

RasProLink

Benutzer- und Inbetriebnahmehandbuch

Ein Windows-Dienst, fünf industrielle Protokoll-Engines

Verwandeln Sie jeden Windows-PC in eine echte EtherCAT-, EtherNet/IP- oder PROFINET-Station — ohne Zusatzhardware.

RAS Automation — RasProLink v1.0

September 2026

Inhaltsverzeichnis

Kapitel 1 Einführung.....	3
Kapitel 2 Systemanforderungen.....	5
Kapitel 3 Installation.....	6
Kapitel 4 Die Weboberfläche.....	7
Kapitel 5 Datenzuordnung und Tags.....	11
Kapitel 6 EtherCAT Master.....	14
Kapitel 7 EtherNet/IP Scanner.....	19
Kapitel 8 EtherNet/IP Adapter.....	22
Kapitel 9 PROFINET Controller.....	24
Kapitel 10 PROFINET Device.....	36
Kapitel 11 .NET-Clientbibliothek.....	39
Kapitel 12 Lizenzierung.....	44
Kapitel 13 Leistung und Echtzeitverhalten.....	46
Kapitel 14 Allgemeine Fehlerbehebung.....	47
Anhang A HTTP-API-Referenz.....	48
Anhang B Konfigurationsdateien.....	53
Anhang C Glossar.....	55

Kapitel 1

Einführung

1.1 Für wen dieses Handbuch gedacht ist

Dieses Handbuch richtet sich an Automatisierungsingenieure, die einen PC benötigen, der echt mit einem industriellen Feldbus spricht — als Master, Controller, Scanner, Adapter oder Device — sowie an Entwickler, die diese Daten aus einer .NET-Anwendung, einem Skript oder einem beliebigen HTTP-Client nutzen.

Grundkenntnisse der Kernbegriffe Ihres gewählten Protokolls (Station, Slave/Device, zyklische Daten) werden vorausgesetzt; Vorerfahrung mit RAS-Automation-Produkten ist nicht erforderlich.

1.2 Was RasProLink ist

RasProLink ist ein einziger Windows-Dienst, der Ihren PC in eine echte industrielle Kommunikationsstation auf fünf Protokollen verwandelt: EtherCAT (als Master), EtherNet/IP (als Scanner/Originator oder als Adapter/Target) und PROFINET (als IO-Controller oder als IO-Device). Die aktive Protokoll-Engine wählen Sie über eine Weboberfläche — keine zusätzliche Hardware, keine Gateway-Box und kein kostenpflichtiger Drittanbieter-Stack erforderlich.

Engine	Protokoll	Rolle
EtherCAT Master	EtherCAT	Master (Ihr PC steuert den Slave-Ring)
EtherNet/IP Scanner	EtherNet/IP (CIP)	Originator — öffnet Klasse-1-Verbindungen zu Adaptern
EtherNet/IP Adapter	EtherNet/IP (CIP)	Target — akzeptiert Forward Open von einem Scanner/PLC
PROFINET Controller	PROFINET	IO-Controller — die Master-Station, die normalerweise eine SPS ist
PROFINET Device	PROFINET	IO-Device — Slave gegenüber einer echten SPS, durch GSDML beschrieben

Es läuft immer nur eine Engine gleichzeitig; das Wechseln der Engine über die Weboberfläche startet den zugrunde liegenden Dienst innerhalb von Sekunden im neuen Protokoll neu. Jede Engine behält ihre eigene Konfiguration, sodass Sie beim Zurückwechseln genau dort weitermachen, wo Sie aufgehört haben.

1.3 Warum RasProLink

- Keine Hardware-Feldbuskarte, kein Dongle, keine Gateway-Box — ein Standard-Ethernet-Anschluss genügt.
- Keine teuren Konnektivitätsbibliotheken von Drittanbietern — der Protokoll-Stack ist integriert und wird direkt angesteuert.
- Multiprotokoll von Grund auf — eine Installation, fünf Engines, wechseln Sie, sobald ein Projekt einen anderen Feldbus benötigt.
- Drag-and-drop-Inbetriebnahme in Echtzeit in einer gewöhnlichen Windows-Umgebung — echtes Master-/Controller-/Device-Verhalten, kein Simulator.

- .NET-nativ — jede Software, die Sie bereits unter .NET betreiben, wird zum Gehirn hinter dem Controller/Device, vollständig unter Ihrer eigenen Kontrolle.

