

readme.md for @ecobridge.xyz/devicemanager

A comprehensive, TypeScript-first device manager for discovering and communicating with network devices. ☐

[npm version](#) [License: MIT](#)

☐ Overview

`@ecobridge.xyz/devicemanager` provides a unified, object-oriented API for discovering and controlling network devices. Whether you're building a document scanning workflow, managing printers, controlling smart home devices, or monitoring UPS systems — this library has you covered.

Supported Device Types:

- ☐ **Scanners** — eSCL (AirScan), SANE protocols
- ☐ **Printers** — IPP, JetDirect protocols
- ☐ **Speakers** — Sonos, AirPlay, Chromecast, DLNA
- ☐ **UPS Systems** — NUT, SNMP protocols
- ☐ **SNMP Devices** — Generic SNMP v1/v2c/v3 support
- ☐ **Smart Home** — Home Assistant integration (lights, switches, sensors, climate, locks, fans, cameras, covers)

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

📦 Installation

```
# Using pnpm (recommended)
pnpm add @ecobridge.xyz/devicemanager

# Using npm
npm install @ecobridge.xyz/devicemanager

# Using yarn
yarn add @ecobridge.xyz/devicemanager
```

📦 Quick Start

The OOP Pattern: Discovery → Selection → Feature → Operation

```
import { DeviceManager, ScanFeature } from '@ecobridge.xyz/devicemanager';
import * as fs from 'fs/promises';

async function scanDocument() {
  const manager = new DeviceManager();

  // 1️⃣ DISCOVERY - Find scanners in your network
  await manager.discoverScanners('192.168.1.0/24');

  // 2️⃣ INSPECTION - See what's available
  const scanners = manager.getDevices({ hasFeature: 'scan' });
  console.log('Found scanners:', scanners.map(s => `${s.name} at ${s.address}`));

  // 3️⃣ SELECTION - Choose your device (explicit, no magic!)
  const device = manager.selectDevice({ address: '192.168.1.100' });

  // 4️⃣ FEATURE ACCESS - Get the capability you need
  const scanFeature = device.selectFeature<ScanFeature>('scan');
```

```

// 5th OPERATION - Do the thing!
await scanFeature.connect();
const result = await scanFeature.scan({
  source: 'flatbed',
  resolution: 300,
  colorMode: 'color',
  format: 'jpeg',
});

await fs.writeFile('scan.jpg', result.data);
console.log(`Saved: scan.jpg (${result.data.length} bytes)`);

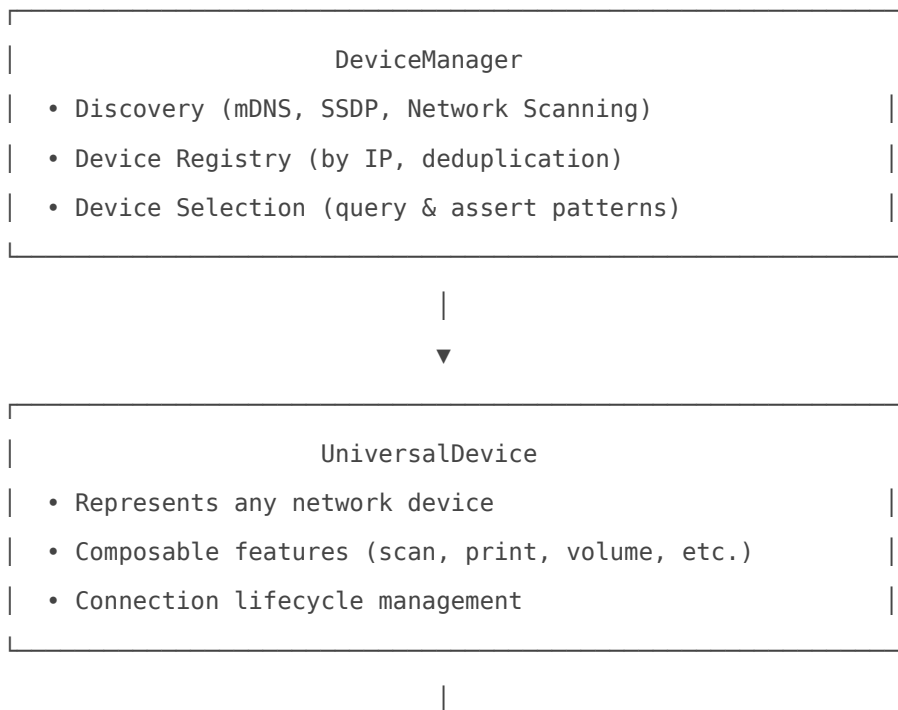
await manager.shutdown();
}

scanDocument();

```

Architecture

The library follows a clean, composable architecture:





Features			
ScanFeature	PrintFeature	PlaybackFeature	VolumeFeature
PowerFeature	SnmpFeature	LightFeature	SwitchFeature
SensorFeature	ClimateFeature	CameraFeature	...



