Welcome to x402-stacks

Welcome to x402-stacks

x402-stacks is the open payment standard that enables services to charge for access to their APIs and content directly over HTTP using the Stacks blockchain with STX or sBTC cryptocurrency.

It is built around the HTTP `402 Payment Required` status code and allows clients to programmatically pay for resources without accounts, sessions, or credential management.

With x402-stacks, any web service can require payment before serving a response, using crypto-native payments for speed, privacy, and efficiency.

x402-stacks v2 is now Coinbase x402 protocol compatible, using standardized CAIP-2 network identifiers and base64-encoded payment headers.

Why Use x402-stacks?

x402-stacks addresses key limitations of existing payment systems:

- High fees and friction with traditional credit cards and fiat payment processors

- Incompatibility with machine-to-machine payments, such as AI agents
- Lack of support for micropayments, making it difficult to monetize usage-based services
- Native STX and sBTC support for the Stacks ecosystem
- Coinbase x402 compatible for interoperability with the broader ecosystem

Who is x402-stacks for?

- Sellers: Service providers who want to monetize their APIs or content using STX or sBTC. x402-stacks enables direct, programmatic payments from clients with minimal setup.
- Buyers: Human developers and AI agents seeking to access paid services without accounts or manual payment flows.

Both sellers and buyers interact directly through HTTP requests, with payment handled transparently through the protocol.

What Can You Build?

x402-stacks enables a range of use cases, including:

- API services paid per request in STX or sBTC
- AI agents that autonomously pay for API access
- Paywalls for digital content
- Microservices and tooling monetized via microtransactions
- Proxy services that aggregate and resell API capabilities

How Does It Work?

At a high level, the flow is simple:

A buyer requests a resource from a server.
If payment is required, the server responds with `402 Payment Required`, including payment instructions.
The buyer prepares and submits a payment payload signed with their STX wallet.
The server verifies and settles the payment using the x402-stacks facilitator.
If payment is valid, the server provides the requested resource.

Resources

- npm package: x402-stacks
- Example Repository
- Main Repository

- x402scan - API Discovery
- Facilitator Service

Get Started

Ready to build? Start here:

- Quickstart for Sellers
- Quickstart for Buyers
- Explore Core Concepts

Upgrading from v1? Check out the Migration Guide for all the changes.

Last updated: January 2026Edit this page on GitHub

On this page

x402 stacks
v2.0.0
Documentation

Quickstart for Buyers

Quickstart for Buyers

This guide walks you through how to use x402-stacks to interact with services that require payment. By the end of this guide, you will be able to programmatically discover payment requirements, complete a payment, and access a paid resource.

Note: This guide uses the V2 API (Coinbase compatible). See the Legacy V1 API section below if you're using an older version.

Prerequisites

Before you begin, ensure you have:

- A Stacks wallet with STX (or generate one using the SDK)
- Node.js and npm installed
- A service that requires payment via x402-stacks

1. Install Dependencies

```
npm install x402-stacks axios dotenv
```

2. Create or Load a Wallet

x402-stacks provides utilities to either load an existing wallet or generate a new one:

```
import {
  privateKeyToAccount,
  generateKeypair,
} from 'x402-stacks';

const NETWORK = 'testnet'; // or 'mainnet'

// Option 1: Load existing wallet from private key
const account = privateKeyToAccount(process.env.CLIENT_PRIVATE_KEY!,
NETWORK);
console.log('Using wallet:', account.address);

// Option 2: Generate a new wallet
const keypair = generateKeypair(NETWORK);
console.log('New wallet generated:', keypair.address);
```

```
console.log('Private key:', keypair.privateKey);
console.log('Fund this wallet with STX before making payments');

// Use the generated keypair
const newAccount = privateKeyToAccount(keypair.privateKey, NETWORK);
```

Important: Store your private key securely! Add it to your `.env` file as `CLIENT_PRIVATE_KEY` and never commit it to version control.