Tipp: Echtzeit-Hinweis: Bei allen fünf Engines läuft die zyklische Austauschschleife auf einem dedizierten Thread mit hoher Priorität, mit Windows-Multimedia-Timer-Auflösung und CPU-Kernaffinität, wobei Heap-Zuweisungen pro Zyklus eliminiert wurden. Auf einem typischen Windows-PC hält dies den Zyklus-Jitter im Sub-Millisekundenbereich (bei einem 4-ms-EtherCAT-Zyklus wurden Werte um 0,3 ms gemessen) — siehe §13.4.

1.4 Nutzungsmöglichkeiten

Sobald Geräte verbunden sind, gibt es zwei Möglichkeiten, die Daten zu nutzen:

- Weboberfläche — Geräte hinzufügen, Status beobachten, rohe IO-Bytes lesen/schreiben, Lizenz verwalten und das Dienstprotokoll verfolgen. Gedacht für Inbetriebnahme und Diagnose, nicht für den Produktionsbetrieb.
- .NET-Clientbibliothek (RasProLink.Client) — eine kleine, abhängigkeitsfreie DLL, die Ihre eigene Anwendung referenziert, um Gerätestatus und IO im Code zu lesen, Ausgänge zu schreiben und Geräte/Tags programmgesteuert zu verwalten. Darauf wird eine echte Anwendung aufgebaut — siehe Kapitel 11.

Kapitel 2

Systemanforderungen

- Windows 10/11 oder Windows Server 2016 oder neuer, 64-Bit.
- Ein freier, kabelgebundener Ethernet-Adapter im selben physischen/logischen Netzwerk wie Ihre Feldgeräte. WLAN eignet sich für die feldseitige Anbindung keiner der fünf Engines — zyklischer Echtzeitverkehr benötigt ein geschwitchtes, kabelgebundenes Segment.
- Nur EtherCAT Master: ein Netzwerkadapter mit einem vom integrierten Rohframe-Treiber unterstützten Chipsatz (die meisten Intel- und Realtek-Adapter werden unterstützt).
- PROFINET Controller und PROFINET Device: Npcap (<https://npcap.com>) für Rohframe-Zugriff — im Installationspaket enthalten oder vorab installierbar.
- Es muss keine separate .NET-Runtime installiert werden — der Dienst wird vollständig eigenständig veröffentlicht.
- Administratorrechte zur Installation des Windows-Dienstes und des Npcap-/Rohframe-Treibers.

Warnung: Stellen Sie die feldseitige Anbindung keiner Engine hinter eine WLAN-Strecke, ein VPN oder einen stark ausgelasteten Shared-Switch: EtherCAT- und PROFINET-Zyklusrahmen sowie EtherNet/IP-CIP-Verbindungen sind zeitkritisch — zusätzliche Latenz oder Jitter führt direkt zu Verbindungsabbrüchen oder Watchdog-Timeouts.

Kapitel 3

Installation

3.1 Installationsschritte

1. Kopieren Sie das Paket auf den Ziel-PC.
2. Falls ein npcap-*.exe-Installer enthalten ist, führen Sie ihn zuerst aus (oder installieren Sie Npcap vorab von npcap.com). Für die EtherNet/IP-Engines nicht erforderlich.
3. Klicken Sie mit der rechten Maustaste auf install.bat und wählen Sie Als Administrator ausführen.
4. Der Dienst wird installiert, startet automatisch und wird so eingestellt, dass er mit Windows startet.
5. Öffnen Sie <http://localhost:8080> in einem Browser auf dem PC, um zu bestätigen, dass die Weboberfläche antwortet.

3.2 Wie der Dienst läuft

RasProLink läuft als Windows-Dienst ohne Fenster oder Taskleistensymbol unter dem Namen RasProLink.Service. Er startet, bevor sich jemand anmeldet, und läuft über Benutzersitzungen hinweg weiter. Verwenden Sie die Windows-Dienstekonsole (services.msc), um den Status zu prüfen oder ihn bei der Fehlersuche manuell zu stoppen/starten.

3.3 Fernzugriff auf die Weboberfläche

Standardmäßig hört die Weboberfläche auf allen lokalen Adressen an Port 8080, sodass <http://<pc-ip>:8080> sie von einem anderen Rechner im Netzwerk erreicht — nützlich, wenn der Industrie-PC an anderer Stelle montiert ist und Sie ihn von einem Laptop aus in Betrieb nehmen. Ist die Windows-Firewall aktiv, lassen Sie beim ersten Zugriff von außen den eingehenden TCP-8080-Verkehr zu.