Protocols						
eSCL	SANE	IPP	SNMP	NUT	UPnP/SOAP	Home Assistant

☐ API Reference

DeviceManager

The central orchestrator for device discovery and management.

```
const manager = new DeviceManager({
  autoDiscovery: true,      // Enable mDNS/SSDP auto-discovery
  discoveryTimeout: 10000, // Discovery timeout in ms
  enableRetry: true,       // Enable retry with exponential backoff
  maxRetries: 5,           // Maximum retry attempts
});
```

Discovery Methods

```
// Focused scanner discovery
const scanners = await manager.discoverScanners('192.168.1.0/24', {
  timeout: 3000,
  concurrency: 50,
});

// Focused printer discovery
const printers = await manager.discoverPrinters('192.168.1.0/24');
```

```
// General network scan (all device types)
const devices = await manager.scanNetwork({
  ipRange: '192.168.1.0/24',
  probeEscl: true,
  probeIpp: true,
  probeSane: true,
  probeAirplay: true,
  probeSonos: true,
  probeChromecast: true,
});

// mDNS/SSDP continuous discovery
await manager.startDiscovery();
manager.on('device:found', ({ device, featureType }) => {
  console.log(`Found: ${device.name}`);
});
await manager.stopDiscovery();
```

Device Selection

```
// Query pattern (returns array, may be empty)
const allDevices = manager.getDevices();
const scanners = manager.getDevices({ hasFeature: 'scan' });
const brotherDevices = manager.getDevices({ name: 'Brother' });

// Assert pattern (returns single device, throws if not found)
const device = manager.selectDevice({ address: '192.168.1.100' });
const scanner = manager.selectDevice({ name: 'Brother', hasFeature: 'scan' });
```

IDeviceSelector Interface

```
interface IDeviceSelector {
  id?: string; // Exact match on device ID
  address?: string; // Exact match on IP address
  name?: string; // Partial match (case-insensitive)
  model?: string; // Partial match (case-insensitive)
  manufacturer?: string; // Partial match (case-insensitive)
  hasFeature?: TFeatureType; // Must have this feature
```

```
hasFeatures?: TFeatureType[]; // Must have ALL features
hasAnyFeature?: TFeatureType[]; // Must have ANY feature
}
```

UniversalDevice

Represents any network device with composable features.

```
const device = manager.selectDevice({ address: '192.168.1.100' });

// Device properties
console.log(device.name); // "Brother MFC-J5730DW"
console.log(device.address); // "192.168.1.100"
console.log(device.manufacturer); // "Brother"
console.log(device.model); // "MFC-J5730DW"
console.log(device.status); // 'online' | 'offline' | 'busy' | 'error'

// Feature access (safe query - returns undefined)
const maybeScan = device.getFeature<ScanFeature>('scan');

// Feature access (assert - throws if not available)
const scanFeature = device.selectFeature<ScanFeature>('scan');

// Check capabilities
device.hasFeature('scan'); // true/false
device.hasFeatures(['scan', 'print']); // must have ALL
device.hasAnyFeature(['scan', 'print']); // must have ANY
device.getFeatureTypes(); // ['scan', 'print', ...]
```

Features

☐ ScanFeature

```
const scanFeature = device.selectFeature<ScanFeature>('scan');
await scanFeature.connect();

// Get capabilities
const caps = await scanFeature.getCapabilities();
```

```

// { resolutions: [100, 200, 300, 600], formats: ['jpeg', 'png', 'pdf'], ... }

// Scan a document
const result = await scanFeature.scan({
  source: 'flatbed',      // 'flatbed' | 'adf' | 'adf-duplex'
  resolution: 300,       // DPI
  colorMode: 'color',    // 'color' | 'grayscale' | 'blackwhite'
  format: 'jpeg',        // 'jpeg' | 'png' | 'pdf' | 'tiff'
  quality: 85,           // JPEG quality (1-100)
  area: { x: 0, y: 0, width: 210, height: 297 }, // mm
});

// result.data is a Buffer containing the scanned image
await fs.writeFile('scan.jpg', result.data);

```

☐ PrintFeature

```

const printFeature = device.selectFeature<PrintFeature>('print');
await printFeature.connect();

// Get printer capabilities
const caps = await printFeature.getCapabilities();

// Print a document
const job = await printFeature.print(pdfBuffer, {
  copies: 2,
  mediaSize: 'iso_a4_210x297mm',
  sides: 'two-sided-long-edge',
  quality: 'high',
  colorMode: 'color',
  jobName: 'My Document',
});

// Check job status
const status = await printFeature.getJobStatus(job.id);