3. Make Paid Requests Automatically

x402-stacks provides an Axios wrapper that automatically handles 402 Payment Required responses:

```
import 'dotenv/config';
import axios from 'axios';
import {
  wrapAxiosWithPayment,
  privateKeyToAccount,
  decodePaymentResponse,
  generateKeypair,
  getExplorerURL,
} from 'x402-stacks';

const NETWORK = (process.env.NETWORK as 'mainnet' | 'testnet') ||
'testnet';
const SERVER_URL = process.env.SERVER_URL || 'http://localhost:3003';

// Load or generate account
let account;

if (process.env.CLIENT_PRIVATE_KEY) {
  account = privateKeyToAccount(process.env.CLIENT_PRIVATE_KEY,
NETWORK);
  console.log('Using wallet:', account.address);
} else {
  const keypair = generateKeypair(NETWORK);
  console.log('New wallet generated:', keypair.address);
```

```
  console.log('Private key:', keypair.privateKey);
  console.log('Add to .env: CLIENT_PRIVATE_KEY and fund with STX');
  account = privateKeyToAccount(keypair.privateKey, NETWORK);
}

// Create axios with automatic payment handling (V2)
const api = wrapAxiosWithPayment(
  axios.create({
    baseURL: SERVER_URL,
    timeout: 60000,
  }),
  account
);

async function main() {
  // Check server health
  try {
    await axios.get(`${SERVER_URL}/health`);
    console.log('Server is running');
  } catch {
    console.error('Server not running. Start with: npm run
dev:server');
    return;
  }

  // Make paid request - payment is handled automatically!
  try {
    console.log('Requesting premium data...');
    const response = await api.get('/api/premium-data');

    console.log('Success! Data:', response.data);

    // Decode payment response from headers (V2 uses base64-encoded
'payment-response')
    const paymentResponse = decodePaymentResponse(
      response.headers['payment-response']
    );
    if (paymentResponse) {
```

```
      console.log('Payment transaction:',
paymentResponse.transaction);
      console.log('Explorer:',
getExplorerURL(paymentResponse.transaction, NETWORK));
    }
  } catch (error: any) {
    console.error('Error:', error.response?.data?.error ||
error.message);
  }
}

main().catch(console.error);
```

4. Environment Variables

Create a `.env` file:

```
NETWORK=testnet
SERVER_URL=http://localhost:3003
CLIENT_PRIVATE_KEY=your_private_key_here
```

5. Error Handling

Clients will throw errors if:

- The wallet has insufficient STX balance
- The payment signature is invalid
- The server's payment requirements cannot be met
- Network connectivity issues occur

Summary

- Install the `x402-stacks` package
- Create or load a Stacks wallet using `privateKeyToAccount` or `generateKeypair`
- Use `wrapAxiosWithPayment` to wrap your Axios instance
- Payment flows are handled automatically for you
- Use `decodePaymentResponse` to get transaction details

Legacy V1 API

If you're using x402-stacks < 2.0, use these functions instead:

V2 (Recommended)	V1 (Legacy)
`wrapAxiosWithPayment()`	`withPaymentInterceptor()`
`decodePaymentResponse()`	`decodeXPaymentResponse()`
`payment-response` header	`x-payment-response` header

```
// V1 Legacy Example
import { withPaymentInterceptor, decodeXPaymentResponse } from 'x402-stacks';

const api = withPaymentInterceptor(axios.create(), account);
const response = await api.get('/api/premium-data');
const payment = decodeXPaymentResponse(response.headers['x-payment-response']);
```

See the Migration Guide for full details.

References

- x402-stacks npm package
- Example Repository
- Stacks Explorer - Testnet
- Stacks Explorer - Mainnet

Last updated: January 2026Edit this page on GitHub

On this page

Quickstart for Sellers

Quickstart for Sellers

This guide walks you through integrating with x402-stacks to enable payments for your API or service. By the end, your API will be able to charge buyers and AI agents for access using STX.

Note: This guide uses the V2 API (Coinbase compatible). See the Legacy V1 API section below if you're using an older version.

Prerequisites

Before you begin, ensure you have:

- A Stacks wallet address to receive STX payments
- Node.js and npm installed
- An existing Express API or server

1. Install Dependencies

```
npm install x402-stacks express dotenv
```

```
npm install -D @types/express typescript ts-node
```

2. Add Payment Middleware

Integrate the payment middleware into your Express application. You will need to provide:

- Your receiving wallet address (`payTo`)
- The payment amount in microSTX
- The network (mainnet or testnet, or CAIP-2 format)
- The facilitator URL

```
import 'dotenv/config';
import express, { Request, Response } from 'express';
import { paymentMiddleware, getPayment, STXtoMicroSTX } from
'x402-stacks';

const NETWORK = (process.env.NETWORK as 'mainnet' | 'testnet') ||
'testnet';
const SERVER_ADDRESS = process.env.SERVER_ADDRESS!;
const PORT = process.env.PORT || 3003;
const FACILITATOR_URL = process.env.FACILITATOR_URL ||
'https://facilitator.stacksx402.com';

const app = express();
app.use(express.json());

// Protected endpoint - requires STX payment (V2)
app.get(
  '/api/premium-data',
  paymentMiddleware({
    amount: STXtoMicroSTX(0.00001), // 0.00001 STX = 10 microSTX
    payTo: SERVER_ADDRESS,          // V2 uses 'payTo' instead of
'address'
    network: NETWORK,               // Accepts 'testnet', 'mainnet', or
CAIP-2 format
    facilitatorUrl: FACILITATOR_URL,
  }),
  (req: Request, res: Response) => {
    // Get payment details from the verified request
```

```
    const payment = getPayment(req);

    res.json({
      success: true,
      message: 'Premium data access granted!',
      data: {
        secretValue: 42,
        timestamp: new Date().toISOString()
      },
      payment: {
        transaction: payment?.transaction,
        payer: payment?.payer,
        network: payment?.network,
      },
    });
  }
);

// Health check endpoint (no payment required)
app.get('/health', (req: Request, res: Response) => {
  res.json({ status: 'ok', network: NETWORK });
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
  console.log(`Network: ${NETWORK}`);
  console.log(`Facilitator: ${FACILITATOR_URL}`);
});
```

3. Environment Variables

Create a `.env` file:

```
NETWORK=testnet
SERVER_ADDRESS=SP2... # Your Stacks wallet address
PORT=3003
FACILITATOR_URL=https://facilitator.stacksx402.com
```

4. Understanding the Middleware

paymentMiddleware Options (V2)

Option	Type	Description
`amount`	string | bigint	Payment amount in microSTX
`payTo`	string	Your Stacks wallet address to receive payments
`network`	string	Network: `'testnet'`, `'mainnet'`, or CAIP-2 format (`'stacks:1'`, `'stacks:2147483648'`)
`facilitatorUrl`	string	URL of the payment facilitator service
`asset`	string	(Optional) Asset type: `'STX'`, `'sBTC'`, or contract identifier
`description`	string	(Optional) Human-readable resource description
`maxTimeoutSeconds`	number	(Optional) Payment timeout, default 300

Helper Functions

- `STXtoMicroSTX(stx)`: Converts STX to microSTX (1 STX = 1,000,000 microSTX)
- `getPayment(req)`: Extracts verified payment details from the request

5. Test Your Integration

Start your server:

```
npx ts-node server.ts
```

Make a request without payment:

```
curl http://localhost:3003/api/premium-data
```

The server responds with `402 Payment Required` and payment instructions.

Use a compatible client (see Quickstart for Buyers) to complete the payment flow.

6. Payment Flow (V2)

When a request is made to a protected endpoint:

Middleware checks for `payment-signature` header (base64-encoded)
If missing, returns `402 Payment Required` with `payment-required` header
If present, verifies and settles payment via the facilitator
If valid, attaches payment info to request and calls your handler
Response includes `payment-response` header with settlement details

Summary

- Install `x402-stacks` and Express dependencies
- Use `paymentMiddleware` on protected routes
- Configure `payTo` (your wallet address) and `facilitatorUrl`
- Use `getPayment(req)` to access verified payment details

Your API is now ready to accept STX payments through x402-stacks!

Legacy V1 API

If you're using x402-stacks < 2.0, use these functions instead:

V2 (Recommended)	V1 (Legacy)
`paymentMiddleware()`	`x402PaymentRequired()`
`payTo` config	`address` config
`payment-signature` header	`X-PAYMENT` header
`payment-response` header	`X-PAYMENT-RESPONSE` header

```
// V1 Legacy Example
import { x402PaymentRequired, getPayment } from 'x402-stacks';
```

```
app.get('/api/premium-data',
  x402PaymentRequired({
    amount: STXtoMicroSTX(0.00001),
    address: SERVER_ADDRESS,  // V1 uses 'address'
    network: NETWORK,
    facilitatorUrl: FACILITATOR_URL,
  }),
  handler
);
```

See the Migration Guide for full details.

References

- x402-stacks npm package
- Example Repository
- Main Repository

Last updated: January 2026Edit this page on GitHub

On this page

x402 stacks

v2.0.0

Documentation

HTTP 402
Client / Server
Facilitator
Docs

Register with x402scan
Register with x402scan

Once your x402 API is live, register it with x402scan to make it discoverable by buyers and AI agents in the Stacks ecosystem.