3.4 Aktualisierung und Deinstallation

Für ein Update führen Sie die install.bat der neuen Version über der bestehenden Installation aus: Sie stoppt den laufenden Dienst, ersetzt ausführbare Datei und Dateien und startet ihn neu. Konfigurationsdateien bleiben bei einem Update unberührt, sodass Gerätelisten und Netzwerkeinstellungen erhalten bleiben. Zum Entfernen führen Sie uninstall.bat aus; der Konfigurationsordner bleibt standardmäßig erhalten (falls Sie später neu installieren) — übergeben Sie -RemoveConfig, um ihn und die Protokolle ebenfalls zu löschen.

Kapitel 4

Die Weboberfläche

Das Layout der Weboberfläche ist bei allen fünf Engines identisch: Nur der Name der aktiven Engine und ihre protokollspezifischen Karten ändern sich. Die folgenden Screenshots stammen stellvertretend von der PROFINET-Controller-Engine; EtherCAT Master, EtherNet/IP Scanner/Adapter und PROFINET Device verwenden denselben Rahmen mit ihren eigenen Geräte-/Konfigurationskarten (behandelt in ihren eigenen Kapiteln, §6–§10).

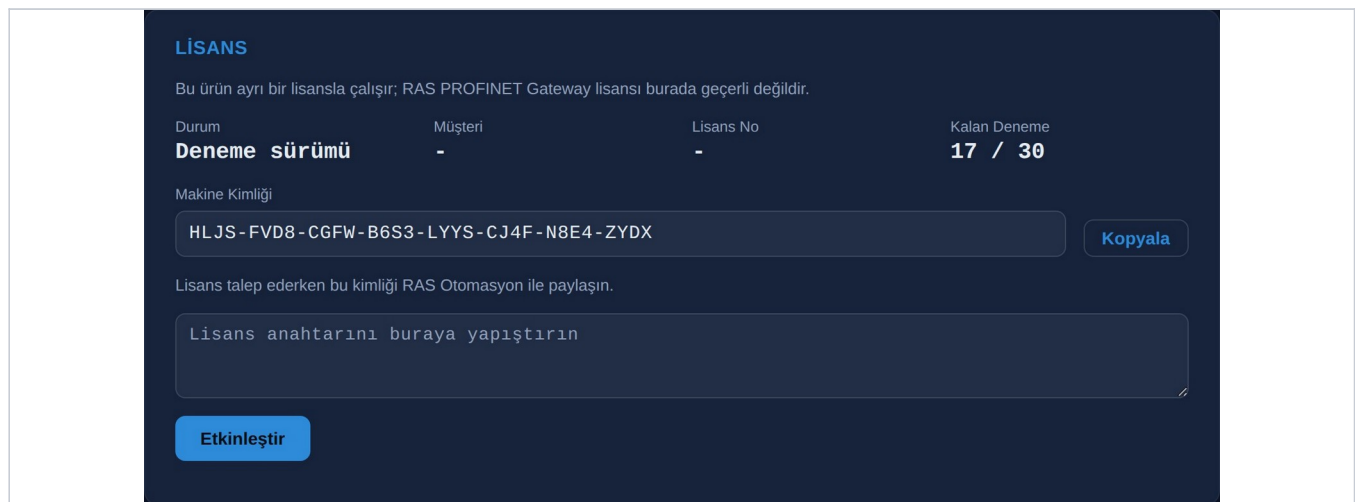
4.1 Kopfleiste und Sprachauswahl



Kopfleiste — Name der aktiven Engine, Sprachauswahl, Live-Verbindungsanzeige (Beispiel: PROFINET Controller)

Die Kopfleiste zeigt den Produktnamen und die aktive Engine, eine Sprachauswahl sowie eine Verbindungsstatusanzeige, die grün wird, sobald der Browser den Dienst erreichen kann. Die gesamte Oberfläche wird bei einem Sprachwechsel sofort neu gerendert; die Seite wird nicht neu geladen.

4.2 Lizenzkarte



Lizenzkarte während der Testphase

Lizenzkarte nach Aktivierung eines Schlüssels

Die Karte zeigt Status, Kunde, Lizenznummer und verbleibende Testtage; die bei einer Lizenzanfrage an RAS Automation zu übermittelnde Maschinen-ID; sowie ein Feld zum Einfügen eines Lizenzschlüssels. Vollständiger Ablauf in Kapitel 12.

4.3 Auswahl des Netzwerkadapters

Auswahl des Netzwerkadapters mit optionaler manueller MAC-Überschreibung

Auf einem PC mit mehr als einem Netzwerkadapter wählen Sie hier den physisch mit den Feldgeräten verbundenen aus. Diese Auswahl ist für jede Engine gleich, da jede Engine zu einem Zeitpunkt über genau einen Adapter mit dem Feld spricht.