```

☐ PlaybackFeature & VolumeFeature

```

const playback = device.selectFeature<PlaybackFeature>('playback');
const volume = device.selectFeature<VolumeFeature>('volume');

```

```
await playback.connect();

// Control playback
await playback.play('http://example.com/audio.mp3');
await playback.pause();
await playback.stop();
await playback.seek(120); // seconds

// Get playback status
const status = await playback.getStatus();
// { state: 'playing', position: 45, duration: 180, track: { title: '...' } }

// Control volume
await volume.setVolume(50); // 0-100
await volume.mute();
await volume.unmute();
const level = await volume.getVolume();
```

☐☐ PowerFeature (UPS)

```
const power = device.selectFeature<PowerFeature>('power');
await power.connect();

// Get UPS status
const status = await power.getStatus();
// { status: 'online', battery: { charge: 100, runtime: 1800, voltage: 13.8 }, ... }

// Get battery info
const battery = await power.getBatteryInfo();
// { charge: 95, runtime: 1500, health: 'good' }

// Run self-test
await power.runTest();
```

☐☐ Smart Home Features

```
// Light control
const light = device.selectFeature<LightFeature>('light');
await light.turnOn();
```

```

await light.setBrightness(80);
await light.setColor({ r: 255, g: 100, b: 50 });
await light.setColorTemperature(4000); // Kelvin

// Switch control
const switch_ = device.selectFeature<SwitchFeature>('switch');
await switch_.turnOn();
await switch_.turnOff();
await switch_.toggle();

// Climate control
const climate = device.selectFeature<ClimateFeature>('climate');
await climate.setTargetTemperature(22);
await climate.setMode('heat'); // 'heat' | 'cool' | 'auto' | 'off'

// Sensor reading
const sensor = device.selectFeature<SensorFeature>('sensor');
const reading = await sensor.getState();
// { temperature: 22.5, humidity: 45, battery: 85 }

```

Protocol Direct Access

For advanced use cases, you can access protocols directly:

```

import { EsclProtocol, IppProtocol, SnmpProtocol } from '@ecobridge.xyz/devicemanager';

// Direct eSCL (AirScan) access
const escl = new EsclProtocol('192.168.1.100', 80, false);
const caps = await escl.getCapabilities();
const result = await escl.scan({ source: 'flatbed', resolution: 300 });

// Direct IPP access
const ipp = new IppProtocol('ipp://192.168.1.100:631/ipp/print');
const printerAttrs = await ipp.getPrinterAttributes();

// Direct SNMP access
const snmp = new SnmpProtocol('192.168.1.100', { community: 'public' });
const sysDescr = await snmp.get('1.3.6.1.2.1.1.1.0');

```

Home Assistant Integration

```
import { HomeAssistantProtocol, HomeAssistantDiscovery } from '@ecobridge.xyz/devicemanager';

// Connect to Home Assistant
const ha = new HomeAssistantProtocol({
  host: 'homeassistant.local',
  port: 8123,
  accessToken: 'your_long_lived_access_token',
});

await ha.connect();

// Get all entities
const entities = await ha.getStates();

// Control a light
await ha.callService('light', 'turn_on', {
  entity_id: 'light.living_room',
  brightness: 200,
});

// Subscribe to state changes
ha.on('state_changed', (event) => {
  console.log(`${event.entity_id}: ${event.new_state.state}`);
});

// Auto-discover Home Assistant instances via mDNS
const discovery = new HomeAssistantDiscovery();
discovery.on('found', (instance) => {
  console.log(`Found HA at ${instance.address}:${instance.port}`);
});
await discovery.start();
```

Helper Utilities

```

import {
  withRetry,
  isValidIp,
  cidrToIps,
  getLocalSubnet,
} from '@ecobridge.xyz/devicemanager';

// Retry with exponential backoff
const result = await withRetry(
  () => someFlakeyOperation(),
  { maxRetries: 3, baseDelay: 1000, multiplier: 2 }
);

// IP utilities
isValidIp('192.168.1.1');           // true
cidrToIps('192.168.1.0/30');       // ['192.168.1.0', '192.168.1.1', ...]
getLocalSubnet();                   // '192.168.1.0/24'

```

☐ Discovery Methods

The library supports multiple discovery mechanisms:

Method	Protocol	Use Case
<code>discoverScanners()</code>	eSCL, SANE	Find network scanners
<code>discoverPrinters()</code>	IPP	Find network printers
<code>scanNetwork()</code>	All	Comprehensive subnet scan
<code>startDiscovery()</code>	mDNS, SSDP	Continuous auto-discovery

mDNS Service Types

```

import { SERVICE_TYPES } from '@ecobridge.xyz/devicemanager';

// Available: escl, ipp, ipp-tls, airplay, raop,
// googlecast, sonos, sane, http, https, printer