What is x402scan?

x402scan is a registry and discovery service for x402 endpoints. It allows:

- Sellers to list their payment-enabled APIs
- Buyers to discover available services and their pricing
- AI agents to programmatically find and pay for API access

How to Register

Visit scan.stacksx402.com and submit your API URL. The system will automatically fetch and validate your x402 schema.

Note: Re-registering the same URL updates your existing listing.

Schema Requirements

When x402scan fetches your URL, your endpoint must return a valid x402 response. Here are the requirements:

Required Fields

Field	Requirement
`name`	Must be non-empty

`accepts`	Must be a non-empty array
`accepts[].network`	Must be `"stacks"`
`accepts[].outputSchema`	Required for x402scan listing

URL Requirements

- Must use HTTPS (HTTP URLs are rejected)
- Must return HTTP `402 Payment Required` or `200 OK`
- Must return valid JSON with the x402 schema

Complete Schema Example

Your endpoint should return this structure:

```
{
  "x402Version": 1,
  "name": "Weather API",
  "image": "https://your-api.com/logo.png",
  "accepts": [
    {
      "scheme": "exact",
      "network": "stacks",
      "maxAmountRequired": "1000000",
      "resource": "https://your-api.com/weather",
      "description": "Get current weather data for any city",
      "mimeType": "application/json",
      "payTo": "SP2J6ZY48GV1EZ5V2V5RB9MP66SW86PYKKNRV9EJ7",
      "maxTimeoutSeconds": 60,
      "asset": "STX",
      "outputSchema": {
        "input": {
          "type": "request",
          "method": "GET",
          "queryParams": {
            "city": {
```

```
              "type": "string",
              "required": true,
              "description": "City name to get weather for"
            },
            "units": {
              "type": "string",
              "required": false,
              "description": "Temperature units",
              "enum": ["celsius", "fahrenheit"]
            }
          }
        },
        "output": {
          "temperature": { "type": "number" },
          "conditions": { "type": "string" },
          "humidity": { "type": "number" }
        }
      }
    }
  ]
}
```

outputSchema Deep Dive

The `outputSchema` field is required for x402scan. It describes how to call your API and what it returns, enabling:

- Documentation generation
- Client code generation
- AI agent integration

Input Definition

```
{
  "input": {
    "type": "request",
    "method": "GET",
    "queryParams": {
      "paramName": {
```

```
        "type": "string",
        "required": true,
        "description": "What this parameter does"
      }
    }
  }
}
```

Supported input fields:

Field	Description
`type`	Always `"request"`
`method`	HTTP method: `GET`, `POST`, etc.
`queryParams`	URL query parameters
`bodyFields`	Request body fields (for POST/PUT)
`bodyType`	Body content type (e.g., `"json"`)
`headerFields`	Custom header fields

Field Definition

Each parameter or field is defined as:

```
{
  "type": "string",
  "required": true,
  "description": "Human-readable description",
  "enum": ["option1", "option2"]
}
```

Output Definition

Describe your API's response structure:

```
{
  "output": {
    "fieldName": { "type": "string" },
    "nestedObject": {
      "properties": {
        "innerField": { "type": "number" }
      }
    }
  }
}
```

Validation Errors

If registration fails, you'll receive one of these errors:

Error	Cause
`invalid_url`	URL is not valid HTTPS
`invalid_network`	Network is not `"stacks"`
`missing_output_schema`	`outputSchema` is missing from accepts
`empty_accepts`	`accepts` array is empty
`invalid_name`	`name` field is empty

After Registration

Once registered, your API:

Appears in the x402scan directory at scan.stacksx402.com
Is discoverable via the x402scan API
Gets periodically re-validated to ensure schema freshness

Summary

Build your x402 API using the Quickstart for Sellers guide
Ensure your endpoint returns a valid schema with `outputSchema`
Register at scan.stacksx402.com
Your API is now discoverable by the x402 ecosystem!

Last updated: January 2026Edit this page on GitHub

On this page

x402 stacks
v2.0.0
Documentation

Migrating from V1 to V2

Migrating from V1 to V2

x402-stacks v2.0 introduces Coinbase-compatible protocol changes. This guide covers all breaking changes and how to update your code.

Why V2?

