string`	Yes	The product ID (e.g. `prod_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "name": "Pro Plan",
    "description": "Professional subscription plan with advanced features",
    "price": 4999,
    "currency": "INR",
    "sku": "PRO-PLAN-001",
    "status": "ACTIVE",
    "categoryId": 1,
    "images": [
      {
        "url": "https://example.com/images/pro-plan.png",
        "alt": "Pro Plan",
        "isPrimary": true
      }
    ],
    "pricingTiers": [
      {
        "name": "Monthly",
        "price": 4999,
        "description": "Billed monthly",
        "features": ["Unlimited access", "Priority support"]
      }
    ],
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Product retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 25,
    "api_version": "v1",
    "endpoint": "GET /products/:productId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`PRODUCT_NOT_FOUND`	404	Product does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/products/prod_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/products/prod_a1b2c3d4e5f6g7h8', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.name); // "Pro Plan"
```

SDK

```
const product = await zenpays.products.get('prod_a1b2c3d4e5f6g7h8');

console.log(product.name); // "Pro Plan"
```

Related Endpoints

- Create Product
- List Products
- Update Product
- Delete Product

REST API / BILLING / PRODUCTS

Update Product

Update an existing product. All fields are optional — only provided fields will be updated.

`PUT` **`/merchant/api/v1/products/:productId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`productId`	`string`	Yes	The product ID (e.g. `prod_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Required	Description
`name`	`string`	No	Product name (1-255 characters)
`description`	`string`	No	Product description
`price`	`number`	No	Product price (minimum 0)
`currency`	`string`	No	Currency code. One of `USD`, `INR`, `EUR`, `GBP`
`sku`	`string`	No	Stock keeping unit (1-100 characters)
`status`	`string`	No	Product status. One of `ACTIVE`, `INACTIVE`, `ARCHIVED`
`categoryId`	`number`	No	Category ID to assign the product to
`images`	`array`	No	Array of product images
`images[].url`	`string`	Yes	Image URL

Parameter	Type	Required	Description
`images[].alt`	`string`	No	Alt text for the image
`images[].isPrimary`	`boolean`	No	Whether this is the primary image
`pricingTiers`	`array`	No	Array of pricing tiers
`pricingTiers[].name`	`string`	Yes	Tier name
`pricingTiers[].price`	`number`	Yes	Tier price (minimum 0)
`pricingTiers[].description`	`string`	No	Tier description
`pricingTiers[].features`	`string[]`	No	List of features included in the tier
`metadata`	`object`	No	Custom key-value pairs

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "name": "Pro Plan Plus",
    "description": "Enhanced professional subscription plan",
    "price": 5999,
    "currency": "INR",
    "sku": "PRO-PLAN-001",
    "status": "ACTIVE",
    "categoryId": 1,
    "images": [
      {
        "url": "https://example.com/images/pro-plan-plus.png",
        "alt": "Pro Plan Plus",
        "isPrimary": true
      }
    ],
    "pricingTiers": [
      {
        "name": "Monthly",
        "price": 5999,
        "description": "Billed monthly",
        "features": ["Unlimited access", "Priority support", "API access"]
      }
    ],
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-16T14:20:00.000Z"
  },
  "message": "Product updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 40,
    "api_version": "v1",
    "endpoint": "PUT /products/:productId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`PRODUCT_NOT_FOUND`	404	Product does not exist
`DUPLICATE_SKU`	409	Product with this SKU already exists

Examples

CURL

```
curl -X PUT https://api.zenpays.com/merchant/api/v1/products/prod_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Pro Plan Plus",
    "price": 5999,
    "description": "Enhanced professional subscription plan"
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/products/prod_a1b2c3d4e5f6g7h8', {
  method: 'PUT',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Pro Plan Plus',
    price: 5999,
    description: 'Enhanced professional subscription plan',
  }),
});

const result = await response.json();
console.log(result.data.name); // "Pro Plan Plus"
```

SDK

```
const product = await zenpays.products.update('prod_a1b2c3d4e5f6g7h8', {
  name: 'Pro Plan Plus',
  price: 5999,
  description: 'Enhanced professional subscription plan',
});

console.log(product.name); // "Pro Plan Plus"
```

Related Endpoints

- Create Product
- Get Product
- List Products
- Delete Product

REST API / BILLING / PRODUCTS

Delete Product

Archive a product (soft delete). The product will be set to `ARCHIVED` status and will no longer appear in active product listings.

`DELETE` **`/merchant/api/v1/products/:productId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`productId`	`string`	Yes	The product ID (e.g. `prod_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "status": "ARCHIVED"
  },
  "message": "Product archived successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 30,
    "api_version": "v1",
    "endpoint": "DELETE /products/:productId",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`PRODUCT_NOT_FOUND`	404	Product does not exist

Examples

CURL

```
curl -X DELETE https://api.zenpays.com/merchant/api/v1/products/prod_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/products/prod_a1b2c3d4e5f6g7h8', {
  method: 'DELETE',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.status); // "ARCHIVED"
```

SDK

```
const result = await zenpays.products.delete('prod_a1b2c3d4e5f6g7h8');

console.log(result.status); // "ARCHIVED"
```

Related Endpoints

- Create Product
- Get Product
- List Products
- Update Product

REST API / BILLING / PRODUCTS

List Categories

Retrieve all product categories.

`GET` **`/merchant/api/v1/products/categories`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "SaaS Plans",
      "description": "Software-as-a-service subscription plans",
      "sortOrder": 0,
      "createdAt": "2024-01-10T08:00:00.000Z",
      "updatedAt": "2024-01-10T08:00:00.000Z"
    },
    {
      "id": 2,
      "name": "Digital Goods",
      "description": "Downloadable digital products",
      "sortOrder": 1,
      "createdAt": "2024-01-10T08:00:00.000Z",
      "updatedAt": "2024-01-10T08:00:00.000Z"
    }
  ],
  "message": "Categories retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 20,
    "api_version": "v1",
    "endpoint": "GET /products/categories",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/products/categories \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/products/categories', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data); // Array of categories
```

SDK

```
const categories = await zenpays.products.listCategories();

console.log(categories); // Array of categories
```

Related Endpoints

- Manage Categories
- Create Product
- List Products

REST API / BILLING / PRODUCTS

Manage Categories

Create, update, and delete product categories.

Create Category

Endpoint

```
POST /merchant/api/v1/products/categories
```

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`name`	`string`	Yes	Category name (1-100 characters)
`description`	`string`	No	Category description
`sortOrder`	`number`	No	Display sort order (default: `0`)

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "id": 1,
    "name": "SaaS Plans",
    "description": "Software-as-a-service subscription plans",
    "sortOrder": 0,
    "createdAt": "2024-01-10T08:00:00.000Z",
    "updatedAt": "2024-01-10T08:00:00.000Z"
  },
  "message": "Category created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-10T08:00:00.000Z",
    "processing_time_ms": 30,
    "api_version": "v1",
    "endpoint": "POST /products/categories",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`DUPLICATE_CATEGORY`	409	Category with this name already exists

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/products/categories \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-10T08:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "SaaS Plans",
    "description": "Software-as-a-service subscription plans",
    "sortOrder": 0
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/products/categories', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'SaaS Plans',
    description: 'Software-as-a-service subscription plans',
    sortOrder: 0,
  }),
});

const result = await response.json();
console.log(result.data.id); // 1
```

SDK

```
const category = await zenpays.products.createCategory({
  name: 'SaaS Plans',
  description: 'Software-as-a-service subscription plans',
  sortOrder: 0,
});

console.log(category.id); // 1
```

Update Category

Endpoint

```
PUT /merchant/api/v1/products/categories/:id
```

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`number`	Yes	The category ID

Body Parameters

Parameter	Type	Description
`name`	`string`	Category name (1-100 characters)
`description`	`string`	Category description
`sortOrder`	`number`	Display sort order

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "id": 1,
    "name": "SaaS Subscriptions",
    "description": "Software-as-a-service subscription plans",
    "sortOrder": 0,
    "createdAt": "2024-01-10T08:00:00.000Z",
    "updatedAt": "2024-01-16T14:20:00.000Z"
  },
  "message": "Category updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 25,
    "api_version": "v1",
    "endpoint": "PUT /products/categories/:id",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`CATEGORY_NOT_FOUND`	404	Category does not exist
`DUPLICATE_CATEGORY`	409	Category with this name already exists

Examples

CURL

```
curl -X PUT https://api.zenpays.com/merchant/api/v1/products/categories/1 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "SaaS Subscriptions"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/products/categories/1', {
  method: 'PUT',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'SaaS Subscriptions',
  }),
});

const result = await response.json();
console.log(result.data.name); // "SaaS Subscriptions"
```

SDK

```
const category = await zenpays.products.updateCategory(1, {
  name: 'SaaS Subscriptions',
});

console.log(category.name); // "SaaS Subscriptions"
```

Delete Category

Endpoint

```
DELETE /merchant/api/v1/products/categories/:id
```

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`id`	`number`	Yes	The category ID

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "id": 1
  },
  "message": "Category deleted successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 20,
    "api_version": "v1",
    "endpoint": "DELETE /products/categories/:id",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`CATEGORY_NOT_FOUND`	404	Category does not exist
`CATEGORY_IN_USE`	409	Category has products assigned to it

Examples

CURL

```
curl -X DELETE https://api.zenpays.com/merchant/api/v1/products/categories/1 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/products/categories/1', {
  method: 'DELETE',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.message); // "Category deleted successfully"
```

SDK

```
await zenpays.products.deleteCategory(1);
```

Related Endpoints

- List Categories
- Create Product
- List Products

Invoices

REST API / BILLING / INVOICES

Create Invoice

Create a new invoice for a customer with line items.

`POST` **`/merchant/api/v1/invoices`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`customerId`	`string`	Yes	Customer ID
`customerEmail`	`string`	Yes	Customer email address
`customerName`	`string`	No	Customer display name (1-255 characters)
`currency`	`string`	No	Currency code (default: `USD`)
`issueDate`	`string`	Yes	Invoice issue date (ISO 8601)
`dueDate`	`string`	Yes	Payment due date (ISO 8601)
`taxRate`	`number`	No	Tax rate percentage (minimum 0, default: `0`)
`discountAmount`	`number`	No	Discount amount (minimum 0, default: `0`)
`notes`	`string`	No	Notes displayed on the invoice
`terms`	`string`	No	Payment terms and conditions
`lineItems`	`array`	Yes	Array of line items (minimum 1)
`lineItems[].productId`	`string`	No	Associated product ID

Parameter	Type	Required	Description
`lineItems[].description`	`string`	Yes	Line item description (1-255 characters)
`lineItems[].quantity`	`number`	Yes	Quantity (minimum 1, default: `1`)
`lineItems[].unitPrice`	`number`	Yes	Unit price (minimum 0)
`lineItems[].taxRate`	`number`	No	Item-level tax rate (minimum 0, default: `0`)
`lineItems[].sortOrder`	`number`	No	Display order (default: `0`)
`metadata`	`object`	No	Custom key-value pairs

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "invoiceNumber": "INV-2024-0001",
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "customerEmail": "john@example.com",
    "customerName": "John Doe",
    "currency": "USD",
    "status": "DRAFT",
    "issueDate": "2024-01-15T00:00:00.000Z",
    "dueDate": "2024-02-15T00:00:00.000Z",
    "subtotal": 250.00,
    "taxRate": 10,
    "taxAmount": 25.00,
    "discountAmount": 0,
    "totalAmount": 275.00,
    "amountPaid": 0,
    "amountDue": 275.00,
    "notes": "Thank you for your business",
    "terms": "Net 30",
    "lineItems": [
      {
        "id": "li_001",
        "productId": "prod_a1b2c3d4e5f6g7h8",
        "description": "Pro Plan - Monthly",
        "quantity": 5,
        "unitPrice": 50.00,
        "taxRate": 0,
        "amount": 250.00,
        "sortOrder": 0
      }
    ],
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Invoice created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 65,
    "api_version": "v1",
    "endpoint": "POST /invoices",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`CUSTOMER_NOT_FOUND`	404	Customer does not exist

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/invoices \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "customerEmail": "john@example.com",
    "customerName": "John Doe",
    "currency": "USD",
    "issueDate": "2024-01-15T00:00:00.000Z",
    "dueDate": "2024-02-15T00:00:00.000Z",
    "taxRate": 10,
    "notes": "Thank you for your business",
    "terms": "Net 30",
    "lineItems": [
      {
        "productId": "prod_a1b2c3d4e5f6g7h8",
        "description": "Pro Plan - Monthly",
        "quantity": 5,
        "unitPrice": 50.00
      }
    ]
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/invoices', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    customerId: 'cust_a1b2c3d4e5f6g7h8',
    customerEmail: 'john@example.com',
    customerName: 'John Doe',
    currency: 'USD',
    issueDate: '2024-01-15T00:00:00.000Z',
    dueDate: '2024-02-15T00:00:00.000Z',
    taxRate: 10,
    notes: 'Thank you for your business',
    terms: 'Net 30',
    lineItems: [
      {
        productId: 'prod_a1b2c3d4e5f6g7h8',
        description: 'Pro Plan - Monthly',
        quantity: 5,
        unitPrice: 50.00,
      },
    ],
  }),
});

const result = await response.json();
console.log(result.data.invoiceId); // inv_a1b2c3d4e5f6g7h8
```

SDK

```
const invoice = await zenpays.invoices.create({
  customerId: 'cust_a1b2c3d4e5f6g7h8',
  customerEmail: 'john@example.com',
  customerName: 'John Doe',
  currency: 'USD',
  issueDate: '2024-01-15T00:00:00.000Z',
  dueDate: '2024-02-15T00:00:00.000Z',
  taxRate: 10,
  notes: 'Thank you for your business',
  terms: 'Net 30',
  lineItems: [
    {
      productId: 'prod_a1b2c3d4e5f6g7h8',
      description: 'Pro Plan - Monthly',
      quantity: 5,
      unitPrice: 50.00,
    },
  ],
});

console.log(invoice.invoiceId); // inv_a1b2c3d4e5f6g7h8
```

Related Endpoints

- List Invoices

- Get Invoice
- Send Invoice
- Invoice Analytics

REST API / BILLING / INVOICES

List Invoices

Retrieve a paginated list of invoices with optional filtering and sorting.