```

SSDP Service Types

```
import { SSDP_SERVICE_TYPES } from '@ecobridge.xyz/devicemanager';

// Available: all, rootdevice, mediaRenderer, mediaServer,
// contentDirectory, avtransport, renderingControl,
// connectionManager, zonePlayer
```

☐ Feature Types

```
type TFeatureType =
  | 'scan'          // Document scanning
  | 'print'         // Document printing
  | 'fax'           // Fax send/receive
  | 'copy'          // Copy (scan + print)
  | 'playback'     // Media playback
  | 'volume'       // Volume control
  | 'power'        // Power/UPS status
  | 'snmp'         // SNMP queries
  | 'dlna-render' // DLNA renderer
  | 'dlna-serve'  // DLNA server
  | 'light'        // Smart lights
  | 'climate'     // HVAC/thermostats
  | 'sensor'       // Sensors
  | 'camera'       // Cameras
  | 'cover'        // Blinds, garage doors
  | 'switch'       // Smart switches
  | 'lock'         // Smart locks
  | 'fan'          // Fans
;
```

☐ Advanced Usage

Custom Device Creation

Use the factory functions for creating devices with specific features:

```
import { createScanner, createPrinter, createSpeaker } from '@ecobridge.xyz/devicemanager';

// Create a scanner device manually
const scanner = createScanner({
  id: 'my-scanner',
  name: 'Office Scanner',
  address: '192.168.1.50',
  port: 80,
  protocol: 'escl',
  txtRecords: {},
});

// Create a printer device
const printer = createPrinter({
  id: 'my-printer',
  name: 'Office Printer',
  address: '192.168.1.51',
  port: 631,
  txtRecords: { rp: '/ipp/print' },
});

// Create a Sonos speaker
const speaker = createSpeaker({
  id: 'living-room-sonos',
  name: 'Living Room',
  address: '192.168.1.52',
  port: 1400,
  protocol: 'sonos',
});
```

Smart Home Factory Functions

```
import {
  createSmartLight,
  createSmartSwitch,
  createSmartSensor,
  createSmartClimate,
```

```
    createSmartCover,  
    createSmartLock,  
    createSmartFan,  
    createSmartCamera,  
  } from '@ecobridge.xyz/devicemanager';  
  
// Create devices with Home Assistant integration  
const light = createSmartLight({  
  id: 'living-room-light',  
  name: 'Living Room Light',  
  address: 'homeassistant.local',  
  port: 8123,  
  entityId: 'light.living_room',  
  protocol: 'home-assistant',  
  protocolClient: haClient, // Your HomeAssistantProtocol instance  
  capabilities: {  
    supportsBrightness: true,  
    supportsColorTemp: true,  
    supportsRgb: true,  
  },  
});
```

Event Handling

```
const manager = new DeviceManager();  
  
// Discovery events  
manager.on('device:found', ({ device, featureType }) => {  
  console.log(`Found ${device.name} with ${featureType} capability`);  
});  
  
manager.on('device:lost', (address) => {  
  console.log(`Device at ${address} went offline`);  
});  
  
// Network scan progress  
manager.on('network:progress', (progress) => {  
  console.log(`Scanning: ${progress.percentage}% - Found ${progress.devicesFound} devices`);  
});
```

```
});

// Device events
const device = manager.selectDevice({ address: '192.168.1.100' });
device.on('status:changed', ({ oldStatus, newStatus }) => {
  console.log(`Status: ${oldStatus} → ${newStatus}`);
});
device.on('feature:connected', (featureType) => {
  console.log(`Feature ${featureType} connected`);
});
```

Error Handling

The library uses a fail-fast approach with clear error messages:

```
try {
  // Throws if no device matches
  const device = manager.selectDevice({ address: '192.168.1.999' });
} catch (err) {
  // "No device found matching: {"address":"192.168.1.999"}"
}

try {
  // Throws if device doesn't have the feature
  const printFeature = device.selectFeature<PrintFeature>('print');
} catch (err) {
  // "Device 'Brother Scanner' does not have feature 'print'"
}

// Safe alternatives that don't throw
const devices = manager.getDevices({ address: '192.168.1.999' }); // []
const maybePrint = device.getFeature<PrintFeature>('print'); // undefined
```

📦 Requirements

- **Node.js** 18+ (native `fetch` support required)
- **TypeScript** 5.0+ (recommended)

- **Network access** to target devices

Credits

Built with ♥ using:

- [bonjour-service](#) - mDNS discovery
- [node-ssdp](#) - SSDP/UPnP discovery
- [net-snmp](#) - SNMP protocol
- [ipp](#) - IPP printing protocol
- [sonos](#) - Sonos control
- [castv2-client](#) - Chromecast protocol

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the [LICENSE](#) file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #6

Created 2026-03-28 10:49:15 UTC by foss.global Team

Updated 2026-03-28 12:14:40 UTC by foss.global Team