- Coinbase x402 protocol compatibility - Standardized protocol for interoperability
- CAIP-2 network identifiers - Industry-standard network identification
- Base64-encoded headers - Cleaner header format
- Improved naming - More intuitive function names

Quick Migration Checklist

- Update npm package: `npm install x402-stacks@latest`
- Replace deprecated function imports with V2 equivalents
- Change `address` to `payTo` in middleware config
- Update header parsing to use V2 functions
- Test with `https://facilitator.stacksx402.com`

Function Renames

Client-Side

V1 (Deprecated)	V2 (Recommended)	Notes
`withPaymentInterceptor()`	`wrapAxiosWithPayment()`	Wraps axios instance
`decodeXPaymentResponse()`	`decodePaymentResponse()`	Decodes settlement header
`x-payment-response` header	`payment-response` header	Base64-encoded JSON

Before (V1):

```
import { withPaymentInterceptor, decodeXPaymentResponse } from 'x402-stacks';

const api = withPaymentInterceptor(axios.create(), account);
const response = await api.get('/api/data');
const payment = decodeXPaymentResponse(response.headers['x-payment-response']);
```

After (V2):

```
import { wrapAxiosWithPayment, decodePaymentResponse } from
'x402-stacks';

const api = wrapAxiosWithPayment(axios.create(), account);
const response = await api.get('/api/data');
const payment =
decodePaymentResponse(response.headers['payment-response']);
```

Server-Side

V1 (Deprecated)	V2 (Recommended)	Notes
`x402PaymentRequired()`	`paymentMiddleware()`	Express middleware
`address` config	`payTo` config	Recipient address
`X-PAYMENT` header	`payment-signature` header	Client payment
`X-PAYMENT-RESPONSE` header	`payment-response` header	Settlement response

Before (V1):

```
import { x402PaymentRequired, getPayment } from 'x402-stacks';

app.get('/api/data',
  x402PaymentRequired({
    amount: STXtoMicroSTX(0.001),
    address: SERVER_ADDRESS,  // V1 config
    network: 'testnet',
    facilitatorUrl: FACILITATOR_URL,
  }),
  (req, res) => {
    const payment = getPayment(req);
    res.json({ txId: payment.txId });
  }
);
```

After (V2):

```
import { paymentMiddleware, getPayment } from 'x402-stacks';

app.get('/api/data',
  paymentMiddleware({
    amount: STXtoMicroSTX(0.001),
    payTo: SERVER_ADDRESS,  // V2 config - renamed!
    network: 'testnet',     // Also accepts CAIP-2: 'stacks:2147483648'
    facilitatorUrl: FACILITATOR_URL,
  }),
  (req, res) => {
    const payment = getPayment(req);
    res.json({ transaction: payment?.transaction });
  }
);
```

Network Format

V2 uses CAIP-2 network identifiers internally, but accepts both formats:

V1 Format	V2 CAIP-2 Format	Notes
`'mainnet'`	`'stacks:1'`	Stacks mainnet
`'testnet'`	`'stacks:2147483648'`	Stacks testnet

Note: V2 middleware auto-converts V1 network strings, so `'testnet'` still works.

HTTP Headers

Purpose	V1 Header	V2 Header	Encoding
Payment requirements	(body only)	`payment-required`	Base64 JSON

Client payment	`X-PAYMENT`	`payment-signature`	Base64 JSON
Settlement response	`X-PAYMENT-RESPONSE`	`payment-response`	Base64 JSON

Facilitator Endpoints

V1 Endpoint	V2 Endpoint
`/api/v1/verify`	`/verify`
`/api/v1/settle`	`/settle`
(none)	`/supported`

Payment Response Changes

V1 Response:

```
{
  txId: string;
  amount: number;
  sender: string;
}
```

V2 Response:

```
{
  success: boolean;
  transaction: string;  // renamed from txId
  payer: string;        // renamed from sender
  network: string;      // CAIP-2 format
  errorReason?: string; // if success is false
}
```

Backward Compatibility

The V2 library exports V1 functions for backward compatibility:

```
// These still work but are deprecated
import {
  withPaymentInterceptor,  // alias for wrapAxiosWithPayment
  x402PaymentRequired,     // alias for paymentMiddleware
  decodeXPaymentResponse,  // V1 decoder
} from 'x402-stacks';
```

We recommend migrating to V2 functions for better compatibility with the broader x402 ecosystem.

Need Help?

- x402-stacks npm package
- GitHub Repository
- Example Repository

Last updated: January 2026Edit this page on GitHub

On this page

Docs

HTTP 402

HTTP 402

For decades, HTTP 402 Payment Required has been reserved for future use. x402-stacks unlocks it for the Stacks ecosystem.

What is HTTP 402?

HTTP 402 is a standard, but rarely used, HTTP response status code indicating that payment is required to access a resource.