`GET` **`/merchant/api/v1/invoices`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Query Parameters

Parameter	Type	Description
`status`	`string`	Filter by status: `DRAFT`, `SENT`, `VIEWED`, `PARTIALLY_PAID`, `PAID`, `OVERDUE`, `CANCELLED`, `VOID`
`currency`	`string`	Filter by currency code
`customerId`	`string`	Filter by customer ID
`startDate`	`string`	Filter invoices issued on or after this date (ISO 8601)
`endDate`	`string`	Filter invoices issued on or before this date (ISO 8601)
`page`	`number`	Page number (default: `1`)
`limit`	`number`	Items per page (default: `20`, max: `100`)
`sortBy`	`string`	Sort field (default: `createdAt`)
`sortOrder`	`string`	Sort direction: `ASC` or `DESC` (default: `DESC`)
`searchTerm`	`string`	Search by invoice number or customer name

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "items": [
      {
        "invoiceId": "inv_a1b2c3d4e5f6g7h8",
        "invoiceNumber": "INV-2024-0001",
        "customerId": "cust_a1b2c3d4e5f6g7h8",
        "customerEmail": "john@example.com",
        "customerName": "John Doe",
        "currency": "USD",
        "status": "SENT",
        "issueDate": "2024-01-15T00:00:00.000Z",
        "dueDate": "2024-02-15T00:00:00.000Z",
        "totalAmount": 275.00,
        "amountPaid": 0,
        "amountDue": 275.00,
        "createdAt": "2024-01-15T10:30:00.000Z",
        "updatedAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "total": 1,
    "page": 1,
    "limit": 20,
    "totalPages": 1
  },
  "message": "Invoices retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 40,
    "api_version": "v1",
    "endpoint": "GET /invoices",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/invoices?status=SENT&page=1&limit=20" \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  status: 'SENT',
  page: '1',
  limit: '20',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/invoices?${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.items); // Array of invoices
console.log(result.data.total); // Total count
```

SDK

```
const invoices = await zenpays.invoices.list({
  status: 'SENT',
  page: 1,
  limit: 20,
});

console.log(invoices.items); // Array of invoices
console.log(invoices.total); // Total count
```

Related Endpoints

- Create Invoice
- Get Invoice
- Invoice Analytics

REST API / BILLING / INVOICES

Get Invoice

Retrieve a single invoice by its ID, including all line items and payment details.

`GET` **`/merchant/api/v1/invoices/:invoiceId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "invoiceNumber": "INV-2024-0001",
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "customerEmail": "john@example.com",
    "customerName": "John Doe",
    "currency": "USD",
    "status": "SENT",
    "issueDate": "2024-01-15T00:00:00.000Z",
    "dueDate": "2024-02-15T00:00:00.000Z",
    "subtotal": 250.00,
    "taxRate": 10,
    "taxAmount": 25.00,
    "discountAmount": 0,
    "totalAmount": 275.00,
    "amountPaid": 0,
    "amountDue": 275.00,
    "notes": "Thank you for your business",
    "terms": "Net 30",
    "lineItems": [
      {
        "id": "li_001",
        "productId": "prod_a1b2c3d4e5f6g7h8",
        "description": "Pro Plan - Monthly",
        "quantity": 5,
        "unitPrice": 50.00,
        "taxRate": 0,
        "amount": 250.00,
        "sortOrder": 0
      }
    ],
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Invoice retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 30,
    "api_version": "v1",
    "endpoint": "GET /invoices/:invoiceId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`INVOICE_NOT_FOUND`	404	Invoice does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.invoiceNumber); // "INV-2024-0001"
console.log(result.data.totalAmount); // 275.00
```

SDK

```
const invoice = await zenpays.invoices.get('inv_a1b2c3d4e5f6g7h8');

console.log(invoice.invoiceNumber); // "INV-2024-0001"
console.log(invoice.totalAmount); // 275.00
```

Related Endpoints

- Create Invoice
- Update Invoice
- Send Invoice
- Cancel Invoice

REST API / BILLING / INVOICES

Update Invoice

Update an existing invoice. Only provided fields will be updated.

`PUT` **`/merchant/api/v1/invoices/:invoiceId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Required	Description
`dueDate`	`string`	No	Payment due date (ISO 8601)
`taxRate`	`number`	No	Tax rate percentage (minimum 0)
`discountAmount`	`number`	No	Discount amount (minimum 0)
`notes`	`string`	No	Notes displayed on the invoice
`terms`	`string`	No	Payment terms and conditions
`lineItems`	`array`	No	Array of line items
`lineItems[].productId`	`string`	No	Associated product ID
`lineItems[].description`	`string`	Yes	Line item description (1-255 characters)
`lineItems[].quantity`	`number`	Yes	Quantity (minimum 1)

Parameter	Type	Required	Description
`lineItems[].unitPrice`	`number`	Yes	Unit price (minimum 0)
`lineItems[].taxRate`	`number`	No	Item-level tax rate (minimum 0)
`lineItems[].sortOrder`	`number`	No	Display order

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "invoiceNumber": "INV-2024-0001",
    "customerId": "cust_a1b2c3d4e5f6g7h8",
    "customerEmail": "john@example.com",
    "customerName": "John Doe",
    "currency": "USD",
    "status": "DRAFT",
    "issueDate": "2024-01-15T00:00:00.000Z",
    "dueDate": "2024-03-15T00:00:00.000Z",
    "subtotal": 300.00,
    "taxRate": 10,
    "taxAmount": 30.00,
    "discountAmount": 20,
    "totalAmount": 310.00,
    "amountPaid": 0,
    "amountDue": 310.00,
    "notes": "Updated payment terms",
    "terms": "Net 60",
    "lineItems": [
      {
        "id": "li_001",
        "productId": "prod_a1b2c3d4e5f6g7h8",
        "description": "Pro Plan - Monthly",
        "quantity": 6,
        "unitPrice": 50.00,
        "taxRate": 0,
        "amount": 300.00,
        "sortOrder": 0
      }
    ],
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-16T14:20:00.000Z"
  },
  "message": "Invoice updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 50,
    "api_version": "v1",
    "endpoint": "PUT /invoices/:invoiceId",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`INVOICE_NOT_FOUND`	404	Invoice does not exist
`INVOICE_NOT_EDITABLE`	409	Invoice cannot be edited in its current status

Examples

CURL

```
curl -X PUT https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "dueDate": "2024-03-15T00:00:00.000Z",
    "discountAmount": 20,
    "notes": "Updated payment terms",
    "terms": "Net 60",
    "lineItems": [
      {
        "productId": "prod_a1b2c3d4e5f6g7h8",
        "description": "Pro Plan - Monthly",
        "quantity": 6,
        "unitPrice": 50.00
      }
    ]
  }'
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8', {
  method: 'PUT',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    dueDate: '2024-03-15T00:00:00.000Z',
    discountAmount: 20,
    notes: 'Updated payment terms',
    terms: 'Net 60',
    lineItems: [
      {
        productId: 'prod_a1b2c3d4e5f6g7h8',
        description: 'Pro Plan - Monthly',
        quantity: 6,
        unitPrice: 50.00,
      },
    ],
  }),
});

const result = await response.json();
console.log(result.data.totalAmount); // 310.00
```

SDK

```
const invoice = await zenpays.invoices.update('inv_a1b2c3d4e5f6g7h8', {
  dueDate: '2024-03-15T00:00:00.000Z',
  discountAmount: 20,
  notes: 'Updated payment terms',
  terms: 'Net 60',
  lineItems: [
    {
      productId: 'prod_a1b2c3d4e5f6g7h8',
      description: 'Pro Plan - Monthly',
      quantity: 6,
      unitPrice: 50.00,
    },
  ],
});

console.log(invoice.totalAmount); // 310.00
```

Related Endpoints

- Get Invoice
- Send Invoice
- Cancel Invoice

REST API / BILLING / INVOICES

Send Invoice

Send an invoice to the customer via email. The invoice status will be updated to `SENT`.

`POST` **`/merchant/api/v1/invoices/:invoiceId/send`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "status": "SENT",
    "sentAt": "2024-01-15T11:00:00.000Z"
  },
  "message": "Invoice sent successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T11:00:00.000Z",
    "processing_time_ms": 150,
    "api_version": "v1",
    "endpoint": "POST /invoices/:invoiceId/send",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`INVOICE_NOT_FOUND`	404	Invoice does not exist
`INVOICE_ALREADY_SENT`	409	Invoice has already been sent
`INVOICE_CANCELLED`	409	Cannot send a cancelled invoice

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/send \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T11:00:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/send', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.status); // "SENT"
```

SDK

```
const result = await zenpays.invoices.send('inv_a1b2c3d4e5f6g7h8');

console.log(result.status); // "SENT"
```

Related Endpoints

- Get Invoice
- Cancel Invoice
- Generate Payment Link

REST API / BILLING / INVOICES

Cancel Invoice

Cancel an invoice. The invoice status will be updated to `CANCELLED`.

`POST` **`/merchant/api/v1/invoices/:invoiceId/cancel`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "status": "CANCELLED",
    "cancelledAt": "2024-01-16T14:20:00.000Z"
  },
  "message": "Invoice cancelled successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 35,
    "api_version": "v1",
    "endpoint": "POST /invoices/:invoiceId/cancel",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`INVOICE_NOT_FOUND`	404	Invoice does not exist
`INVOICE_ALREADY_CANCELLED`	409	Invoice has already been cancelled
`INVOICE_PAID`	409	Cannot cancel a fully paid invoice

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/cancel \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/cancel', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.status); // "CANCELLED"
```

SDK

```
const result = await zenpays.invoices.cancel('inv_a1b2c3d4e5f6g7h8');

console.log(result.status); // "CANCELLED"
```

Related Endpoints

- Get Invoice
- Send Invoice
- List Invoices

REST API / BILLING / INVOICES

Invoice Analytics

Retrieve invoice analytics including totals and counts by status.

`GET` **`/merchant/api/v1/invoices/analytics`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "totalInvoices": 150,
    "totalAmount": 125000.00,
    "totalPaid": 95000.00,
    "totalOutstanding": 30000.00,
    "byStatus": {
      "DRAFT": 10,
      "SENT": 25,
      "VIEWED": 8,
      "PARTIALLY_PAID": 5,
      "PAID": 85,
      "OVERDUE": 12,
      "CANCELLED": 3,
      "VOID": 2
    },
    "averageInvoiceAmount": 833.33,
    "averagePaymentTime": 14.5
  },
  "message": "Invoice analytics retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 80,
    "api_version": "v1",
    "endpoint": "GET /invoices/analytics",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/invoices/analytics \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/invoices/analytics', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.totalInvoices); // 150
console.log(result.data.totalPaid); // 95000.00
```

SDK

```
const analytics = await zenpays.invoices.analytics();

console.log(analytics.totalInvoices); // 150
console.log(analytics.totalPaid); // 95000.00
```

Related Endpoints

- List Invoices
- Create Invoice

REST API / BILLING / INVOICES

Public Invoice

Retrieve invoice details for customer viewing.

> **Note**
> This is a **public endpoint** — no authentication headers are required.

`GET` **`/merchant/api/v1/invoices/:invoiceId/public`**

Request

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "invoiceNumber": "INV-2024-0001",
    "merchantName": "Acme Corp",
    "merchantLogo": "https://example.com/logo.png",
    "customerName": "John Doe",
    "customerEmail": "john@example.com",
    "currency": "USD",
    "status": "SENT",
    "issueDate": "2024-01-15T00:00:00.000Z",
    "dueDate": "2024-02-15T00:00:00.000Z",
    "subtotal": 250.00,
    "taxRate": 10,
    "taxAmount": 25.00,
    "discountAmount": 0,
    "totalAmount": 275.00,
    "amountPaid": 0,
    "amountDue": 275.00,
    "notes": "Thank you for your business",
    "terms": "Net 30",
    "lineItems": [
      {
        "description": "Pro Plan - Monthly",
        "quantity": 5,
        "unitPrice": 50.00,
        "amount": 250.00
      }
    ]
  },
  "message": "Invoice retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 25,
    "api_version": "v1",
    "endpoint": "GET /invoices/:invoiceId/public",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`INVOICE_NOT_FOUND`	404	Invoice does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/public
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/public', {
  method: 'GET',
});

const result = await response.json();
console.log(result.data.invoiceNumber); // "INV-2024-0001"
console.log(result.data.amountDue); // 275.00
```

SDK

```
const invoice = await zenpays.invoices.getPublic('inv_a1b2c3d4e5f6g7h8');

console.log(invoice.invoiceNumber); // "INV-2024-0001"
console.log(invoice.amountDue); // 275.00
```

Related Endpoints

- Pay Invoice
- Get Invoice

REST API / BILLING / INVOICES

Pay Invoice

Initiate payment for an invoice. Creates a payment intent that the customer can use to complete the payment.

> **Note**
> This is a **public endpoint** — no authentication headers are required.

`POST` **`/merchant/api/v1/invoices/:invoiceId/pay`**

Request

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "paymentIntentId": "pi_x1y2z3w4v5u6t7s8",
    "amount": 275.00,
    "currency": "USD",
    "checkoutUrl": "https://checkout.zenpays.com/pay/pi_x1y2z3w4v5u6t7s8"
  },
  "message": "Payment initiated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 120,
    "api_version": "v1",
    "endpoint": "POST /invoices/:invoiceId/pay",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`INVOICE_NOT_FOUND`	404	Invoice does not exist
`INVOICE_ALREADY_PAID`	409	Invoice has already been fully paid
`INVOICE_CANCELLED`	409	Cannot pay a cancelled invoice

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/pay \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/pay', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.checkoutUrl); // Redirect customer to this URL
```

SDK

```
const payment = await zenpays.invoices.pay('inv_a1b2c3d4e5f6g7h8');

console.log(payment.checkoutUrl); // Redirect customer to this URL
```

Related Endpoints

- Public Invoice
- Get Invoice
- Generate Payment Link

REST API / BILLING / INVOICES

Generate Payment Link

Generate a payment link for an invoice that can be shared with the customer.

`POST` **`/merchant/api/v1/invoices/:invoiceId/generate-link`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`invoiceId`	`string`	Yes	The invoice ID (e.g. `inv_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "invoiceId": "inv_a1b2c3d4e5f6g7h8",
    "paymentLinkId": "pl_b2c3d4e5f6g7h8i9",
    "paymentUrl": "https://pay.zenpays.com/pl_b2c3d4e5f6g7h8i9",
    "shortCode": "ZP-ABC123",
    "shortUrl": "https://pay.zenpays.com/s/ZP-ABC123"
  },
  "message": "Payment link generated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 55,
    "api_version": "v1",
    "endpoint": "POST /invoices/:invoiceId/generate-link",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`INVOICE_NOT_FOUND`	404	Invoice does not exist
`INVOICE_ALREADY_PAID`	409	Invoice has already been fully paid
`INVOICE_CANCELLED`	409	Cannot generate link for a cancelled invoice

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/generate-link \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await
fetch('https://api.zenpays.com/merchant/api/v1/invoices/inv_a1b2c3d4e5f6g7h8/generate-link', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.shortUrl); // "https://pay.zenpays.com/s/ZP-ABC123"
```

SDK

```
const link = await zenpays.invoices.generatePaymentLink('inv_a1b2c3d4e5f6g7h8');

console.log(link.shortUrl); // "https://pay.zenpays.com/s/ZP-ABC123"
```

Related Endpoints

- Get Invoice
- Send Invoice
- Pay Invoice
- Create Payment Link

Payment Links

REST API / BILLING / PAYMENT LINKS

Create Payment Link

Create a new payment link that can be shared with customers to collect payments.

`POST` **`/merchant/api/v1/payment-links`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Body Parameters

Parameter	Type	Required	Description
`title`	`string`	Yes	Payment link title (1-255 characters)
`description`	`string`	No	Payment link description
`amount`	`number`	Yes	Payment amount (minimum 0.01)
`currency`	`string`	No	Currency code (default: `INR`)
`usageType`	`string`	No	Usage type (default: `SINGLE`). One of `SINGLE`, `MULTI`
`maxUses`	`number`	No	Maximum number of uses (minimum 1, only for `MULTI` type)
`expiresAt`	`string`	No	Expiration date (ISO 8601)
`productId`	`string`	No	Associated product ID
`customerId`	`string`	No	Associated customer ID
`invoiceId`	`string`	No	Associated invoice ID
`invoiceIds`	`string[]`	No	Array of associated invoice IDs
`metadata`	`object`	No	Custom key-value pairs

Response

Success (201 Created)


```
{
  "success": true,
  "data": {
    "linkId": "pl_a1b2c3d4e5f6g7h8",
    "title": "Pro Plan Payment",
    "description": "Payment for Pro Plan subscription",
    "amount": 4999,
    "currency": "INR",
    "status": "ACTIVE",
    "usageType": "SINGLE",
    "maxUses": null,
    "usageCount": 0,
    "expiresAt": "2024-02-15T23:59:59.000Z",
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "customerId": null,
    "invoiceId": null,
    "paymentUrl": "https://pay.zenpays.com/pl_a1b2c3d4e5f6g7h8",
    "shortCode": "ZP-XYZ789",
    "shortUrl": "https://pay.zenpays.com/s/ZP-XYZ789",
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Payment link created successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 55,
    "api_version": "v1",
    "endpoint": "POST /payment-links",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`PRODUCT_NOT_FOUND`	404	Associated product does not exist
`INVOICE_NOT_FOUND`	404	Associated invoice does not exist

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/payment-links \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Pro Plan Payment",
    "description": "Payment for Pro Plan subscription",
    "amount": 4999,
    "currency": "INR",
    "usageType": "SINGLE",
    "expiresAt": "2024-02-15T23:59:59.000Z",
    "productId": "prod_a1b2c3d4e5f6g7h8"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-links', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    title: 'Pro Plan Payment',
    description: 'Payment for Pro Plan subscription',
    amount: 4999,
    currency: 'INR',
    usageType: 'SINGLE',
    expiresAt: '2024-02-15T23:59:59.000Z',
    productId: 'prod_a1b2c3d4e5f6g7h8',
  }),
});

const result = await response.json();
console.log(result.data.linkId); // pl_a1b2c3d4e5f6g7h8
console.log(result.data.shortUrl); // "https://pay.zenpays.com/s/ZP-XYZ789"
```

SDK

```
const link = await zenpays.paymentLinks.create({
  title: 'Pro Plan Payment',
  description: 'Payment for Pro Plan subscription',
  amount: 4999,
  currency: 'INR',
  usageType: 'SINGLE',
  expiresAt: '2024-02-15T23:59:59.000Z',
  productId: 'prod_a1b2c3d4e5f6g7h8',
});

console.log(link.linkId); // pl_a1b2c3d4e5f6g7h8
console.log(link.shortUrl); // "https://pay.zenpays.com/s/ZP-XYZ789"
```

Related Endpoints

- List Payment Links
- Get Payment Link
- Update Payment Link
- Payment Link Analytics

REST API / BILLING / PAYMENT LINKS

List Payment Links

Retrieve a paginated list of payment links with optional filtering and sorting.

