@ecobridge.xyz/devicemanager

Documentation for @ecobridge.xyz/devicemanager

- readme.md for @ecobridge.xyz/devicemanager
- changelog.md for @ecobridge.xyz/devicemanager

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

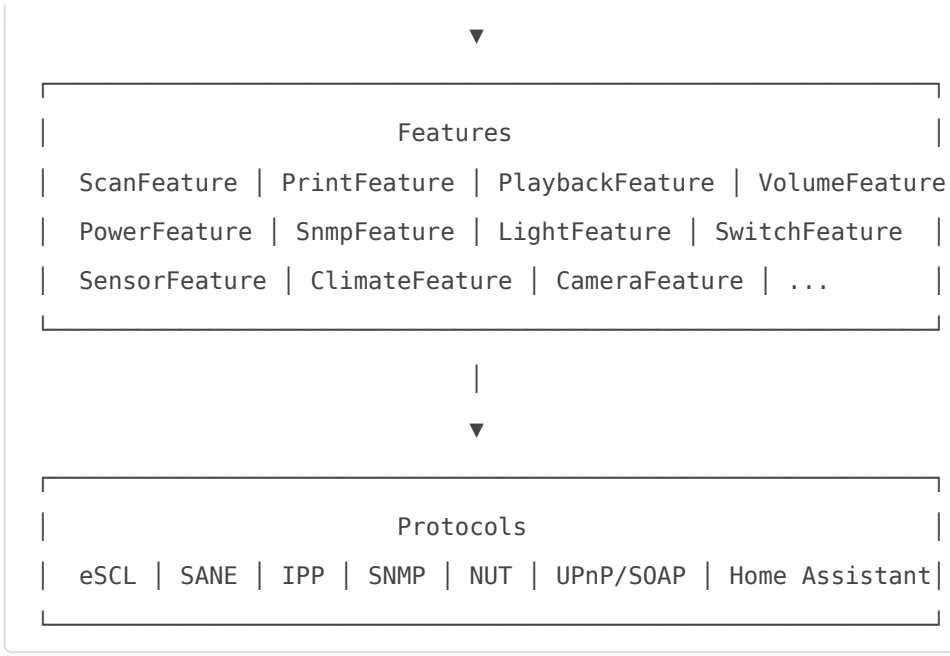
```
import { DeviceManager, ScanFeature } from '@ecobridge.xyz/devicemanager';
import * as fs from 'fs/promises';

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

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

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

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

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

```
  // 5□□ 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:

```
┌─────────────────────────────────────────────────────────┐
│                      DeviceManager                      │
│  • Discovery (mDNS, SSDP, Network Scanning)             │
│  • Device Registry (by IP, deduplication)               │
│  • Device Selection (query & assert patterns)           │
└─────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────┐
│                    UniversalDevice                      │
│  • Represents any network device                        │
│  • Composable features (scan, print, volume, etc.)      │
│  • Connection lifecycle management                      │
└─────────────────────────────────────────────────────────┘
                            │
```



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
`discoverScanners()`	eSCL, SANE	Find network scanners
`discoverPrinters()`	IPP	Find network printers
`scanNetwork()`	All	Comprehensive subnet scan
`startDiscovery()`	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.

changelog.md for @ecobridge.xyz/devicemanager

2026-01-13 - 3.1.0 - feat(print)

use IPP smartPrint and normalize IPP capabilities and job mapping

- Use IppProtocol.smartPrint for automatic format detection/conversion when submitting print jobs.
- Normalize and map IIppJob -> IPrintJob via mapIppJobToInternal, collapsing extended IPP job states into internal states.
- Parse IIppPrinterCapabilities fields (mediaSizeSupported, mediaTypeSupported, sidesSupported, printQualitySupported, copiesSupported) and derive supportsDuplex from sidesSupported and maxCopies from copiesSupported range with a fallback.
- Map numeric IPP printQuality values (3,4,5) to internal quality strings (draft, normal, high).
- Switched calls to getPrinterAttributes/getJobAttributes and adjusted job listing to map returned IIppJob objects.
- Export new IPP types from protocols index: IIppPrinterCapabilities, IIppJob, IIppPrintOptions.

2026-01-12 - 3.0.2 - fix(devicemanager)

no changes detected - nothing to commit

- git diff indicates no modifications, additions, or deletions
- no files were changed in the provided diff

2026-01-12 - 3.0.1 - fix(release)

add npm registries to release config and expand documentation for UniversalDevice architecture and smart-home features

- npmextra.json: add "registries" to release configuration to publish to both Verdaccio (https://verdaccio.lossless.digital) and the npm registry
- readme.hints.md: rewritten/expanded implementation notes to describe the UniversalDevice architecture, composable features (including smart-home types like light, switch, sensor, climate, cover, lock, fan, camera), protocols, discovery, factories, interfaces, and testing guidance
- readme.md: add factory usage examples including smart-home factory functions, update txtRecords usage (rp) in examples, and small copy/emoji edits

2026-01-10 - 3.0.0 - BREAKING CHANGE(devicemanager)

migrate tests to new UniversalDevice/feature-based API, add device factories, SNMP protocol/feature and IP helper utilities

- Replace protocol-specific device classes (Scanner, Printer) with UniversalDevice and feature objects (ScanFeature, PrintFeature, PlaybackFeature, VolumeFeature, PowerFeature, SnmpFeature)
- Add device factory functions: createScanner, createPrinter, createSpeaker, createUpsDevice
- Add DeviceManager.getDevices selector and updated selectDevice behavior (throws when no match)
- Expose SnmpProtocol and other protocol implementations
- Introduce IP helper utilities: isValidIp, cidrToIps, getLocalSubnet
- Update tests and logging to use feature-based APIs and factories (selectFeature/getFeature, hasFeature, featureCount)