In x402-stacks, this status code is activated to:

- Inform clients (buyers or agents) that payment is required
- Communicate the details of the payment, such as amount in STX and destination address
- Provide the information necessary to complete the payment programmatically

402 Response Structure (V2)

When a server requires payment, it responds with a JSON body and a `payment-required` header (base64-encoded):

```
{
  "x402Version": 2,
  "resource": {
    "url": "https://api.example.com/premium-data",
    "description": "Premium API access"
  },
  "accepts": [
    {
      "scheme": "exact",
      "network": "stacks:2147483648",
      "amount": "10",
      "asset": "STX",
      "payTo": "SP2...",
      "maxTimeoutSeconds": 300
    }
  ]
}
```

Field	Description

`x402Version`	Protocol version (must be 2)
`resource`	Information about the protected resource
`accepts`	Array of accepted payment methods
`accepts[].network`	CAIP-2 network ID (`stacks:1` or `stacks:2147483648`)
`accepts[].amount`	Payment amount in atomic units (microSTX)
`accepts[].payTo`	Seller's Stacks wallet address
`accepts[].asset`	Asset type (`STX`, `sBTC`, or contract ID)

Why x402-stacks Uses HTTP 402

The primary purpose of HTTP 402 is to enable frictionless, API-native payments for accessing web resources, especially for:

- Machine-to-machine (M2M) payments (e.g., AI agents)
- Pay-per-use models such as API calls or paywalled content
- Micropayments without account creation or traditional payment rails
- STX-native payments for the Stacks ecosystem

Using the 402 status code keeps the protocol natively web-compatible and easy to integrate into any HTTP-based service.

Payment Headers (V2)

Payment Required Header

Servers include payment requirements in the `payment-required` header (base64-encoded JSON):

```
payment-required: eyJ4NDAyVmVyc2lvbiI6MiwicmVzb3VyY2UiOi4uLn0=
```

Payment Signature Header

Clients include signed payment in the `payment-signature` header (base64-encoded JSON):

```
payment-signature: eyJ4NDAyVmVyc2lvbiI6MiwiYWNjZXB0ZWQiOi4uLn0=
```

Payment Response Header

Servers include settlement details in the `payment-response` header (base64-encoded JSON):

```
payment-response: eyJzdWNjZXNzIjp0cnVlLCJ0cmFuc2FjdGlvbiI6Ii4uLiJ9
```

Decode using `decodePaymentResponse()` to get:

- `success`: Whether settlement succeeded
- `transaction`: Transaction hash on the Stacks blockchain
- `payer`: Payer's Stacks address
- `network`: CAIP-2 network identifier

V1 vs V2 Headers

V2 (Recommended)	V1 (Legacy)
`payment-required`	(body only)
`payment-signature`	`X-PAYMENT`
`payment-response`	`X-PAYMENT-RESPONSE`

Summary

HTTP 402 is the foundation of the x402-stacks protocol, enabling services to declare payment requirements directly within HTTP responses. It:

- Signals payment is required

- Communicates necessary payment details in STX
- Integrates seamlessly with standard HTTP workflows
- Works natively with the Stacks blockchain
- Uses standardized base64-encoded headers (V2)

Last updated: January 2026

Client / Server

Client / Server

This page explains the roles and responsibilities of the client and server in the x402-stacks protocol.

Understanding these roles is essential to designing, building, or integrating services that use x402-stacks for programmatic payments on Stacks.

Note: Client refers to the technical component making an HTTP request (the buyer). Server refers to the technical component responding to the request (the seller).

Client Role

The client is the entity that initiates a request to access a paid resource.

Clients can include:

- Human-operated applications
- Autonomous AI agents
- Programmatic services acting on behalf of users or systems

Client Responsibilities

- Initiate requests: Send an HTTP request to the resource server
- Handle payment requirements: Read the `402 Payment Required` response and `payment-required` header
- Prepare payment payload: Sign a transaction with their Stacks wallet
- Resubmit request with payment: Retry the request with the `payment-signature` header containing the signed payment payload (base64-encoded)

Client Code Example (V2)

```
import {
  wrapAxiosWithPayment,
  privateKeyToAccount
} from 'x402-stacks';
import axios from 'axios';

const account = privateKeyToAccount(privateKey, 'testnet');

// Wrap axios to handle payments automatically (V2)
const api = wrapAxiosWithPayment(
  axios.create({ baseURL: 'http://api.example.com' }),
  account
);

// Make request - payment handled automatically
const response = await api.get('/api/premium-data');
```

Clients do not need to manage accounts, credentials, or session tokens beyond their Stacks wallet. All interactions are stateless and occur over standard HTTP requests.