`GET` **`/merchant/api/v1/payment-links`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Query Parameters

Parameter	Type	Description
`status`	`string`	Filter by status: `ACTIVE`, `INACTIVE`, `EXPIRED`
`currency`	`string`	Filter by currency code
`page`	`number`	Page number (default: `1`)
`limit`	`number`	Items per page (default: `20`, max: `100`)
`sortBy`	`string`	Sort field (default: `createdAt`)
`sortOrder`	`string`	Sort direction: `ASC` or `DESC` (default: `DESC`)
`searchTerm`	`string`	Search by title or description

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "items": [
      {
        "linkId": "pl_a1b2c3d4e5f6g7h8",
        "title": "Pro Plan Payment",
        "description": "Payment for Pro Plan subscription",
        "amount": 4999,
        "currency": "INR",
        "status": "ACTIVE",
        "usageType": "SINGLE",
        "maxUses": null,
        "usageCount": 0,
        "expiresAt": "2024-02-15T23:59:59.000Z",
        "shortCode": "ZP-XYZ789",
        "shortUrl": "https://pay.zenpays.com/s/ZP-XYZ789",
        "createdAt": "2024-01-15T10:30:00.000Z",
        "updatedAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "total": 1,
    "page": 1,
    "limit": 20,
    "totalPages": 1
  },
  "message": "Payment links retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 35,
    "api_version": "v1",
    "endpoint": "GET /payment-links",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid query parameters
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET "https://api.zenpays.com/merchant/api/v1/payment-links?status=ACTIVE&page=1&limit=20"
\
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const params = new URLSearchParams({
  status: 'ACTIVE',
  page: '1',
  limit: '20',
});

const response = await fetch(`https://api.zenpays.com/merchant/api/v1/payment-links?${params}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.items); // Array of payment links
console.log(result.data.total); // Total count
```

SDK

```
const links = await zenpays.paymentLinks.list({
  status: 'ACTIVE',
  page: 1,
  limit: 20,
});

console.log(links.items); // Array of payment links
console.log(links.total); // Total count
```

Related Endpoints

- Create Payment Link
- Get Payment Link
- Payment Link Analytics

REST API / BILLING / PAYMENT LINKS

Get Payment Link

Retrieve a single payment link by its ID.

`GET` **`/merchant/api/v1/payment-links/:linkId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`linkId`	`string`	Yes	The payment link ID (e.g. `pl_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "linkId": "pl_a1b2c3d4e5f6g7h8",
    "title": "Pro Plan Payment",
    "description": "Payment for Pro Plan subscription",
    "amount": 4999,
    "currency": "INR",
    "status": "ACTIVE",
    "usageType": "SINGLE",
    "maxUses": null,
    "usageCount": 0,
    "expiresAt": "2024-02-15T23:59:59.000Z",
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "customerId": null,
    "invoiceId": null,
    "paymentUrl": "https://pay.zenpays.com/pl_a1b2c3d4e5f6g7h8",
    "shortCode": "ZP-XYZ789",
    "shortUrl": "https://pay.zenpays.com/s/ZP-XYZ789",
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-15T10:30:00.000Z"
  },
  "message": "Payment link retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 25,
    "api_version": "v1",
    "endpoint": "GET /payment-links/:linkId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`PAYMENT_LINK_NOT_FOUND`	404	Payment link does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/payment-links/pl_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-
links/pl_a1b2c3d4e5f6g7h8', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.title); // "Pro Plan Payment"
console.log(result.data.shortUrl); // "https://pay.zenpays.com/s/ZP-XYZ789"
```

SDK

```
const link = await zenpays.paymentLinks.get('pl_a1b2c3d4e5f6g7h8');

console.log(link.title); // "Pro Plan Payment"
console.log(link.shortUrl); // "https://pay.zenpays.com/s/ZP-XYZ789"
```

Related Endpoints

- Create Payment Link
- Update Payment Link
- Deactivate Payment Link
- Linked Invoices

REST API / BILLING / PAYMENT LINKS

Update Payment Link

Update an existing payment link. Only provided fields will be updated.

`PUT` **`/merchant/api/v1/payment-links/:linkId`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`linkId`	`string`	Yes	The payment link ID (e.g. `pl_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Description
`title`	`string`	Payment link title (1-255 characters)
`description`	`string`	Payment link description
`amount`	`number`	Payment amount (minimum 0.01)
`currency`	`string`	Currency code
`usageType`	`string`	Usage type. One of `SINGLE`, `MULTI`
`status`	`string`	Link status. One of `ACTIVE`, `INACTIVE`, `EXPIRED`
`maxUses`	`number`	Maximum number of uses (minimum 1)
`expiresAt`	`string`	Expiration date (ISO 8601)
`invoiceIds`	`string[]`	Array of associated invoice IDs

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "linkId": "pl_a1b2c3d4e5f6g7h8",
    "title": "Pro Plan Payment - Updated",
    "description": "Payment for Pro Plan subscription (updated)",
    "amount": 3999,
    "currency": "INR",
    "status": "ACTIVE",
    "usageType": "MULTI",
    "maxUses": 10,
    "usageCount": 0,
    "expiresAt": "2024-03-15T23:59:59.000Z",
    "productId": "prod_a1b2c3d4e5f6g7h8",
    "customerId": null,
    "invoiceId": null,
    "paymentUrl": "https://pay.zenpays.com/pl_a1b2c3d4e5f6g7h8",
    "shortCode": "ZP-XYZ789",
    "shortUrl": "https://pay.zenpays.com/s/ZP-XYZ789",
    "metadata": {},
    "createdAt": "2024-01-15T10:30:00.000Z",
    "updatedAt": "2024-01-16T14:20:00.000Z"
  },
  "message": "Payment link updated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 40,
    "api_version": "v1",
    "endpoint": "PUT /payment-links/:linkId",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`VALIDATION_ERROR`	400	Invalid request parameters
`UNAUTHORIZED`	401	Invalid API key
`PAYMENT_LINK_NOT_FOUND`	404	Payment link does not exist

Examples

CURL

```
curl -X PUT https://api.zenpays.com/merchant/api/v1/payment-links/pl_a1b2c3d4e5f6g7h8 \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Pro Plan Payment - Updated",
    "amount": 3999,
    "usageType": "MULTI",
    "maxUses": 10,
    "expiresAt": "2024-03-15T23:59:59.000Z"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-
links/pl_a1b2c3d4e5f6g7h8', {
  method: 'PUT',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    title: 'Pro Plan Payment - Updated',
    amount: 3999,
    usageType: 'MULTI',
    maxUses: 10,
    expiresAt: '2024-03-15T23:59:59.000Z',
  }),
});

const result = await response.json();
console.log(result.data.amount); // 3999
console.log(result.data.usageType); // "MULTI"
```

SDK

```
const link = await zenpays.paymentLinks.update('pl_a1b2c3d4e5f6g7h8', {
  title: 'Pro Plan Payment - Updated',
  amount: 3999,
  usageType: 'MULTI',
  maxUses: 10,
  expiresAt: '2024-03-15T23:59:59.000Z',
});

console.log(link.amount); // 3999
console.log(link.usageType); // "MULTI"
```

Related Endpoints

- Get Payment Link
- Deactivate Payment Link

- List Payment Links

REST API / BILLING / PAYMENT LINKS

Deactivate Payment Link

Deactivate a payment link. The link status will be set to `INACTIVE` and it will no longer accept payments.

`POST` **`/merchant/api/v1/payment-links/:linkId/deactivate`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`linkId`	`string`	Yes	The payment link ID (e.g. `pl_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)

```
{
  "success": true,
  "data": {
    "linkId": "pl_a1b2c3d4e5f6g7h8",
    "status": "INACTIVE",
    "deactivatedAt": "2024-01-16T14:20:00.000Z"
  },
  "message": "Payment link deactivated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-16T14:20:00.000Z",
    "processing_time_ms": 30,
    "api_version": "v1",
    "endpoint": "POST /payment-links/:linkId/deactivate",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`PAYMENT_LINK_NOT_FOUND`	404	Payment link does not exist
`PAYMENT_LINK_ALREADY_INACTIVE`	409	Payment link is already inactive

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/payment-links/pl_a1b2c3d4e5f6g7h8/deactivate
\
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-16T14:20:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-
links/pl_a1b2c3d4e5f6g7h8/deactivate', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.status); // "INACTIVE"
```

SDK

```
const result = await zenpays.paymentLinks.deactivate('pl_a1b2c3d4e5f6g7h8');

console.log(result.status); // "INACTIVE"
```

Related Endpoints

- Get Payment Link
- Update Payment Link
- List Payment Links

REST API / BILLING / PAYMENT LINKS

Payment Link Analytics

Retrieve payment link analytics including totals, usage statistics, and counts by status.

`GET` **`/merchant/api/v1/payment-links/analytics`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "totalLinks": 75,
    "totalAmountCollected": 450000.00,
    "totalPayments": 120,
    "byStatus": {
      "ACTIVE": 40,
      "INACTIVE": 25,
      "EXPIRED": 10
    },
    "byUsageType": {
      "SINGLE": 50,
      "MULTI": 25
    },
    "averagePaymentAmount": 3750.00,
    "conversionRate": 68.5
  },
  "message": "Payment link analytics retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 90,
    "api_version": "v1",
    "endpoint": "GET /payment-links/analytics",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/payment-links/analytics \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-links/analytics', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data.totalLinks); // 75
console.log(result.data.totalAmountCollected); // 450000.00
```

SDK

```
const analytics = await zenpays.paymentLinks.analytics();

console.log(analytics.totalLinks); // 75
console.log(analytics.totalAmountCollected); // 450000.00
```

Related Endpoints

- List Payment Links
- Create Payment Link

REST API / BILLING / PAYMENT LINKS

Resolve Short Code

Resolve a short code to retrieve the associated payment link details.

> **Note**
> This is a **public endpoint** — no authentication headers are required.

`GET` **`/merchant/api/v1/payment-links/resolve/:shortCode`**

Request

Path Parameters

Parameter	Type	Required	Description
`shortCode`	`string`	Yes	The payment link short code (e.g. `ZP-XYZ789`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "linkId": "pl_a1b2c3d4e5f6g7h8",
    "title": "Pro Plan Payment",
    "description": "Payment for Pro Plan subscription",
    "amount": 4999,
    "currency": "INR",
    "status": "ACTIVE",
    "usageType": "SINGLE",
    "merchantName": "Acme Corp",
    "merchantLogo": "https://example.com/logo.png",
    "paymentUrl": "https://pay.zenpays.com/pl_a1b2c3d4e5f6g7h8",
    "expiresAt": "2024-02-15T23:59:59.000Z"
  },
  "message": "Payment link resolved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 20,
    "api_version": "v1",
    "endpoint": "GET /payment-links/resolve/:shortCode",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`PAYMENT_LINK_NOT_FOUND`	404	Short code does not resolve to any payment link
`PAYMENT_LINK_EXPIRED`	410	Payment link has expired
`PAYMENT_LINK_INACTIVE`	410	Payment link is inactive

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/payment-links/resolve/ZP-XYZ789
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-links/resolve/ZP-XYZ789', {
  method: 'GET',
});

const result = await response.json();
console.log(result.data.linkId); // pl_a1b2c3d4e5f6g7h8
console.log(result.data.paymentUrl); // "https://pay.zenpays.com/pl_a1b2c3d4e5f6g7h8"
```

SDK

```
const link = await zenpays.paymentLinks.resolveShortCode('ZP-XYZ789');

console.log(link.linkId); // pl_a1b2c3d4e5f6g7h8
console.log(link.paymentUrl); // "https://pay.zenpays.com/pl_a1b2c3d4e5f6g7h8"
```

Related Endpoints

- Pay Payment Link
- Get Payment Link

REST API / BILLING / PAYMENT LINKS

Pay Payment Link

Initiate payment for a payment link. Creates a payment intent that the customer can use to complete the payment.

> **Note**
> This is a **public endpoint** — no authentication headers are required.

`POST` **`/merchant/api/v1/payment-links/:linkId/pay`**

Request

Path Parameters

Parameter	Type	Required	Description
`linkId`	`string`	Yes	The payment link ID (e.g. `pl_a1b2c3d4e5f6g7h8`)

Body Parameters

Parameter	Type	Description
`customerEmail`	`string`	Customer email address

Response

Success (200 OK)


```
{
  "success": true,
  "data": {
    "linkId": "pl_a1b2c3d4e5f6g7h8",
    "paymentIntentId": "pi_x1y2z3w4v5u6t7s8",
    "amount": 4999,
    "currency": "INR",
    "checkoutUrl": "https://checkout.zenpays.com/pay/pi_x1y2z3w4v5u6t7s8"
  },
  "message": "Payment initiated successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 130,
    "api_version": "v1",
    "endpoint": "POST /payment-links/:linkId/pay",
    "user_type": "merchant"
  }
}
```

Error Responses

Code	HTTP	Message
`PAYMENT_LINK_NOT_FOUND`	404	Payment link does not exist
`PAYMENT_LINK_EXPIRED`	410	Payment link has expired
`PAYMENT_LINK_INACTIVE`	410	Payment link is inactive
`PAYMENT_LINK_MAX_USES_REACHED`	409	Payment link has reached its maximum number of uses

Examples

CURL

```
curl -X POST https://api.zenpays.com/merchant/api/v1/payment-links/pl_a1b2c3d4e5f6g7h8/pay \
  -H "Content-Type: application/json" \
  -d '{
    "customerEmail": "john@example.com"
  }'
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-
links/pl_a1b2c3d4e5f6g7h8/pay', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    customerEmail: 'john@example.com',
  }),
});

const result = await response.json();
console.log(result.data.checkoutUrl); // Redirect customer to this URL
```

SDK

```
const payment = await zenpays.paymentLinks.pay('pl_a1b2c3d4e5f6g7h8', {
  customerEmail: 'john@example.com',
});

console.log(payment.checkoutUrl); // Redirect customer to this URL
```

Related Endpoints

- Resolve Short Code
- Get Payment Link
- Create Payment Link

REST API / BILLING / PAYMENT LINKS

Linked Invoices

Retrieve all invoices linked to a specific payment link.