2026-01-09 - 2.3.1 - fix(readme)

update README to comprehensive, TypeScript-first documentation covering installation, quick start, examples, API usage, events, error handling, requirements, credits, and legal/company information

- Rewrote readme.md with ~590 additional lines to provide a full usage guide and examples
- Added installation instructions for pnpm, npm, and yarn and badges for npm and license
- Documented OOP usage pattern (Discovery → Selection → Feature → Operation), event handling, and error handling examples
- Clarified requirements (Node.js 18+, TypeScript 5.0+), credits, license, trademark and company contact information
- Docs-only change — no code or API modifications

2026-01-09 - 2.3.0 - feat(devicemanager)

add selector-based device APIs, selectFeature helper, convenience discovery methods, and ESCL scan completion fallback

- Introduce IDeviceSelector and add selector support to getDevices(selector) to filter devices by id,address,name,model,manufacturer and feature capabilities.
- Add selectDevice(selector) which returns exactly one device (throws if none) and logs a warning if multiple matches are returned without a unique identifier.
- Deprecate getDevice(id) and getDeviceByAddress(address) in favor of selector-based retrieval methods.
- Add private matchesSelector(...) implementing exact identity checks (id,address), case-insensitive partial attribute matching (name, model, manufacturer), and feature availability checks (hasFeature, hasFeatures, hasAnyFeature).
- Add selectFeature(type) on devices to provide fail-fast access to a required feature (throws if missing).
- Add discoverScanners(subnet, options) and discoverPrinters(subnet, options) convenience methods that run targeted network scans and return discovered scanners or printers respectively.
- Improve ESCL protocol waitForScanComplete to attempt a direct download first (which triggers/blocks on many scanners) and fall back to polling if direct download fails or returns empty data.

2026-01-09 - 2.2.0 - feat(smarthome)

add smart home features and Home Assistant integration (WebSocket protocol, discovery, factories, interfaces)

- Add concrete smart home feature implementations: light, climate, sensor, switch, cover, lock, fan, camera.
- Introduce Home Assistant WebSocket protocol handler (protocol.homeassistant) and Home Assistant discovery via mDNS (discovery.classes.homeassistant).
- Add generic smart home interfaces and Home Assistant-specific interfaces (smarthome.interfaces, homeassistant.interfaces) and export them.
- Add smart home factories to create devices for discovered/declared smart home entities and export factory helpers.
- Update plugins to include WebSocket (ws) and add ws dependency and @types/ws in package.json.

2026-01-09 - 2.1.0 - feat(devicemanager)

prefer higher-priority discovery source when resolving device names and track per-device name source

- Add TNameSource type and NAME_SOURCE_PRIORITY to rank name sources (generic, manual, airplay, chromecast, mdns, dlna, sonos).
- Replace chooseBestName with shouldUpdateName that validates 'real' names and uses source priority when deciding to update a device name.
- Add nameSourceByIp map to track which discovery source provided the current name and persist updates during registration.
- Register devices with an explicit nameSource (e.g. 'mdns', 'dlna', 'sonos', 'manual') and map speaker protocols to appropriate name sources.
- Ensure manual additions use 'manual' source and non-real names default to 'generic'.
- Clear nameSourceByIp entries when devices are removed/disconnected and on shutdown.

2026-01-09 - 2.0.0 - BREAKING CHANGE(core)

rework core device architecture: consolidate protocols into a protocols/ module, introduce UniversalDevice + factories, and remove many legacy device-specific classes (breaking API changes)

- Consolidated protocol implementations into ts/protocols and added protocols/index.ts for unified exports.

- Added device factory layer at ts/factories/index.ts to create UniversalDevice instances with appropriate features.
- Introduced protocols/protocol.upssnmp.ts (UPS SNMP handler) and other protocol reorganizations.
- Removed legacy concrete device classes and related files (Device abstract, Scanner, Printer, SnmpDevice, UpsDevice, DlnaRenderer/Server, Speaker and Sonos/AirPlay/Chromecast implementations).
- Updated top-level ts/index.ts exports to prefer UniversalDevice, factories and the new protocols module.
- Updated feature and discovery modules to import protocols from the new protocols index (import path changes).
- BREAKING: Consumers must update imports and device creation flows to use the new factories/UniversalDevice and protocols exports instead of the removed legacy classes.

2026-01-09 - 1.1.0 - feat(devicemanager)

Introduce a UniversalDevice architecture with composable Feature system; add extensive new device/protocol support and discovery/refactors

- Add UniversalDevice class and Feature abstraction with concrete features: scan, print, playback, volume, power, snmp, dlna-render/serve.
- Add SSDP discovery and DLNA implementations (renderer + server) and integrate SSDP into DeviceManager.
- Add speaker subsystem and concrete speaker implementations: Sonos, AirPlay, Chromecast, plus generic Speaker API and Volume/Playback features.
- Add SNMP feature and SNMP device handling plus UPS support (NUT and UPS SNMP handlers and UpsDevice).
- Refactor protocol implementations: move/replace scanner/printer protocol code into protocols/ (eSCL, SANE, IPP) and update network scanner to probe additional ports (AirPlay, Sonos, Chromecast) and device types.
- Update exports (ts/index.ts) to expose new modules, types and helpers; update plugins import handling (node-ssdp default export compatibility).
- Add developer docs readme.hints.md describing new architecture and feature APIs, and various helper fixes (iprange/os import, typed socket handlers).

2026-01-09 - 1.0.1 - initial

Initial project release.

- Project initialized (initial commit).

- Duplicate initial commits consolidated into this release.