ZenPays Documentation

SDK Reference

SDK v0.5.0 · 21 documents
Edition 22 September 2026 · commit unknown
docs.zenpayz.com

Contents

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.zenpayz.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 (https://docs.zenpayz.com/docs/guides/webhooks/signature-verification).

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 (https://docs.zenpayz.com/docs/guides/error-handling) 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

```
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

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

listReports

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

getReport

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

downloadReport

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

Report Schedules

createReportSchedule

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

listReportSchedules

```
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

```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

```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

```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.zenpayz.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.zenpayz.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.zenpayz.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.