`GET` **`/merchant/api/v1/payment-links/:linkId/invoices`**

Request

Headers

Header	Required	Description
`Authorization`	Yes	`Bearer {api_key}`
`X-Signature`	Yes	HMAC-SHA256 signature
`X-Timestamp`	Yes	ISO 8601 timestamp
`X-Secret-Salt`	Yes	Secret salt for HMAC validation
`Content-Type`	Yes	`application/json`

Path Parameters

Parameter	Type	Required	Description
`linkId`	`string`	Yes	The payment link ID (e.g. `pl_a1b2c3d4e5f6g7h8`)

Response

Success (200 OK)


```
{
  "success": true,
  "data": [
    {
      "invoiceId": "inv_a1b2c3d4e5f6g7h8",
      "invoiceNumber": "INV-2024-0001",
      "customerId": "cust_a1b2c3d4e5f6g7h8",
      "customerEmail": "john@example.com",
      "customerName": "John Doe",
      "currency": "USD",
      "status": "SENT",
      "totalAmount": 275.00,
      "amountPaid": 0,
      "amountDue": 275.00,
      "issueDate": "2024-01-15T00:00:00.000Z",
      "dueDate": "2024-02-15T00:00:00.000Z",
      "createdAt": "2024-01-15T10:30:00.000Z"
    },
    {
      "invoiceId": "inv_c3d4e5f6g7h8i9j0",
      "invoiceNumber": "INV-2024-0002",
      "customerId": "cust_a1b2c3d4e5f6g7h8",
      "customerEmail": "john@example.com",
      "customerName": "John Doe",
      "currency": "USD",
      "status": "DRAFT",
      "totalAmount": 150.00,
      "amountPaid": 0,
      "amountDue": 150.00,
      "issueDate": "2024-01-20T00:00:00.000Z",
      "dueDate": "2024-02-20T00:00:00.000Z",
      "createdAt": "2024-01-20T10:30:00.000Z"
    }
  ],
  "message": "Linked invoices retrieved successfully",
  "error": null,
  "meta": {
    "request_id": "req_xxxxx",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "processing_time_ms": 30,
    "api_version": "v1",
    "endpoint": "GET /payment-links/:linkId/invoices",
    "user_type": "merchant"
  }
}
```


Error Responses

Code	HTTP	Message
`UNAUTHORIZED`	401	Invalid API key
`PAYMENT_LINK_NOT_FOUND`	404	Payment link does not exist

Examples

CURL

```
curl -X GET https://api.zenpays.com/merchant/api/v1/payment-links/pl_a1b2c3d4e5f6g7h8/invoices \
  -H "Authorization: Bearer zp_test_xxxxx" \
  -H "X-Timestamp: 2024-01-15T10:30:00.000Z" \
  -H "X-Signature: a1b2c3d4e5f6..." \
  -H "X-Secret-Salt: your_secret_salt" \
  -H "Content-Type: application/json"
```

JAVASCRIPT

```
const response = await fetch('https://api.zenpays.com/merchant/api/v1/payment-
links/pl_a1b2c3d4e5f6g7h8/invoices', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'X-Timestamp': timestamp,
    'X-Signature': signature,
    'X-Secret-Salt': secretSalt,
    'Content-Type': 'application/json',
  },
});

const result = await response.json();
console.log(result.data); // Array of linked invoices
console.log(result.data.length); // 2
```

SDK

```
const invoices = await zenpays.paymentLinks.listInvoices('pl_a1b2c3d4e5f6g7h8');

console.log(invoices); // Array of linked invoices
console.log(invoices.length); // 2
```

Related Endpoints

- Get Payment Link
- Create Payment Link
- Update Payment Link
- Get Invoice

PART

Examples

6 documents

EXAMPLES

Basic Payment

A complete example of creating and processing a payment.

JAVASCRIPT

```
import { PaymentError, ValidationError, ZenPays } from '@zenxdigitalholdings/zenpays'

async function processPayment() {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  try {
    // 1. Create a payment intent
    const intent = await zenpays.payments.createPaymentIntent({
      amount: 2999,
      currency: 'INR',
      paymentMethod: 'upi',
      customerEmail: 'john@example.com',
      customerFirstName: 'John',
      customerLastName: 'Doe',
      customerCountry: 'IN',
      description: 'Pro Plan - Monthly',
    })

    console.log('Payment Intent created:', intent.intentId)

    // 2. Confirm with payment details
    const result = await zenpays.payments.confirmPayment(intent.intentId, {
      customerDetails: {
        name: 'John Doe',
        email: 'john@example.com',
        address: { country: 'US' },
      },
      paymentMethodDetails: {
        type: 'upi',
        vpa: 'john@upi',
      },
    })

    // 3. Handle the result
    switch (result.status) {
      case 'succeeded':
        console.log('Payment successful!')
        console.log('Transaction ID:', result.externalTransactionId)
        return { success: true, transactionId: result.externalTransactionId }

      case 'processing':
        console.log('Payment is processing...')
        return { success: false, status: 'processing' }

      case 'failed':
        console.log('Payment failed:', result.message)
        return { success: false, error: result.message }

      default:
        // Handle next actions (3D Secure, etc.)
        if (result.nextAction) {
          return handleNextAction(result.nextAction)
        }
    }
  }
  catch (error) {
    if (error instanceof ValidationError) {
      console.error('Validation error:', error.message)
      return { success: false, error: 'Invalid payment details' }
    }
```

```
    if (error instanceof PaymentError) {
      console.error('Payment error:', error.message)
      return { success: false, error: 'Payment declined' }
    }

    throw error
  }
}

function handleNextAction(nextAction: any) {
  if (nextAction.type === 'redirect') {
    console.log('Redirect required:', nextAction.redirectUrl)
    return { success: false, redirect: nextAction.redirectUrl }
  }

  if (nextAction.type === 'three_ds') {
    console.log('3D Secure required')
    return { success: false, threeDsUrl: nextAction.threeDsUrl }
  }

  return { success: false, error: 'Unknown action required' }
}

// Run the example
processPayment()
  .then(result => console.log('Result:', result))
  .catch(console.error)
```

PYTHON

```python
import os
from zenpays import ZenPays
from zenpays.errors import PaymentError, ValidationError


def process_payment():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    try:
        # 1. Create a payment intent
        intent = zenpays.payments.create_payment_intent({
            "amount": 2999,
            "currency": "INR",
            "paymentMethod": "upi",
            "customerEmail": "john@example.com",
            "customerFirstName": "John",
            "customerLastName": "Doe",
            "customerCountry": "IN",
            "description": "Pro Plan - Monthly",
        })

        print(f"Payment Intent created: {intent['intent_id']}")

        # 2. Confirm with payment details
        result = zenpays.payments.confirm_payment(intent["intent_id"], {
            "customer_details": {
                "name": "John Doe",
                "email": "john@example.com",
                "address": {"country": "US"},
            },
            "payment_method_details": {
                "type": "upi",
                "vpa": "john@upi",
            },
        })

        # 3. Handle the result
        status = result["status"]

        if status == "succeeded":
            print("Payment successful!")
            print(f"Transaction ID: {result['external_transaction_id']}")
            return {"success": True, "transaction_id": result["external_transaction_id"]}

        elif status == "processing":
            print("Payment is processing...")
            return {"success": False, "status": "processing"}

        elif status == "failed":
            print(f"Payment failed: {result['message']}")
            return {"success": False, "error": result["message"]}

        else:
            # Handle next actions (3D Secure, etc.)
            if "next_action" in result:
                return handle_next_action(result["next_action"])

    except ValidationError as e:
        print(f"Validation error: {e}")
        return {"success": False, "error": "Invalid payment details"}

    except PaymentError as e:
```

```
        print(f"Payment error: {e}")
        return {"success": False, "error": "Payment declined"}


def handle_next_action(next_action):
    if next_action["type"] == "redirect":
        print(f"Redirect required: {next_action['redirect_url']}")
        return {"success": False, "redirect": next_action["redirect_url"]}

    if next_action["type"] == "three_ds":
        print("3D Secure required")
        return {"success": False, "three_ds_url": next_action["three_ds_url"]}

    return {"success": False, "error": "Unknown action required"}


# Run the example
if __name__ == "__main__":
    result = process_payment()
    print(f"Result: {result}")
```

EXAMPLES

Checkout Flow

Create a checkout session for on-ramp payments.

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

async function createCheckout() {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  // 1. Detect customer's location
  const geo = await zenpays.checkout.detectGeo()
  console.log('Detected country:', geo.country, 'Currency:', geo.currency)

  // 2. Create checkout session
  const session = await zenpays.checkout.createSession({
    playerId: 'player_123',
    amount: 100, // Amount in fiat currency
    targetSettlementAsset: 'USDT',
    successUrl: 'https://example.com/success',
    cancelUrl: 'https://example.com/cancel',
  })

  console.log('Session created:', session.id)

  // 3. Get available banks (if bank transfer)
  const { banks } = await zenpays.checkout.getBanks(geo.currency)
  console.log('Available banks:', banks.length)

  // 4. Select optimal provider
  const provider = await zenpays.checkout.selectProvider({
    sessionId: session.id,
    playerGeo: geo.country,
    currency: geo.currency,
    amount: 100,
    paymentMethod: 'card',
    targetSettlementAsset: 'USDT',
  })

  console.log('Selected provider:', provider.provider)
  console.log('Widget URL:', provider.widgetUrl)
  console.log('Embed mode:', provider.embedMode)

  // 5. Redirect or embed widget
  if (provider.embedMode === 'redirect') {
    // Redirect to provider
    window.location.href = provider.widgetUrl
  }
  else {
    // Embed in iframe
    const iframe = document.createElement('iframe')
    iframe.src = provider.widgetUrl
    iframe.style.width = '100%'
    iframe.style.height = '600px'
    document.body.appendChild(iframe)
  }

  // 6. Monitor order status
  const checkStatus = async () => {
    const order = await zenpays.checkout.getRampOrder(session.rampOrderId!)

    switch (order.status) {
      case 'pending':
        console.log('Waiting for payment...')
        break
```

```
      case 'pending_funds':
        console.log('Payment received, waiting for confirmation...')
        break
      case 'confirmed':
        console.log('Payment confirmed!')
        break
      case 'matched':
        console.log('Crypto matched!')
        break
      case 'cleared':
        console.log('Order complete!')
        return true
      case 'failed':
        console.log('Order failed')
        return false
    }

    // Check again in 5 seconds
    setTimeout(checkStatus, 5000)
  }

  checkStatus()
}

createCheckout()
```

PYTHON

```python
import os
import time
from zenpays import ZenPays


def create_checkout():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    # 1. Detect customer's location
    geo = zenpays.checkout.detect_geo()
    print(f"Detected country: {geo['country']}, Currency: {geo['currency']}")

    # 2. Create checkout session
    session = zenpays.checkout.create_session({
        "player_id": "player_123",
        "amount": 100,  # Amount in fiat currency
        "target_settlement_asset": "USDT",
        "success_url": "https://example.com/success",
        "cancel_url": "https://example.com/cancel",
    })

    print(f"Session created: {session['id']}")

    # 3. Get available banks (if bank transfer)
    result = zenpays.checkout.get_banks(geo["currency"])
    banks = result["banks"]
    print(f"Available banks: {len(banks)}")

    # 4. Select optimal provider
    provider = zenpays.checkout.select_provider({
        "session_id": session["id"],
        "player_geo": geo["country"],
        "currency": geo["currency"],
        "amount": 100,
        "payment_method": "card",
        "target_settlement_asset": "USDT",
    })

    print(f"Selected provider: {provider['provider']}")
    print(f"Widget URL: {provider['widget_url']}")
    print(f"Embed mode: {provider['embed_mode']}")

    # 5. Redirect or embed widget
    if provider["embed_mode"] == "redirect":
        # Redirect to provider
        print(f"Redirect to: {provider['widget_url']}")
        # In a web app: redirect user to provider["widget_url"]
    else:
        # Embed in iframe
        print("Embed widget in iframe")
        # In a web app: create iframe with provider["widget_url"]

    # 6. Monitor order status
    def check_status():
        order = zenpays.checkout.get_ramp_order(session["ramp_order_id"])

        status = order["status"]

        if status == "pending":
            print("Waiting for payment...")
        elif status == "pending_funds":
            print("Payment received, waiting for confirmation...")
```

```
        elif status == "confirmed":
            print("Payment confirmed!")
        elif status == "matched":
            print("Crypto matched!")
        elif status == "cleared":
            print("Order complete!")
            return True
        elif status == "failed":
            print("Order failed")
            return False

        # Check again in 5 seconds
        time.sleep(5)
        return check_status()

    check_status()

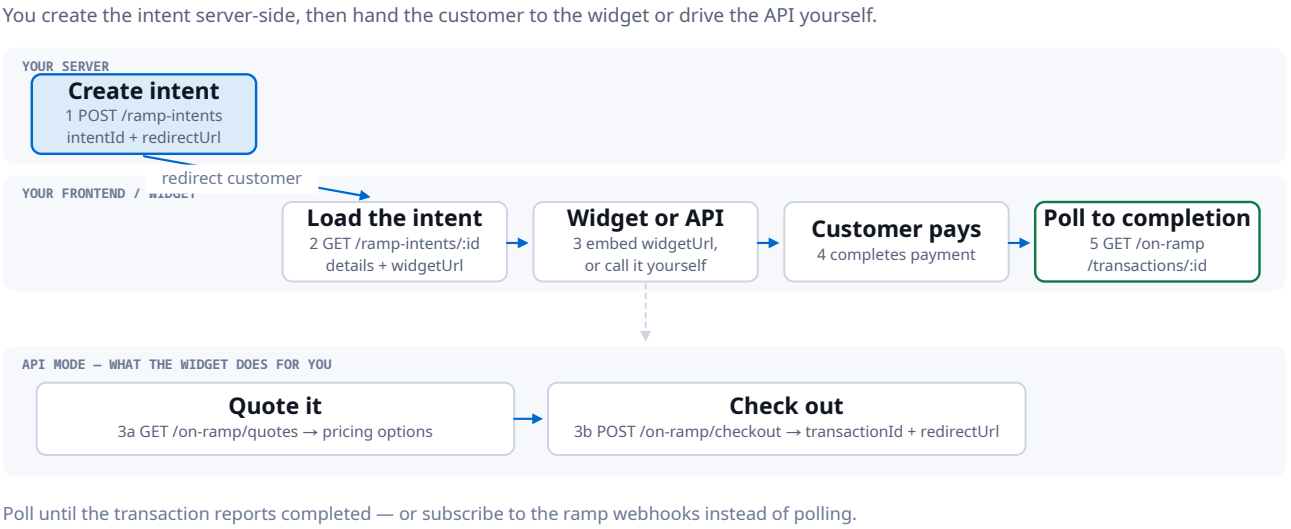

if __name__ == "__main__":
    create_checkout()
```

EXAMPLES

On-Ramp (Buy) Flow

This guide walks through the complete on-ramp integration — allowing your customers to buy crypto with fiat. The flow uses **ramp intents** to track the entire lifecycle from creation to completion.

Flow Overview



Buying crypto: intent on your server, purchase in your frontend

Step 1: Create a Ramp Intent (Server-Side)

Create a ramp intent from your backend. This requires your API key and merchant ID.

JAVASCRIPT

```
// Server-side — requires API key
const createBuyIntent = async (customerEmail, walletAddress) => {
  const response = await fetch('https://api.zenpays.com/payment/api/v1/ramp-intents', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${process.env.ZENPAYS_API_KEY}`,
      'x-merchant-id': process.env.ZENPAYS_MERCHANT_ID,
      'x-request-id': `req_${Date.now()}`,
    },
    body: JSON.stringify({
      type: 'buy',
      wallets: [{ network: 'ethereum', address: walletAddress }],
      fiatCurrency: 'USD',
      cryptoCurrency: 'eth_ethereum',
      amount: 100,
      customerEmail,
      successUrl: 'https://yourapp.com/buy/success',
      cancelUrl: 'https://yourapp.com/buy/cancel',
    }),
  });

  const { data } = await response.json();
  // data = { intentId, redirectUrl, expiresAt }
  return data;
};
```

PYTHON