4.4 Wechseln der aktiven Engine

Die Kopfleiste (§4.1) jeder Engine trägt einen dauerhaften Link zu einer gemeinsamen Protokollauswahl — zum Beispiel Protokoll: PROFINET Controller ⇄ auf den Seiten von PROFINET Controller, Protokoll: EtherCAT Master ⇄ auf den Seiten von EtherCAT Master usw. Unabhängig davon, welche Engine gerade aktiv ist, öffnet dieser Link immer dieselbe, unter /engine.html bereitgestellte Seite — und ist der einzige Weg, um festzulegen, welche der fünf Engines der Dienst ausführt.

Die Auswahlseite

Die Auswahlseite zeigt eine Karte pro Engine — EtherCAT Master, EtherNet/IP Scanner, EtherNet/IP Adapter, PROFINET Controller, PROFINET Device. Die Karte der gerade aktiven Engine trägt ein Abzeichen AKTİF (aktiv) und einen deaktivierten Button Şu an aktif (derzeit aktiv); jede andere Karte zeigt entweder ein Abzeichen Kullanima hazır (einsatzbereit) oder, bei einer noch nicht fertigen Engine, ein Abzeichen Geliştirme aşamasında (in Entwicklung) mit einem kurzen Warnhinweis — sowie einen Button Bu protokolü seç (dieses Protokoll

wählen). Zum Zeitpunkt dieser Ausgabe sind alle fünf Engines einsatzbereit, sodass in der Praxis jede nicht aktive Karte derzeit das Bereit-Abzeichen zeigt; das Entwicklungs-Abzeichen ist in der Oberfläche für eine künftige Engine in diesem Zustand bereits vorgesehen.

Hinweis: Anders als der Rest der Oberfläche (§4.1) wird diese Seite von der Sprachauswahl nicht beeinflusst — ihre Beschriftungen bleiben Türkisch, unabhängig davon, welche Sprache auf den Engine-Seiten, von denen Sie kommen, aktiv ist. Die oben zitierten Beschriftungen (AKTİF, Kullanima hazır, Bu protokolü seç usw.) sind der exakte Text, den Sie in jeder Sprachausgabe von RasProLink sehen.

Der Wechsel

Ein Klick auf den Button einer Karte fordert eine Bestätigung an — mit einer stärkeren Warnung, wenn die Ziel-Engine noch nicht einsatzbereit ist — und sendet dann den Schlüssel der gewählten Engine (zum Beispiel `profinet-controller` oder `ethernetip-scanner`) per POST `/api/engine`. Der Dienst schreibt diese Wahl in `Config\enginesettings.json` und beendet sich etwa drei Sekunden später bewusst selbst. Als Windows-Dienst installiert (Kapitel 3), ist RasProLink so konfiguriert, dass er sich nach jedem Beenden — einschließlich dieses sauberen Beendens — automatisch neu startet; die Dienstverwaltung (SCM) bringt den Prozess daher innerhalb weniger Sekunden wieder hoch. Beim neuen Start wird `Config\enginesettings.json` erneut gelesen, und der Dienst läuft anschließend mit der neu gewählten Engine.

Während der Dienst nicht erreichbar ist, ruft die Auswahlseite etwa zwei Minuten lang alle drei Sekunden GET `/api/host/info` ab, bis das `activeEngine` im Antwort-JSON mit der gerade gewählten Engine übereinstimmt — dann leitet sie den Browser an `/index.html` weiter, das nun die Oberfläche der neuen Engine liefert. Stimmt nach zwei Minuten nichts überein, stoppt die Seite das Abfragen und bittet Sie, statt endlosem Weiterversuchen die Seite manuell neu zu laden.

Hinweis: Ein Wechsel ist ein echter Prozessneustart, keine nahtlose Übergabe: Jede andere Browser-Registerkarte oder jeder andere Client, der noch auf die Seiten oder die API der vorherigen Engine zeigt, verliert die Verbindung in dem Moment, in dem sich der Dienst beendet, und die zyklischen Daten stoppen, bis die neue Engine vollständig gestartet ist. Verlassen Sie sich auf die automatische Weiterleitung der Auswahlseite (oder die Verbindungsanzeige aus §4.1), um zu bestätigen, dass die neue Engine läuft, statt eine alte Registerkarte weiter zu benutzen. Wird RasProLink interaktiv (`dotnet run`) statt als installierter Dienst ausgeführt, findet der oben beschriebene automatische Neustart nicht statt — der Prozess beendet sich einfach und muss von Hand neu gestartet werden.