Server Role

The server is the resource provider enforcing payment for access to its services.

Servers can include:

- API services
- Content providers
- Any HTTP-accessible resource requiring monetization

Server Responsibilities

- Define payment requirements: Respond to unauthenticated requests with HTTP `402 Payment Required`, including payment details
- Verify payment payloads: Validate incoming payment payloads using the facilitator service
- Settle transactions: Upon successful verification, payment is settled on the Stacks blockchain

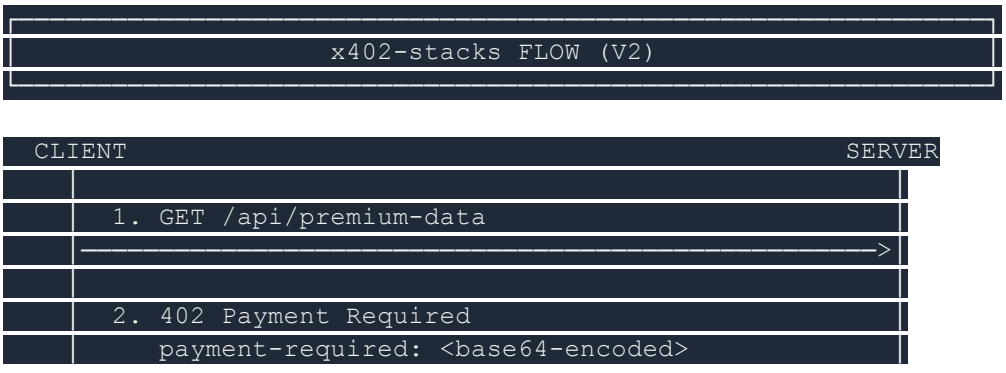
- Provide the resource: Once payment is confirmed, return the requested resource to the client

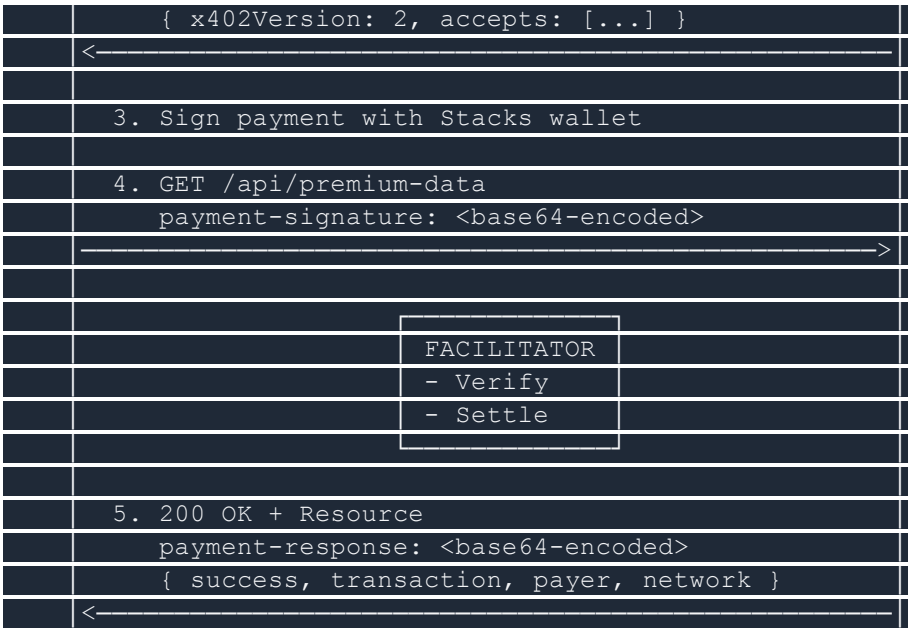
Server Code Example (V2)

```
import express from 'express';
import { paymentMiddleware, getPayment, STXtoMicroSTX } from
'x402-stacks';

const app = express();

app.get('/api/premium-data',
  paymentMiddleware({
    amount: STXtoMicroSTX(0.001), // 0.001 STX
    payTo: 'SP2...',              // V2 uses 'payTo' instead of
'address'
    network: 'testnet',           // Accepts 'testnet' or CAIP-2
'stacks:2147483648'
    facilitatorUrl: 'https://facilitator.stacksx402.com',
  }),
  (req, res) => {
    const payment = getPayment(req);
    res.json({ data: 'Premium content', transaction:
payment?.transaction });
  }
);
```

Servers do not need to manage client identities or maintain session state. Verification and settlement are handled per request.