```
import requests
import os
import time

def create_buy_intent(customer_email: str, wallet_address: str):
    response = requests.post(
        "https://api.zenpays.com/payment/api/v1/ramp-intents",
        headers={
            "Authorization": f"Bearer {os.environ['ZENPAYS_API_KEY']}",
            "x-merchant-id": os.environ["ZENPAYS_MERCHANT_ID"],
            "x-request-id": f"req_{int(time.time())}",
        },
        json={
            "type": "buy",
            "wallets": [{"network": "ethereum", "address": wallet_address}],
            "fiatCurrency": "USD",
            "cryptoCurrency": "eth_ethereum",
            "amount": 100,
            "customerEmail": customer_email,
            "successUrl": "https://yourapp.com/buy/success",
            "cancelUrl": "https://yourapp.com/buy/cancel",
        },
    )
    return response.json()["data"]
```

Step 2: Redirect Customer to Widget

Send the customer to the `redirectUrl` from Step 1, or fetch the intent on your frontend to get the widget URL for iframe embedding.

JAVASCRIPT

```
// Option A: Simple redirect
window.location.href = intentData.redirectUrl;

// Option B: Fetch intent and embed widget in iframe
const fetchAndEmbed = async (intentId) => {
  const response = await fetch(
    `https://api.zenpays.com/payment/api/v1/ramp-intents/${intentId}`
  );
  const { data } = await response.json();

  // Terminal lifecycle states
  if (data.intent.status === 'expired') {
    showError('This session has expired. Please start again.');
    return;
  }
  if (data.intent.status === 'cancelled') {
    // When KYC is declined the intent transitions to cancelled with
    // cancelReason='kyc_rejected'. See get-ramp-intent for details.
    showError(
      data.intent.cancelReason === 'kyc_rejected'
        ? 'Identity verification could not be completed.'
        : 'This session has been cancelled.'
    );
    return;
  }

  // KYC gate — surface a friendly state-specific message
  switch (data.intent.kycStatus) {
    case 'in_review':
      showInfo('Verification under review — we\'ll continue automatically once approved.');
      return;
    case 'pending':
      // Customer needs to start / finish verification. If `kycRequired` is
      // present in the response and `kycVerificationUrl` is set, embed it;
      // otherwise call initiate-kyc to create a session.
      break;
  }

  if (data.widgetUrl) {
    const iframe = document.createElement('iframe');
    iframe.src = data.widgetUrl;
    iframe.style.cssText = 'width:100%;height:600px;border:none;border-radius:12px;';
    iframe.allow = 'payment';
    document.getElementById('widget-container').appendChild(iframe);
  }
};
```

PYTHON

```python
# Server-side: return the redirect URL to your frontend
def get_intent_widget(intent_id: str):
    response = requests.get(
        f"https://api.zenpays.com/payment/api/v1/ramp-intents/{intent_id}"
    )
    data = response.json()["data"]

    if data["intent"]["status"] == "expired":
        raise Exception("Intent expired")

    return {
        "intent": data["intent"],
        "widgetUrl": data["widgetUrl"],
    }
```

Step 3: Get Quotes (API Mode)

If you're building a custom UI instead of using the widget, fetch buy quotes to show pricing options.

JAVASCRIPT

```javascript
const getQuotes = async (source, destination, amount) => {
  const params = new URLSearchParams({ source, destination, amount: String(amount) });

  const response = await fetch(
    `https://api.zenpays.com/payment/api/v1/on-ramp/quotes?${params}`
  );
  const { data } = await response.json();

  // data.quotes = array of quote options
  data.quotes.forEach((quote) => {
    console.log(`${quote.onramp}: ${quote.outputAmount} crypto`);
    console.log(`  Rate: ${quote.rate}, Fee: ${quote.totalFee}`);
    console.log(`  Payment: ${quote.paymentMethod}`);
    if (quote.recommended) console.log('  ☆ Recommended');
  });

  return data.quotes;
};

const quotes = await getQuotes('usd', 'eth_ethereum', 100);
```

PYTHON

```python
def get_quotes(source: str, destination: str, amount: float):
    response = requests.get(
        "https://api.zenpays.com/payment/api/v1/on-ramp/quotes",
        params={
            "source": source,
            "destination": destination,
            "amount": amount,
        },
    )
    quotes = response.json()["data"]["quotes"]

    for quote in quotes:
        print(f"{quote['onramp']}: {quote['outputAmount']} crypto")
        print(f"  Rate: {quote['rate']}, Fee: {quote['totalFee']}")
    return quotes
```

Step 4: Create Checkout

After the customer selects a quote, create a checkout to start the payment.

JAVASCRIPT

```javascript
const createCheckout = async (selectedQuote, walletAddress) => {
  const response = await fetch(
    'https://api.zenpays.com/payment/api/v1/on-ramp/checkout',
    {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        onramp: selectedQuote.onramp,
        source: 'usd',
        destination: 'eth_ethereum',
        amount: 100,
        type: 'buy',
        paymentMethod: selectedQuote.paymentMethod,
        walletAddress,
      }),
    }
  );

  const { data } = await response.json();
  // Redirect to payment page
  window.location.href = data.redirectUrl;
};
```

PYTHON

```python
def create_checkout(selected_quote, wallet_address: str):
    response = requests.post(
        "https://api.zenpays.com/payment/api/v1/on-ramp/checkout",
        json={
            "onramp": selected_quote["onramp"],
            "source": "usd",
            "destination": "eth_ethereum",
            "amount": 100,
            "type": "buy",
            "paymentMethod": selected_quote["paymentMethod"],
            "walletAddress": wallet_address,
        },
    )
    return response.json()["data"]
```

Step 5: Poll Transaction Status

After the customer completes payment, poll the transaction status until it reaches a terminal state.

JAVASCRIPT

```javascript
const pollTransaction = async (transactionId) => {
  const terminalStatuses = ['completed', 'failed', 'expired', 'refunded'];

  while (true) {
    const response = await fetch(
      `https://api.zenpays.com/payment/api/v1/on-ramp/transactions/${transactionId}`
    );
    const { data } = await response.json();

    console.log(`Status: ${data.status}`);

    if (terminalStatuses.includes(data.status)) {
      return data;
    }

    // Wait 5 seconds before next poll
    await new Promise((resolve) => setTimeout(resolve, 5000));
  }
};

const result = await pollTransaction('txn_abc123');

if (result.status === 'completed') {
  console.log(`Success! ${result.outAmount} crypto delivered`);
  console.log(`TX Hash: ${result.transactionHash}`);
} else {
  console.log(`Transaction ${result.status}`);
}
```

PYTHON

```python
import time

def poll_transaction(transaction_id: str):
    terminal = {"completed", "failed", "expired", "refunded"}

    while True:
        response = requests.get(
            f"https://api.zenpays.com/payment/api/v1/on-ramp/transactions/{transaction_id}"
        )
        data = response.json()["data"]
        print(f"Status: {data['status']}")

        if data["status"] in terminal:
            return data

        time.sleep(5)

result = poll_transaction("txn_abc123")
if result["status"] == "completed":
    print(f"Success! {result['outAmount']} crypto delivered")
```

Step 6: Receive Webhook Notification

Instead of polling (or in addition to it), you can receive a webhook when the ramp completes or fails. Configure your webhook URL in the merchant dashboard.

`ramp.completed` — crypto has been sent to the customer's wallet:

```
{
  "event_type": "ramp.completed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "buy",
    "status": "completed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "chain": "ethereum",
    "destinationAddress": "0x742d35Cc6634C0532925a3b844Bc9e7595f2bD28",
    "fireblocksTxId": "fb_tx_9a8b7c6d5e4f",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

`ramp.failed` — crypto send failed:

```
{
  "event_type": "ramp.failed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "buy",
    "status": "failed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "eth_ethereum",
    "amount": 100,
    "reason": "Crypto send failed",
    "failedAt": "2026-03-16T14:30:00.000Z"
  }
}
```


Webhook vs Polling

We recommend using webhooks as the primary notification mechanism and polling as a fallback. Webhooks are delivered as soon as the status changes — no delay. See the Webhooks guide for setup instructions, signature verification, and retry behavior.

Intent Status Lifecycle

```
created  →  active  →  completed
                    →  cancelled
                    →  expired (auto, after 1 hour)
```

Status	Description
`created`	Intent created, not yet accessed
`active`	Customer has accessed the intent (first GET)
`completed`	Transaction completed successfully
`cancelled`	Manually cancelled by merchant or customer
`expired`	Auto-expired after 1 hour

Error Handling

- **Intent expired**: Create a new intent and redirect the customer
- **Quote expired**: Fetch fresh quotes — prices change frequently
- **Payment failed**: Check the transaction status for details; the customer can retry
- **Network errors**: Implement retry with exponential backoff

Next Steps

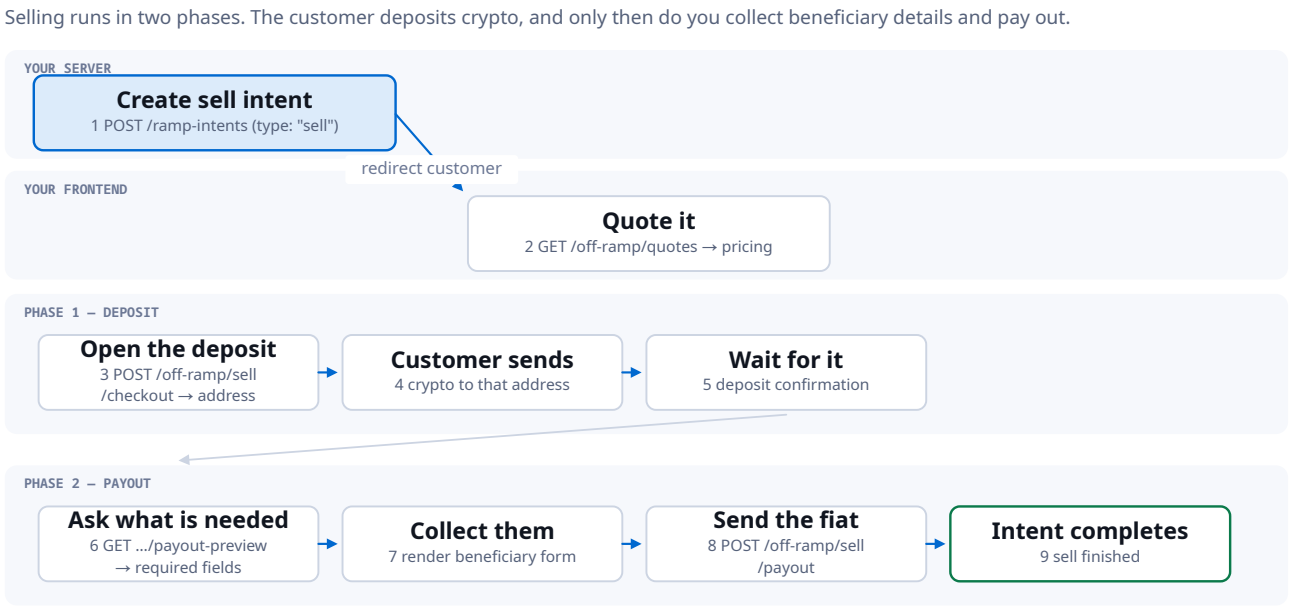
- Create Ramp Intent — API reference
- Buy Quotes — Quote endpoint details
- Buy Checkout — Checkout endpoint details
- Transaction Status — Status polling details
- Off-Ramp Flow — Sell crypto integration guide

EXAMPLES

Off-Ramp (Sell) Flow

This guide walks through the complete off-ramp integration — allowing your customers to sell crypto and receive fiat. The sell flow uses a **two-phase process**: Phase 1 generates a crypto deposit address, and Phase 2 submits beneficiary details for the fiat payout.

Flow Overview



Selling crypto: deposit first, then the fiat payout

Step 1: Create a Sell Intent (Server-Side)

Create a ramp intent with `type: "sell"` from your backend.

JAVASCRIPT

```
// Server-side — requires API key
const createSellIntent = async (customerEmail, walletAddress) => {
  const response = await fetch('https://api.zenpays.com/payment/api/v1/ramp-intents', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${process.env.ZENPAYS_API_KEY}`,
      'x-merchant-id': process.env.ZENPAYS_MERCHANT_ID,
      'x-request-id': `req_${Date.now()}`,
    },
    body: JSON.stringify({
      type: 'sell',
      wallets: [{ network: 'tron', address: walletAddress }],
      fiatCurrency: 'USD',
      cryptoCurrency: 'usdt_tron',
      customerEmail,
      successUrl: 'https://yourapp.com/sell/success',
      cancelUrl: 'https://yourapp.com/sell/cancel',
    }),
  });

  const { data } = await response.json();
  return data; // { intentId, redirectUrl, expiresAt }
};
```

PYTHON

```
import requests
import os
import time

def create_sell_intent(customer_email: str, wallet_address: str):
    response = requests.post(
        "https://api.zenpays.com/payment/api/v1/ramp-intents",
        headers={
            "Authorization": f"Bearer {os.environ['ZENPAYS_API_KEY']}",
            "x-merchant-id": os.environ["ZENPAYS_MERCHANT_ID"],
            "x-request-id": f"req_{int(time.time())}",
        },
        json={
            "type": "sell",
            "wallets": [{"network": "tron", "address": wallet_address}],
            "fiatCurrency": "USD",
            "cryptoCurrency": "usdt_tron",
            "customerEmail": customer_email,
            "successUrl": "https://yourapp.com/sell/success",
            "cancelUrl": "https://yourapp.com/sell/cancel",
        },
    )
    return response.json()["data"]
```

Step 2: Get Sell Quotes

Fetch quotes to show the customer how much fiat they'll receive for their crypto.

JAVASCRIPT

```
const getSellQuotes = async (cryptoAmount) => {
  const params = new URLSearchParams({
    source: 'usdt_tron',
    destination: 'USD',
    amount: String(cryptoAmount),
  });

  const response = await fetch(
    `https://api.zenpays.com/payment/api/v1/off-ramp/quotes?${params}`
  );
  const { data } = await response.json();

  data.quotes.forEach((quote) => {
    console.log(`Provider: ${quote.onramp}`);
    console.log(`  You send: ${quote.inputAmount} USDT`);
    console.log(`  You receive: ${quote.outputAmount} USD`);
    console.log(`  Rate: ${quote.rate}, Fee: ${quote.totalFee}`);
  });

  return data.quotes;
};

const quotes = await getSellQuotes(20);
const selectedQuote = quotes[0]; // Or let user choose
```

PYTHON

```
def get_sell_quotes(crypto_amount: float):
    response = requests.get(
        "https://api.zenpays.com/payment/api/v1/off-ramp/quotes",
        params={
            "source": "usdt_tron",
            "destination": "USD",
            "amount": crypto_amount,
        },
    )
    quotes = response.json()["data"]["quotes"]

    for quote in quotes:
        print(f"Provider: {quote['onramp']}")
        print(f"  Send: {quote['inputAmount']} USDT → Receive: {quote['outputAmount']} USD")
        print(f"  Rate: {quote['rate']}, Fee: {quote['totalFee']}")
    return quotes
```

Step 3: Phase 1 — Create Sell Deposit

Generate a deposit address for the customer to send crypto to.

JAVASCRIPT

```
const createDeposit = async (intentId, quote) => {
  const response = await fetch(
    'https://api.zenpays.com/payment/api/v1/off-ramp/sell/checkout',
    {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        intentId,
        cryptoCurrency: 'USDT',
        chain: 'tron',
        fiatCurrency: 'USD',
        cryptoAmount: quote.inputAmount,
        fiatAmount: quote.outputAmount,
        rate: quote.rate,
        totalFee: quote.totalFee,
      }),
    }
  );

  const { data } = await response.json();

  // Display to customer
  console.log(`Send ${data.expectedCryptoAmount} ${data.currency} to:`);
  console.log(`Address: ${data.depositAddress}`);
  console.log(`Network: ${data.chain}`);
  if (data.memo) console.log(`Memo: ${data.memo}`);

  // Save for Phase 2
  return {
    cryptoDepositId: data.cryptoDepositId,
    depositAddress: data.depositAddress,
    expectedAmount: data.expectedCryptoAmount,
  };
};
```

PYTHON

```python
def create_deposit(intent_id: str, quote: dict):
    response = requests.post(
        "https://api.zenpays.com/payment/api/v1/off-ramp/sell/checkout",
        json={
            "intentId": intent_id,
            "cryptoCurrency": "USDT",
            "chain": "tron",
            "fiatCurrency": "USD",
            "cryptoAmount": quote["inputAmount"],
            "fiatAmount": quote["outputAmount"],
            "rate": quote["rate"],
            "totalFee": quote["totalFee"],
        },
    )
    data = response.json()["data"]

    print(f"Send {data['expectedCryptoAmount']} {data['currency']} to:")
    print(f"Address: {data['depositAddress']}")
    print(f"Network: {data['chain']}")

    return {
        "cryptoDepositId": data["cryptoDepositId"],
        "depositAddress": data["depositAddress"],
    }
```

Display to Customer

Show the deposit address with a QR code and a countdown timer based on `expiresAt`. Remind the customer to send the exact `expectedCryptoAmount` on the correct chain.

Step 4: Wait for Deposit Confirmation

After the customer sends crypto, the system automatically detects and confirms the deposit. You can poll the intent status or listen for WebSocket events.

JAVASCRIPT

```javascript
const waitForDeposit = async (intentId) => {
  console.log('Waiting for crypto deposit...');

  while (true) {
    const response = await fetch(
      `https://api.zenpays.com/payment/api/v1/ramp-intents/${intentId}`
    );
    const { data } = await response.json();
    const phase = data.intent.metadata?.phase;

    if (phase === 'awaiting_payout' || phase === 'completed') {
      console.log('Deposit confirmed!');
      return data.intent;
    }

    if (data.intent.status === 'expired' || data.intent.status === 'cancelled') {
      throw new Error(`Intent ${data.intent.status}`);
    }

    // Poll every 10 seconds
    await new Promise((resolve) => setTimeout(resolve, 10000));
  }
};
```

PYTHON

```python
def wait_for_deposit(intent_id: str):
    print("Waiting for crypto deposit...")

    while True:
        response = requests.get(
            f"https://api.zenpays.com/payment/api/v1/ramp-intents/{intent_id}"
        )
        data = response.json()["data"]
        phase = (data["intent"].get("metadata") or {}).get("phase")

        if phase in ("awaiting_payout", "completed"):
            print("Deposit confirmed!")
            return data["intent"]

        if data["intent"]["status"] in ("expired", "cancelled"):
            raise Exception(f"Intent {data['intent']['status']}")

        time.sleep(10)
```

Step 5: Phase 2a — Get Payout Preview

Once the deposit is confirmed, fetch the required beneficiary fields for the fiat payout.

JAVASCRIPT

```javascript
const getPayoutPreview = async (intentId, fiatCurrency, country = 'US') => {
  const params = new URLSearchParams({ intentId, fiatCurrency, country });

  const response = await fetch(
    `https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout-preview?${params}`
  );
  const { data } = await response.json();

  console.log(`Provider: ${data.tspDisplayName}`);
  console.log(`Payout: ${data.fiatAmount} ${data.fiatCurrency}`);
  console.log(`Fees: ${data.fees.total}`);
  console.log(`Required fields: ${data.requiredFields.length}`);

  return data;
};

const preview = await getPayoutPreview(intentId, 'USD', 'US');

// Dynamically render form from requiredFields
preview.requiredFields.forEach((field) => {
  // Create form inputs based on field.type, field.label, field.required, etc.
  console.log(`  ${field.required ? '*' : ' '} ${field.label} (${field.fieldName})`);
});
```

PYTHON

```python
def get_payout_preview(intent_id: str, fiat_currency: str, country: str = "US"):
    response = requests.get(
        "https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout-preview",
        params={
            "intentId": intent_id,
            "fiatCurrency": fiat_currency,
            "country": country,
        },
    )
    data = response.json()["data"]

    print(f"Provider: {data['tspDisplayName']}")
    print(f"Payout: {data['fiatAmount']} {data['fiatCurrency']}")
    print(f"Fees: {data['fees']['total']}")
    print(f"\nRequired fields:")
    for field in data["requiredFields"]:
        print(f"  {'*' if field['required'] else ' '} {field['label']} ({field['fieldName']})")
    return data
```

Step 6: Phase 2b — Submit Payout

Collect the beneficiary details from the customer and submit the payout.

JAVASCRIPT

```javascript
const submitPayout = async (intentId, cryptoDepositId, beneficiaryDetails) => {
  const response = await fetch(
    'https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout',
    {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        intentId,
        cryptoDepositId,
        fiatCurrency: 'USD',
        country: 'US',
        beneficiaryDetails,
      }),
    }
  );

  const { data } = await response.json();

  console.log(`Payout ID: ${data.payoutId}`);
  console.log(`Status: ${data.status}`);
  console.log(`Amount: ${data.amount} ${data.currency}`);

  return data;
};

// Example: customer filled in the form
const payout = await submitPayout(intentId, deposit.cryptoDepositId, {
  receiverFirstName: 'John',
  receiverLastName: 'Doe',
  receiverCountry: 'US',
  receiverAccountNumber: '123456789',
  receiverBankName: 'Bank of America',
  receiverBankCode: 'BOFAUS3N',
  receiverAddressLine1: '123 Main St',
  receiverCity: 'New York',
  receiverState: 'NY',
  receiverPinCode: '10001',
  remittancePurpose: 'PAYP001 - Family Support',
  sourceOfFund: 'PAYF001 - Salary',
  relationship: 'PAYR001 - Self',
});

// Intent is now completed — redirect customer
window.location.href = '/sell/success';
```

PYTHON

```python
def submit_payout(intent_id: str, crypto_deposit_id: str, beneficiary_details: dict):
    response = requests.post(
        "https://api.zenpays.com/payment/api/v1/off-ramp/sell/payout",
        json={
            "intentId": intent_id,
            "cryptoDepositId": crypto_deposit_id,
            "fiatCurrency": "USD",
            "country": "US",
            "beneficiaryDetails": beneficiary_details,
        },
    )
    data = response.json()["data"]
    print(f"Payout ID: {data['payoutId']}")
    print(f"Status: {data['status']}")
    print(f"Amount: {data['amount']} {data['currency']}")
    return data

# Example
payout = submit_payout(intent_id, deposit["cryptoDepositId"], {
    "receiverFirstName": "John",
    "receiverLastName": "Doe",
    "receiverCountry": "US",
    "receiverAccountNumber": "123456789",
    "receiverBankName": "Bank of America",
    "receiverBankCode": "BOFAUS3N",
    "receiverAddressLine1": "123 Main St",
    "receiverCity": "New York",
    "receiverState": "NY",
    "receiverPinCode": "10001",
    "remittancePurpose": "PAYP001 - Family Support",
    "sourceOfFund": "PAYF001 - Salary",
    "relationship": "PAYR001 - Self",
})
```

Step 7: Receive Webhook Notification

After the payout is submitted (or if it fails), ZenPay delivers a webhook to your configured endpoint.

`ramp.completed` — fiat payout has been submitted successfully:

```
{
  "event_type": "ramp.completed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "sell",
    "status": "completed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "usdt_tron",
    "amount": 20,
    "payoutId": "po_xyz789",
    "payoutAmount": 19.50,
    "payoutCurrency": "USD",
    "completedAt": "2026-03-16T14:30:00.000Z"
  }
}
```

`ramp.failed` — payout submission failed:

```
{
  "event_type": "ramp.failed",
  "payment_data": {
    "merchant_id": "m_abc123",
    "intentId": "ri_1710345678000_a1b2c3d4e5f6g7h8",
    "type": "sell",
    "status": "failed",
    "fiatCurrency": "USD",
    "cryptoCurrency": "usdt_tron",
    "amount": 20,
    "reason": "Payout submission failed",
    "failedAt": "2026-03-16T14:30:00.000Z"
  }
}
```


Webhook vs Polling

We recommend using webhooks as the primary notification mechanism. Webhooks are delivered as soon as the ramp completes or fails — no delay. See the Webhooks guide for setup, signature verification, and retry behavior.

Complete Example

JAVASCRIPT

```
// Full off-ramp flow
const runSellFlow = async () => {
  // Step 1: Create intent (server-side)
  const intent = await createSellIntent('customer@example.com', 'TLfMk...');
  console.log(`Intent: ${intent.intentId}`);

  // Step 2: Get quotes
  const quotes = await getSellQuotes(20);
  const selectedQuote = quotes[0];

  // Step 3: Phase 1 — create deposit
  const deposit = await createDeposit(intent.intentId, selectedQuote);
  console.log(`Send USDT to: ${deposit.depositAddress}`);

  // Step 4: Wait for deposit confirmation
  await waitForDeposit(intent.intentId);

  // Step 5: Phase 2a — get payout preview
  const preview = await getPayoutPreview(intent.intentId, 'USD', 'US');

  // Step 6: Phase 2b — submit payout (with customer's details)
  const payout = await submitPayout(intent.intentId, deposit.cryptoDepositId, {
    receiverFirstName: 'John',
    receiverLastName: 'Doe',
    receiverCountry: 'US',
    receiverAccountNumber: '123456789',
    receiverBankName: 'Bank of America',
    // ... other fields from preview.requiredFields
  });

  console.log(`Done! Payout ${payout.payoutId} is ${payout.status}`);
};
```

PYTHON

```python
def run_sell_flow():
    # Step 1: Create intent
    intent = create_sell_intent("customer@example.com", "TLfMk...")
    print(f"Intent: {intent['intentId']}")

    # Step 2: Get quotes
    quotes = get_sell_quotes(20)
    selected_quote = quotes[0]

    # Step 3: Phase 1 — create deposit
    deposit = create_deposit(intent["intentId"], selected_quote)

    # Step 4: Wait for deposit confirmation
    wait_for_deposit(intent["intentId"])

    # Step 5: Phase 2a — get payout preview
    preview = get_payout_preview(intent["intentId"], "USD", "US")

    # Step 6: Phase 2b — submit payout
    payout = submit_payout(intent["intentId"], deposit["cryptoDepositId"], {
        "receiverFirstName": "John",
        "receiverLastName": "Doe",
        "receiverCountry": "US",
        "receiverAccountNumber": "123456789",
        "receiverBankName": "Bank of America",
    })

    print(f"Done! Payout {payout['payoutId']} is {payout['status']}")

run_sell_flow()
```

Error Handling

Phase	Error	Recovery
Intent creation	Validation error	Check required fields and retry
Deposit (Phase 1)	Intent expired	Create a new intent
Deposit (Phase 1)	Deposit address generation failed	Retry — endpoint is idempotent
Waiting	Deposit not detected	Ensure correct chain and address; check with block explorer
Payout preview	Deposit not confirmed	Wait longer for blockchain confirmation
Payout (Phase 2)	FX quotation failed	Retry — the system will obtain a new quote
Payout (Phase 2)	Insufficient amount (fees > value)	Customer sent too little crypto

Next Steps

- Create Ramp Intent — API reference
- Sell Quotes — Quote endpoint details
- Create Sell Deposit — Phase 1 reference
- Get Payout Preview — Phase 2a reference

- Submit Sell Payout — Phase 2b reference
- On-Ramp Flow — Buy crypto integration guide

EXAMPLES

Payout Flow

Send money to customers and vendors using the two-step intent flow.

Payout Intent Flow (Recommended)

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

async function payoutWithIntent() {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  // Step 1: Create payout intent with basic details
  const intent = await zenpays.payouts.createPayoutIntent({
    amount: 10000,
    currency: 'INR',
    country: 'IN',
    beneficiaryName: 'John Doe',
    beneficiaryEmail: 'john@example.com',
    purpose: 'SALARY',
  })

  console.log('Intent created:', intent.intentId)
  console.log('Status:', intent.status) // 'requires_confirmation'
  console.log('Expires at:', intent.expiresAt)

  // The response tells you exactly what fields to collect
  console.log('Required fields:')
  intent.requiredFields?.forEach(field => {
    console.log(`  - ${field.label} (${field.fieldName}) [${field.type}]`)
  })

  // Step 2: Confirm with the required fields
  const result = await zenpays.payouts.confirmPayoutIntent(intent.intentId, {
    beneficiaryAccount: '1234567890',
    beneficiaryIfsc: 'HDFC0001234',
  })

  console.log('Payout confirmed:', result.payoutId)
  console.log('Status:', result.status) // 'processing'

  // Step 3: Poll for completion
  const poll = async (id: string): Promise<void> => {
    const current = await zenpays.payouts.getPayoutIntent(id)

    if (current.status === 'succeeded') {
      console.log('Payout completed!')
      return
    }
    if (current.status === 'failed') {
      console.log('Payout failed:', current.failureReason)
      return
    }

    console.log('Status:', current.status)
    await new Promise(r => setTimeout(r, 5000))
    return poll(id)
  }

  await poll(intent.intentId)
}

// Crypto payout example
async function cryptoPayout() {
  const zenpays = new ZenPays({
```

```
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  // Step 1: Create intent for crypto (USDT)
  const intent = await zenpays.payouts.createPayoutIntent({
    amount: 500,
    currency: 'USDT',
    beneficiaryName: 'Crypto Wallet',
  })

  // For crypto, requiredFields will include "chain" and "walletAddress"
  console.log(intent.requiredFields)
  // [{ fieldName: "chain", type: "select", options: ["ethereum", "tron", ...] },
  //  { fieldName: "walletAddress", type: "text" }]

  // Step 2: Confirm with chain and address
  const result = await zenpays.payouts.confirmPayoutIntent(intent.intentId, {
    chain: 'tron',
    walletAddress: 'TXyz123abc...',
  })

  console.log('Crypto payout processing:', result.payoutId)
}

payoutWithIntent()
```

PYTHON

```python
import os
import time
from zenpays import ZenPays


def payout_with_intent():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    # Step 1: Create payout intent
    intent = zenpays.payouts.create_payout_intent({
        "amount": 10000,
        "currency": "INR",
        "country": "IN",
        "beneficiary_name": "John Doe",
        "beneficiary_email": "john@example.com",
        "purpose": "SALARY",
    })

    print(f"Intent created: {intent['intent_id']}")
    print(f"Status: {intent['status']}")  # "requires_confirmation"

    # The response tells you what fields to collect
    for field in intent["required_fields"]:
        print(f"  - {field['label']} ({field['field_name']})")

    # Step 2: Confirm with the required fields
    result = zenpays.payouts.confirm_payout_intent(intent["intent_id"], {
        "beneficiary_account": "1234567890",
        "beneficiary_ifsc": "HDFC0001234",
    })

    print(f"Payout confirmed: {result['payout_id']}")
    print(f"Status: {result['status']}")  # "processing"

    # Step 3: Poll for completion
    def poll(intent_id):
        current = zenpays.payouts.get_payout_intent(intent_id)
        if current["status"] == "succeeded":
            print("Payout completed!")
            return
        if current["status"] == "failed":
            print(f"Payout failed: {current.get('failure_reason')}")
            return
        print(f"Status: {current['status']}")
        time.sleep(5)
        poll(intent_id)

    poll(intent["intent_id"])


def crypto_payout():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    # Step 1: Crypto intent
    intent = zenpays.payouts.create_payout_intent({
        "amount": 500,
        "currency": "USDT",
        "beneficiary_name": "Crypto Wallet",
    })