Communication Flow (V2)





Summary

In the x402-stacks protocol:

- The client requests resources and supplies the signed payment payload using their Stacks wallet
- The server enforces payment requirements, verifies transactions via the facilitator, and provides the resource upon successful payment

This interaction is stateless, HTTP-native, and compatible with both human applications and automated agents.

Last updated: January 2026Edit this page on GitHu

Facilitator

Facilitator

This page explains the role of the facilitator in the x402-stacks protocol.

The facilitator is a service that simplifies the process of verifying and settling payments between clients (buyers) and servers (sellers) on the Stacks blockchain.

What is a Facilitator?

The facilitator is a service that:

- Verifies payment payloads submitted by clients
- Settles payments on the Stacks blockchain on behalf of servers

By using a facilitator, servers do not need to maintain direct blockchain connectivity or implement payment verification logic themselves. This reduces operational complexity and ensures accurate, real-time validation of transactions.

x402-stacks Facilitator

The official x402-stacks facilitator is hosted at:

```
https://facilitator.stacksx402.com
```

This facilitator:

- Works on both mainnet and testnet
- Handles STX payment verification
- Submits transactions to the Stacks blockchain
- Returns transaction IDs for tracking

Facilitator Responsibilities

1. Verify Payments

Confirm that the client's payment payload:

- Is properly signed by the client's Stacks wallet
- Meets the server's declared payment requirements (amount, recipient)
- Has not been previously used (replay protection)

2. Settle Payments

- Submit validated payments to the Stacks blockchain
- Monitor for transaction confirmation
- Return transaction details to the server

3. Provide Responses

Return verification and settlement results to the server, allowing the server to decide whether to fulfill the client's request.

Note: The facilitator does not hold funds or act as a custodian. It performs verification and execution of on-chain transactions based on signed payloads provided by clients.

Why Use a Facilitator?

Using a facilitator provides:

Benefit	Description
Reduced complexity	Servers don't need to interact directly with Stacks nodes
Protocol consistency	Standardized verification and settlement flows
Faster integration	Start accepting payments with minimal blockchain-specific development
Security	Centralized verification prevents common attack vectors

Interaction Flow (V2)

```
1. CLIENT makes HTTP request to RESOURCE SERVER

2. RESOURCE SERVER responds with 402 Payment Required
   Headers: payment-required: <base64-encoded>
   Body: {
     "x402Version": 2,
     "accepts": [{ "network": "stacks:2147483648", "payTo": "SP2...", ... }]
   }

3. CLIENT signs payment with Stacks wallet

4. CLIENT retries request with payment-signature header (base64-encoded)

5. RESOURCE SERVER sends payment to FACILITATOR /settle

6. FACILITATOR verifies signature and payment details
```

```
7. If valid, FACILITATOR submits to Stacks blockchain

8. FACILITATOR returns settlement response to RESOURCE SERVER

9. RESOURCE SERVER includes payment-response header (base64-encoded)
   and returns the requested resource to CLIENT
```

Configuration (V2)

When setting up a server, configure the facilitator URL:

```
import { paymentMiddleware } from 'x402-stacks';

app.get('/api/data',
  paymentMiddleware({
    amount: '1000',  // microSTX (string or bigint)
    payTo: 'SP2...', // V2 uses 'payTo'
    network: 'testnet', // or CAIP-2: 'stacks:2147483648'
    facilitatorUrl: 'https://facilitator.stacksx402.com',
  }),
  handler
);
```

Environment Variable

Set in your `.env` file:

```
FACILITATOR_URL=https://facilitator.stacksx402.com
```

Endpoints (V2)

The facilitator exposes these endpoints:

Endpoint	Method	Purpose
`/verify`	POST	Verify a payment payload without settling

`/settle`	POST	Verify and settle a payment on-chain
`/supported`	GET	List supported payment schemes and networks
`/health`	GET	Health check

Note: V1 used `/api/v1/verify` and `/api/v1/settle`. V2 uses the root paths directly.

Summary

The facilitator acts as an independent verification and settlement layer within the x402-stacks protocol. It helps servers:

- Confirm payments without blockchain infrastructure
- Submit transactions on-chain reliably
- Focus on their core API functionality

The official facilitator at `https://facilitator.stacksx402.com` is ready for both testnet and mainnet use.