    # Step 2: Confirm with chain + address
    result = zenpays.payouts.confirm_payout_intent(intent["intent_id"], {
```

```
        "chain": "tron",
        "wallet_address": "TXyz123abc...",
    })

    print(f"Crypto payout processing: {result['payout_id']}")


if __name__ == "__main__":
    payout_with_intent()
```

Legacy Direct Payout

> **Deprecated**
> The direct payout API is deprecated. Use the Payout Intent flow above instead.

JAVASCRIPT

```
import { ValidationError, ZenPays } from '@zenxdigitalholdings/zenpays'

async function processPayout() {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  // 1. Check available payout methods
  const methods = await zenpays.payouts.getMethods()
  console.log('Available payout methods:', methods.map(m => m.type).join(', '))

  // 2. Check wallet balance
  const balance = await zenpays.wallet.getBalance('USD')
  console.log('Available balance:', balance)

  // 3. Create payout
  const payout = await zenpays.payouts.create({
    customerId: 'cust_xxx',
    amount: 10000, // $100.00
    currency: 'USD',
    payoutMethod: 'bank_transfer',
    beneficiaryDetails: {
      name: 'John Doe',
      email: 'john@example.com',
      bankName: 'Chase Bank',
      accountNumber: '123456789',
      accountType: 'checking',
      routingNumber: '021000021',
    },
    description: 'Withdrawal request',
    idempotencyKey: 'payout_unique_123', // Prevent duplicates
  })

  console.log('Payout created:', payout.id)
  console.log('Status:', payout.status)

  // 4. Monitor payout status
  const waitForPayout = async (payoutId: string): Promise<boolean> => {
    const p = await zenpays.payouts.get(payoutId)

    switch (p.status) {
      case 'completed':
        console.log('Payout completed!')
        console.log('Processed at:', p.processedAt)
        return true

      case 'failed':
        console.log('Payout failed:', p.failureReason)
        return false

      case 'processing':
        console.log('Payout processing...')
        break

      default:
        console.log('Payout status:', p.status)
    }

    await new Promise(resolve => setTimeout(resolve, 5000))
    return waitForPayout(payoutId)
  }
```

```
  return waitForPayout(payout.id)
}

// Retry a failed payout
async function retryFailedPayout(payoutId: string) {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  const payout = await zenpays.payouts.get(payoutId)

  if (payout.status !== 'failed') {
    console.log('Payout is not failed, cannot retry')
    return
  }

  console.log('Retrying payout:', payoutId)
  const retried = await zenpays.payouts.retry(payoutId)
  console.log('Retry status:', retried.status)
}

// Get customer payouts
async function getCustomerPayouts(customerId: string) {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  const { data, total } = await zenpays.payouts.listByCustomer(customerId, {
    limit: 10,
  })

  console.log(`Found ${total} payouts for customer ${customerId}:`)
  data.forEach((p) => {
    console.log(`  ${p.id}: ${p.amount} ${p.currency} - ${p.status}`)
  })
}

processPayout()
```

PYTHON

```python
import os
import time
from zenpays import ZenPays
from zenpays.errors import ValidationError


def process_payout():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    # 1. Check available payout methods
    methods = zenpays.payouts.get_methods()
    print(f"Available payout methods: {', '.join(m['type'] for m in methods)}")

    # 2. Check wallet balance
    balance = zenpays.wallet.get_balance("USD")
    print(f"Available balance: {balance}")

    # 3. Create payout
    payout = zenpays.payouts.create({
        "customer_id": "cust_xxx",
        "amount": 10000,  # $100.00
        "currency": "USD",
        "payout_method": "bank_transfer",
        "beneficiary_details": {
            "name": "John Doe",
            "email": "john@example.com",
            "bank_name": "Chase Bank",
            "account_number": "123456789",
            "account_type": "checking",
            "routing_number": "021000021",
        },
        "description": "Withdrawal request",
        "idempotency_key": "payout_unique_123",  # Prevent duplicates
    })

    print(f"Payout created: {payout['id']}")
    print(f"Status: {payout['status']}")

    # 4. Monitor payout status
    def wait_for_payout(payout_id: str) -> bool:
        p = zenpays.payouts.get(payout_id)

        status = p["status"]

        if status == "completed":
            print("Payout completed!")
            print(f"Processed at: {p.get('processed_at')}")
            return True

        elif status == "failed":
            print(f"Payout failed: {p.get('failure_reason')}")
            return False

        elif status == "processing":
            print("Payout processing...")

        else:
            print(f"Payout status: {status}")

        time.sleep(5)
        return wait_for_payout(payout_id)
```

```
    return wait_for_payout(payout["id"])


# Retry a failed payout
def retry_failed_payout(payout_id: str):
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    payout = zenpays.payouts.get(payout_id)

    if payout["status"] != "failed":
        print("Payout is not failed, cannot retry")
        return

    print(f"Retrying payout: {payout_id}")
    retried = zenpays.payouts.retry(payout_id)
    print(f"Retry status: {retried['status']}")


# Get customer payouts
def get_customer_payouts(customer_id: str):
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    result = zenpays.payouts.list_by_customer(customer_id, {"limit": 10})
    data = result["data"]
    total = result["total"]

    print(f"Found {total} payouts for customer {customer_id}:")
    for p in data:
        print(f"  {p['id']}: {p['amount']} {p['currency']} - {p['status']}")


if __name__ == "__main__":
    process_payout()
```

Cancel a Payout

Cancel a payout that hasn't started processing yet.

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

async function cancelPayout(payoutId: string) {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  // Check current status
  const payout = await zenpays.payouts.get(payoutId)

  if (payout.status !== 'pending') {
    console.log(`Cannot cancel — payout is already ${payout.status}`)
    return
  }

  // Cancel the payout
  const cancelled = await zenpays.payouts.cancel(payoutId, 'Customer requested cancellation')
  console.log('Cancelled:', cancelled.status) // 'cancelled'
}

cancelPayout('pay_xxx')
```

PYTHON

```python
import os
from zenpays import ZenPays


def cancel_payout(payout_id: str):
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    # Check current status
    payout = zenpays.payouts.get(payout_id)

    if payout["status"] != "pending":
        print(f"Cannot cancel — payout is already {payout['status']}")
        return

    # Cancel the payout
    cancelled = zenpays.payouts.cancel(payout_id, "Customer requested cancellation")
    print(f"Cancelled: {cancelled['status']}")  # "cancelled"


if __name__ == "__main__":
    cancel_payout("pay_xxx")
```

Batch Payout Flow

Send payouts to multiple beneficiaries in a single batch.

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

async function processBatchPayout() {
  const zenpays = new ZenPays({
    apiKey: process.env.ZENPAYS_API_KEY!,
  })

  // 1. Create batch payout
  const batch = await zenpays.payouts.createBatch({
    items: [
      {
        beneficiaryName: 'Alice Smith',
        beneficiaryAccount: '1234567890',
        beneficiaryIfsc: 'HDFC0001234',
        amount: 5000,
        currency: 'INR',
        payoutType: 'imps',
        purpose: 'SALARY',
      },
      {
        beneficiaryName: 'Bob Johnson',
        beneficiaryVpa: 'bob@upi',
        amount: 3000,
        currency: 'INR',
        payoutType: 'upi',
        purpose: 'COMMISSION',
      },
      {
        beneficiaryName: 'Carol Williams',
        beneficiaryAccount: '9876543210',
        beneficiaryIfsc: 'ICIC0004567',
        amount: 7000,
        currency: 'INR',
        payoutType: 'neft',
        purpose: 'PAYOUT',
      },
    ],
    currency: 'INR',
    webhookUrl: 'https://example.com/webhooks/batch',
  })

  console.log('Batch created:', batch.batchId)
  console.log('Total amount:', batch.totalAmount, batch.currency)
  console.log('Estimated fees:', batch.estimatedFees)
  console.log('Status:', batch.status) // 'validated'

  // 2. Confirm the batch to start processing
  const confirmed = await zenpays.payouts.confirmBatch(batch.batchId)
  console.log('Processing started:', confirmed.status) // 'processing'

  // 3. Poll for completion
  const waitForBatch = async (batchId: string): Promise<void> => {
    const status = await zenpays.payouts.getBatch(batchId)
    console.log(`Progress: ${status.progressPercentage}% (${status.successfulPayouts} succeeded,
${status.failedPayouts} failed)`)

    if (status.status === 'completed' || status.status === 'partially_completed') {
      console.log('Batch finished!')

      // 4. Check individual results
      const { data: payouts } = await zenpays.payouts.getBatchPayouts(batchId)
      payouts.forEach(p => {
```

```
        console.log(`  ${p.id}: ${p.amount} ${p.currency} → ${p.status}`)
      })

      // 5. Check for failures
      const { data: failed } = await zenpays.payouts.getBatchPayouts(batchId, { status: 'failed'
})
      if (failed.length > 0) {
        console.log(`${failed.length} payouts failed:`)
        failed.forEach(p => console.log(`  ${p.id}: ${p.failureReason}`))
      }
      return
    }

    await new Promise(resolve => setTimeout(resolve, 5000))
    return waitForBatch(batchId)
  }

  await waitForBatch(batch.batchId)
}

processBatchPayout()
```

PYTHON

```python
import os
import time
from zenpays import ZenPays


def process_batch_payout():
    zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])

    # 1. Create batch payout
    batch = zenpays.payouts.create_batch({
        "items": [
            {
                "beneficiary_name": "Alice Smith",
                "beneficiary_account": "1234567890",
                "beneficiary_ifsc": "HDFC0001234",
                "amount": 5000,
                "currency": "INR",
                "payout_type": "imps",
                "purpose": "SALARY",
            },
            {
                "beneficiary_name": "Bob Johnson",
                "beneficiary_vpa": "bob@upi",
                "amount": 3000,
                "currency": "INR",
                "payout_type": "upi",
                "purpose": "COMMISSION",
            },
            {
                "beneficiary_name": "Carol Williams",
                "beneficiary_account": "9876543210",
                "beneficiary_ifsc": "ICIC0004567",
                "amount": 7000,
                "currency": "INR",
                "payout_type": "neft",
                "purpose": "PAYOUT",
            },
        ],
        "currency": "INR",
        "webhook_url": "https://example.com/webhooks/batch",
    })

    print(f"Batch created: {batch['batch_id']}")
    print(f"Total amount: {batch['total_amount']} {batch['currency']}")
    print(f"Status: {batch['status']}")  # "validated"

    # 2. Confirm the batch to start processing
    confirmed = zenpays.payouts.confirm_batch(batch["batch_id"])
    print(f"Processing started: {confirmed['status']}")  # "processing"

    # 3. Poll for completion
    def wait_for_batch(batch_id: str):
        status = zenpays.payouts.get_batch(batch_id)
        print(f"Progress: {status['progress_percentage']}% "
              f"({status['successful_payouts']} succeeded, {status['failed_payouts']} failed)")

        if status["status"] in ("completed", "partially_completed"):
            print("Batch finished!")

            # 4. Check individual results
            result = zenpays.payouts.get_batch_payouts(batch_id)
            for p in result["data"]:
```

```
                print(f"  {p['id']}: {p['amount']} {p['currency']} -> {p['status']}")

            # 5. Check for failures
            failed = zenpays.payouts.get_batch_payouts(batch_id, {"status": "failed"})
            if failed["data"]:
                print(f"{len(failed['data'])} payouts failed:")
                for p in failed["data"]:
                    print(f"  {p['id']}: {p.get('failure_reason')}")
            return

        time.sleep(5)
        wait_for_batch(batch_id)

    wait_for_batch(batch["batch_id"])


if __name__ == "__main__":
    process_batch_payout()
```

EXAMPLES

Refund Flow

Process INR refunds -- full refunds, partial refunds, and batch operations. Refunds are paid out to the customer's bank account via IMPS.

JAVASCRIPT

```
import { ZenPays } from '@zenxdigitalholdings/zenpays'

const zenpays = new ZenPays({ apiKey: process.env.ZENPAYS_API_KEY! })

// ---------------------------------------------------------------------------
// 1. Full refund
// ---------------------------------------------------------------------------
async function fullRefund(transactionId: string) {
  const refund = await zenpays.refunds.create({
    transactionId,
    reason: 'Customer requested full refund',
    beneficiaryEmail: 'priya@example.com',
    beneficiaryAccount: '50100012345678',
    beneficiaryIfsc: 'HDFC0001234',
  })

  console.log('Refund created:', refund.refundId)
  console.log('Amount:', refund.amount, refund.currency)
  console.log('Status:', refund.status) // pending_approval
}

// ---------------------------------------------------------------------------
// 2. Partial refund
// ---------------------------------------------------------------------------
async function partialRefund(transactionId: string) {
  const refund = await zenpays.refunds.create({
    transactionId,
    amount: 500, // Refund 500 INR out of the total
    reason: 'Partial item return',
    beneficiaryEmail: 'priya@example.com',
    beneficiaryAccount: '50100012345678',
    beneficiaryIfsc: 'HDFC0001234',
  })

  console.log('Partial refund:', refund.refundId)
  console.log('Refund amount:', refund.amount) // 500
  console.log('Original amount:', refund.originalAmount) // e.g. 2000
  console.log('Type:', refund.refundType) // partial
}

// ---------------------------------------------------------------------------
// 3. Refund with beneficiary name
// ---------------------------------------------------------------------------
async function refundViaBankAccount(transactionId: string) {
  const refund = await zenpays.refunds.create({
    transactionId,
    amount: 750,
    reason: 'Order cancelled',
    beneficiaryEmail: 'rahul@example.com',
    beneficiaryAccount: '50100012345678',
    beneficiaryIfsc: 'HDFC0001234',
    beneficiaryName: 'Rahul Sharma',
  })

  console.log('Bank refund:', refund.refundId)
  console.log('Status:', refund.status)
}

// ---------------------------------------------------------------------------
// 4. Multiple partial refunds on the same transaction
// ---------------------------------------------------------------------------
async function progressivePartialRefunds(transactionId: string) {
```

```
  // Original transaction: 1000 INR

  // First partial refund: 300 INR
  const refund1 = await zenpays.refunds.create({
    transactionId,
    amount: 300,
    reason: 'Damaged item',
    beneficiaryEmail: 'customer@example.com',
    beneficiaryAccount: '91020034567890',
    beneficiaryIfsc: 'SBIN0001234',
  })
  console.log(`Refund 1: ${refund1.refundId}, amount: 300`)
  // Transaction: refundedAmount = 300, status = "partial_refund"

  // Second partial refund: 200 INR
  const refund2 = await zenpays.refunds.create({
    transactionId,
    amount: 200,
    reason: 'Shipping overcharge',
    beneficiaryEmail: 'customer@example.com',
    beneficiaryAccount: '91020034567890',
    beneficiaryIfsc: 'SBIN0001234',
  })
  console.log(`Refund 2: ${refund2.refundId}, amount: 200`)
  // Transaction: refundedAmount = 500, status = "partial_refund"

  // Final refund: remaining 500 INR
  const refund3 = await zenpays.refunds.create({
    transactionId,
    amount: 500,
    reason: 'Remaining balance',
    beneficiaryEmail: 'customer@example.com',
    beneficiaryAccount: '91020034567890',
    beneficiaryIfsc: 'SBIN0001234',
  })
  console.log(`Refund 3: ${refund3.refundId}, amount: 500`)
  // Transaction: refundedAmount = 1000, status = "refunded"

  // This would fail -- already fully refunded:
  // await zenpays.refunds.create({ transactionId, amount: 1, ... })
  // Error: "Transaction has already been fully refunded"
}

// ---------------------------------------------------------------------------
// 5. Batch refund
// ---------------------------------------------------------------------------
async function batchRefund() {
  const result = await zenpays.refunds.bulkUpload({
    refunds: [
      {
        transactionId: 'txn_aaa',
        reason: 'Customer request',
        beneficiaryEmail: 'john@example.com',
        beneficiaryAccount: '30200045678901',
        beneficiaryIfsc: 'ICIC0001234',
      },
      {
        transactionId: 'txn_bbb',
        amount: 500,
        reason: 'Partial refund',
        beneficiaryEmail: 'jane@example.com',
        beneficiaryAccount: '1234567890',
        beneficiaryIfsc: 'HDFC0001234',
      },
    ],
```

```
  })

  console.log('Total:', result.total)
  console.log('Succeeded:', result.succeeded)
  console.log('Failed:', result.failed)

  result.results.forEach((r) => {
    if (r.status === 'success') {
      console.log(`  ${r.transactionId}: Refund ${r.refundId}`)
    } else {
      console.log(`  ${r.transactionId}: Failed - ${r.error}`)
    }
  })
}

// ---------------------------------------------------------------------------
// 6. Poll for refund status
// ---------------------------------------------------------------------------
async function waitForRefund(refundId: string): Promise<string> {
  const refund = await zenpays.refunds.get(refundId)

  switch (refund.status) {
    case 'completed':
      console.log('Refund completed successfully')
      return 'completed'

    case 'failed':
      console.log('Refund failed:', refund.failureReason)
      return 'failed'

    case 'rejected':
      console.log('Refund rejected')
      return 'rejected'

    case 'cancelled':
      console.log('Refund cancelled')
      return 'cancelled'

    default:
      // pending_approval, approved, processing -- keep polling
      console.log('Refund status:', refund.status)
      await new Promise(resolve => setTimeout(resolve, 5000))
      return waitForRefund(refundId)
  }
}

// ---------------------------------------------------------------------------
// 7. Get refund statistics
// ---------------------------------------------------------------------------
async function getRefundStats() {
  const stats = await zenpays.refunds.getStats('2026-01-01', '2026-12-31', 'INR')

  console.log('Refund Statistics:')
  console.log('  Total refunds:', stats.totalCount)
  console.log('  Total amount:', stats.totalAmount, 'INR')
  console.log('  Pending:', stats.pendingCount)
  console.log('  Completed:', stats.completedCount)
  console.log('  Failed:', stats.failedCount)
}
```

PYTHON

```python
import os
import time
from zenpays import ZenPays

zenpays = ZenPays(api_key=os.environ["ZENPAYS_API_KEY"])


# ---------------------------------------------------------------------------
# 1. Full refund
# ---------------------------------------------------------------------------
def full_refund(transaction_id: str):
    refund = zenpays.refunds.create({
        "transactionId": transaction_id,
        "reason": "Customer requested full refund",
        "beneficiaryEmail": "priya@example.com",
        "beneficiaryAccount": "50100012345678",
        "beneficiaryIfsc": "HDFC0001234",
    })

    print(f"Refund created: {refund['refundId']}")
    print(f"Amount: {refund['amount']} {refund['currency']}")
    print(f"Status: {refund['status']}")  # pending_approval


# ---------------------------------------------------------------------------
# 2. Partial refund
# ---------------------------------------------------------------------------
def partial_refund(transaction_id: str):
    refund = zenpays.refunds.create({
        "transactionId": transaction_id,
        "amount": 500,
        "reason": "Partial item return",
        "beneficiaryEmail": "priya@example.com",
        "beneficiaryAccount": "50100012345678",
        "beneficiaryIfsc": "HDFC0001234",
    })

    print(f"Partial refund: {refund['refundId']}")
    print(f"Refund amount: {refund['amount']}")  # 500
    print(f"Original amount: {refund['originalAmount']}")  # e.g. 2000
    print(f"Type: {refund['refundType']}")  # partial


# ---------------------------------------------------------------------------
# 3. Refund with beneficiary name
# ---------------------------------------------------------------------------
def refund_via_bank_account(transaction_id: str):
    refund = zenpays.refunds.create({
        "transactionId": transaction_id,
        "amount": 750,
        "reason": "Order cancelled",
        "beneficiaryEmail": "rahul@example.com",
        "beneficiaryAccount": "50100012345678",
        "beneficiaryIfsc": "HDFC0001234",
        "beneficiaryName": "Rahul Sharma",
    })

    print(f"Bank refund: {refund['refundId']}")
    print(f"Status: {refund['status']}")


# ---------------------------------------------------------------------------
```

```python
# 4. Multiple partial refunds on the same transaction
# ------------------------------------------------------------------------
def progressive_partial_refunds(transaction_id: str):
    # Original transaction: 1000 INR

    refund1 = zenpays.refunds.create({
        "transactionId": transaction_id,
        "amount": 300,
        "reason": "Damaged item",
        "beneficiaryEmail": "customer@example.com",
        "beneficiaryAccount": "91020034567890",
        "beneficiaryIfsc": "SBIN0001234",
    })
    print(f"Refund 1: {refund1['refundId']}, amount: 300")
    # Transaction: refundedAmount = 300, status = "partial_refund"

    refund2 = zenpays.refunds.create({
        "transactionId": transaction_id,
        "amount": 200,
        "reason": "Shipping overcharge",
        "beneficiaryEmail": "customer@example.com",
        "beneficiaryAccount": "91020034567890",
        "beneficiaryIfsc": "SBIN0001234",
    })
    print(f"Refund 2: {refund2['refundId']}, amount: 200")
    # Transaction: refundedAmount = 500, status = "partial_refund"

    refund3 = zenpays.refunds.create({
        "transactionId": transaction_id,
        "amount": 500,
        "reason": "Remaining balance",
        "beneficiaryEmail": "customer@example.com",
        "beneficiaryAccount": "91020034567890",
        "beneficiaryIfsc": "SBIN0001234",
    })
    print(f"Refund 3: {refund3['refundId']}, amount: 500")
    # Transaction: refundedAmount = 1000, status = "refunded"


# ------------------------------------------------------------------------
# 5. Batch refund
# ------------------------------------------------------------------------
def batch_refund():
    result = zenpays.refunds.bulk_upload({
        "refunds": [
            {
                "transactionId": "txn_aaa",
                "reason": "Customer request",
                "beneficiaryEmail": "john@example.com",
                "beneficiaryAccount": "30200045678901",
                "beneficiaryIfsc": "ICIC0001234",
            },
            {
                "transactionId": "txn_bbb",
                "amount": 500,
                "reason": "Partial refund",
                "beneficiaryEmail": "jane@example.com",
                "beneficiaryAccount": "1234567890",
                "beneficiaryIfsc": "HDFC0001234",
            },
        ],
    })

    print(f"Total: {result['total']}")
    print(f"Succeeded: {result['succeeded']}")
```

```
    print(f"Failed: {result['failed']}")

    for r in result["results"]:
        if r["status"] == "success":
            print(f"  {r['transactionId']}: Refund {r['refundId']}")
        else:
            print(f"  {r['transactionId']}: Failed - {r['error']}")


# ---------------------------------------------------------------------------
# 6. Poll for refund status
# ---------------------------------------------------------------------------
def wait_for_refund(refund_id: str) -> str:
    refund = zenpays.refunds.get(refund_id)

    if refund["status"] == "completed":
        print("Refund completed successfully")
        return "completed"

    if refund["status"] == "failed":
        print(f"Refund failed: {refund.get('failureReason')}")
        return "failed"

    if refund["status"] in ("rejected", "cancelled"):
        print(f"Refund {refund['status']}")
        return refund["status"]

    # pending_approval, approved, processing -- keep polling
    print(f"Refund status: {refund['status']}")
    time.sleep(5)
    return wait_for_refund(refund_id)


# ---------------------------------------------------------------------------
# 7. Get refund statistics
# ---------------------------------------------------------------------------
def get_refund_stats():
    stats = zenpays.refunds.get_stats("2026-01-01", "2026-12-31", "INR")

    print("Refund Statistics:")
    print(f"  Total refunds: {stats['totalCount']}")
    print(f"  Total amount: {stats['totalAmount']} INR")
    print(f"  Pending: {stats['pendingCount']}")
    print(f"  Completed: {stats['completedCount']}")
    print(f"  Failed: {stats['failedCount']}")
```

PART

Resources

2 documents

RESOURCES

Postman Collection

Postman Collection

Test and explore the ZenPays API using our pre-configured Postman collection with 60+ API endpoints across 10 categories.

Download

Download the Postman collection (JSON)

Features

- **60+ Pre-configured Endpoints** - All API endpoints ready to use
- **Auto HMAC Signing** - Pre-request script auto-generates `X-Signature` and `X-Timestamp` headers
- **Example Request Bodies** - Realistic sample data for every endpoint
- **Environment Variables** - Easy switching between sandbox and production
- **Optional Query Params** - Disabled filters you can toggle on as needed

Included Endpoints

Category	Endpoints	Description
Payment Intents	3	Create, get, and confirm payments
Transactions	4	List, search, and export
Customers	7	Create, update, risk, top customers
Payouts	8	Create, preview, retry, stats, export
Refunds	5	Create, list, cancel, stats
Settlements	9	Create, cancel, bank accounts management
Wallet	4	Balances, transactions, summary
Ledger	8	Entries, balances, reserves, reconciliation
Reports	5	Generate, list, download, delete
Health	1	Health check

Quick Start

1. **Download** the collection using the button above
2. **Open Postman** and click "Import"
3. **Select** the downloaded JSON file
4. **Set variables**: `apiKey` and `secretSalt` from your ZenPays Dashboard
5. **Start testing** - signatures are generated automatically!

Collection Variables

Variable	Description
`baseUrl`	API base URL (default: `https://api.zenpays.com`)
`apiKey`	Your API key (e.g., `zp_test_xxxxx` or `zp_live_xxxxx`)
`secretSalt`	Your secret salt for HMAC signature generation

Authentication

All requests use **API Key + HMAC Signature** authentication. The collection includes a **pre-request script** that automatically generates all required headers before every request — you just need to set your credentials once.

Required Headers (auto-generated)

Every request sends these 5 headers, all handled automatically by the pre-request script:

Header	Value	Description
`Authorization`	`Bearer {apiKey}`	Your API key from the collection variables
`X-Timestamp`	ISO 8601 UTC timestamp	Generated at request time (e.g., `2024-01-15T10:30:00.000Z`)
`X-Signature`	HMAC-SHA256 hex string	Computed from timestamp + body using your secret salt
`X-Secret-Salt`	Your secret salt	Passed for server-side HMAC validation
`Content-Type`	`application/json`	Set on all requests

How the Pre-Request Script Works

The collection's pre-request script runs before every request and does the following:

```
// 1. Reads your credentials from collection variables
const apiKey = pm.collectionVariables.get('apiKey');
const secretSalt = pm.collectionVariables.get('secretSalt');

// 2. Generates a fresh timestamp
const timestamp = new Date().toISOString();

// 3. Compacts the request body (strips whitespace/newlines)
//    For GET requests or empty bodies, uses '{}'
const body = pm.request.body?.raw
  ? JSON.stringify(JSON.parse(pm.request.body.raw))
  : '{}';

// 4. Computes HMAC-SHA256 signature
const signature = CryptoJS.HmacSHA256(timestamp + body, secretSalt).toString();

// 5. Sets all required headers automatically
pm.request.headers.upsert({ key: 'Authorization', value: 'Bearer ' + apiKey });
pm.request.headers.upsert({ key: 'X-Timestamp', value: timestamp });
pm.request.headers.upsert({ key: 'X-Signature', value: signature });
pm.request.headers.upsert({ key: 'X-Secret-Salt', value: secretSalt });
pm.request.headers.upsert({ key: 'Content-Type', value: 'application/json' });
```

Important

The `apiKey` collection variable should contain **only** the key itself (e.g., `zp_test_abc123`), **not** the `Bearer` prefix. The script adds `Bearer` automatically.

Environment Setup

Option 1: Collection Variables (Recommended)

After importing, set the variables directly in the collection:

1. Click the collection name **"ZenPays API"** in the sidebar
2. Go to the **Variables** tab
3. Set these values in the **Current Value** column:

Variable	Current Value
`baseUrl`	`https://api.zenpays.com`
`apiKey`	`zp_test_your_key_here`
`secretSalt`	`your_secret_salt_here`

4. Click **Save**

Option 2: Postman Environments

Create a Postman Environment for each stage:

1. Click the **Environments** tab (gear icon) in the sidebar
2. Click **+** to create a new environment
3. Name it (e.g., "ZenPays Sandbox")
4. Add the same 3 variables: `baseUrl`, `apiKey`, `secretSalt`

5. Select the environment from the dropdown in the top-right corner

> **Tip**
> If you use **both** environment variables and collection variables, environment variables take priority. Make sure to clear any environment overrides if you want the collection variables to be used.

Environments

Environment	Base URL
Sandbox	`https://api.zenpays.com`

Troubleshooting

Issue	Cause	Fix
`401 Missing Authorization header`	Pre-request script not running	Re-import the collection; ensure `apiKey` and `secretSalt` are set
`401 Invalid HMAC signature`	Body format mismatch or wrong secret salt	Verify `secretSalt` is correct; check Postman Console for sent headers
`Bearer Bearer zp_test_...` in logs	`apiKey` variable includes `Bearer` prefix	Remove `Bearer` from the `apiKey` value — the script adds it
Headers not appearing on request	Environment variable overrides with empty values	Clear or delete environment variable overrides
`Request timestamp expired`	System clock drift or stale request	Ensure your system clock is accurate; the signature expires after 5 minutes

Tips

- **View Console** - Use Postman Console (**View → Show Postman Console**) to see the exact headers being sent and debug authentication issues
- **Toggle filters** - Many GET endpoints have optional query parameters (disabled by default) — enable the ones you need
- **Idempotency** - POST endpoints include `X-Idempotency-Key` headers to prevent duplicate requests
- **Use environments** - Create separate Postman environments for sandbox vs production to avoid accidental production calls

RESOURCES

Using ZenPays docs with AI tools

Every page on this site is published as plain Markdown as well as HTML, so a coding agent can read the real source instead of scraping a rendered page.

Copy one page

Every page has a **Copy page** button at the top right. It copies that page as Markdown, ready to paste into ChatGPT, Claude, Cursor or Copilot Chat.

View as Markdown opens the same content as a plain file, which is the URL to give a tool that fetches its own context.

Fetch a page directly

Append the page's path to `/md/` and add `.md`:

```
# The HTML page
https://docs.zenpays.com/docs/rest-api/endpoints/payments/create-payment-intent

# The same page as Markdown
curl https://docs.zenpays.com/md/rest-api/endpoints/payments/create-payment-intent.md
```

The Markdown is the page's own source with the site's plumbing removed. Code samples keep their language labels, so a model can tell the cURL example from the TypeScript one.

Point an agent at the whole site

`llms.txt` is an index of every page, grouped by section, linking to the Markdown rather than the HTML:

```
curl https://docs.zenpays.com/llms.txt
```

Give an agent that one URL and it can find and fetch any page it needs.

Cursor

Settings → **Features** → **Docs** → **Add new doc**, and enter:

```
https://docs.zenpays.com/llms.txt
```

Then reference it in chat with `@ZenPays`.

Claude Code

Point Claude at the index in a prompt, or add it to your project's `CLAUDE.md` so it is always available:

```
ZenPays API reference: https://docs.zenpays.com/llms.txt
Fetch any page as Markdown from https://docs.zenpays.com/md/<path>.md
```

Anything that fetches URLs

Give it `https://docs.zenpays.com/llms.txt` and it can navigate from there. No key, no account, no rate limit.

Offline copies

For a model with a large context window, or for a reader with no network, the PDF downloads contain the entire documentation set as four volumes. The complete edition is a single file covering every published page.

> **Tip**
> When you ask an AI to write ZenPays integration code, paste the specific endpoint page rather than describing the endpoint. Amounts in particular are easy to get wrong from memory — every `amount` is an integer in the currency's smallest unit, and the divisor is not always 100.