Konfiguration bleibt pro Engine erhalten

Jede Engine hält ihre eigenen Einstellungen in ihrer eigenen Datei bzw. ihren eigenen Dateien unter `Config\` — nach derselben Konvention, die in diesem Handbuch durchgängig verwendet wird (§9.7 sowie die entsprechenden Abschnitte in den Kapiteln 6–10). Ein Engine-Wechsel schreibt ausschließlich in `Config\enginesettings.json` — der Dienst baut den Konfigurationsspeicher nur für die jeweils aktiv werdende Engine auf und öffnet oder verändert keine Datei einer anderen Engine. Anhand des Startup-Codes des Dienstes bestätigt: Wechselt man von einer Engine weg und später wieder zu ihr zurück, bleiben deren Geräteliste, Ziele, Tags und übrige Einstellungen exakt so erhalten, wie sie verlassen wurden.

Engine	Eigene Datei(en) unter Config\
PROFINET Controller	<code>controllersettings.json</code> , <code>devices.json</code> , <code>tags.json</code> *
PROFINET Device	<code>deviceconfig.json</code> , <code>tags.json</code> *
EtherNet/IP Scanner	<code>ethernetip-scanner-targets.json</code> , <code>ethernetip-scanner-tags.json</code>
EtherNet/IP Adapter	<code>ethernetip-adapter-config.json</code> , <code>ethernetip-adapter-tags.json</code>
EtherCAT Master	<code>ethercat-master-config.json</code> , <code>ethercat-master-tags.json</code>

Hinweis: * Eine im aktuellen Build bestätigte Ausnahme: PROFINET Controller und PROFINET Device lesen und schreiben für ihre Tag-Listen dieselbe Datei Config\tags.json. Anders als ihre übrigen, vollständig getrennten Einstellungen erhält daher beim Wechsel allein zwischen diesen beiden Engines nicht jede eine eigene, unabhängige Tag-Liste. Die Lizenzkarte (§4.2) ist davon in keiner Richtung betroffen — die Lizenz selbst liegt in einer einzigen, von allen fünf Engines gemeinsam genutzten Config\lisans.json, sodass ein Engine-Wechsel nie zu einer erneuten Aktivierung auffordert.

Kapitel 5

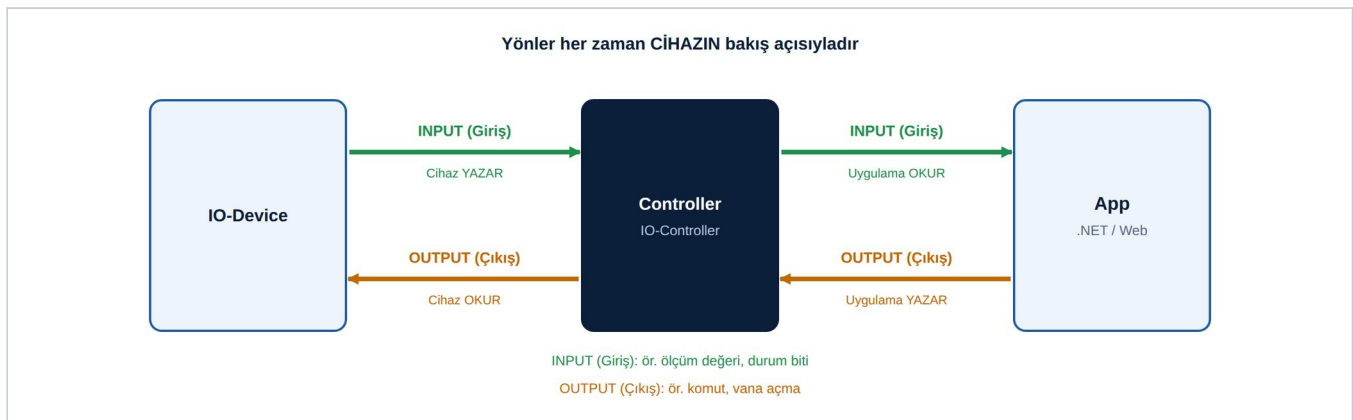
Datenzuordnung und Tags

5.1 Slots und Submodule

Jede Engine organisiert die Prozessdaten eines verbundenen Geräts auf dieselbe Weise: Ein Gerät ist ein Baum aus Slots, jeder Slot enthält ein oder mehrere Submodule, und jedes Submodul trägt seine eigene Eingangs- und/oder Ausgangsbytelänge. Dies gilt unabhängig davon, ob das zugrunde liegende Konzept ein PROFINET-Slot/Subslot, ein EtherCAT-PDO-gemappter Eintrag oder ein EtherNet/IP-Assembly ist — RasProLink normalisiert alle drei überall in der Weboberfläche, im .NET-Client und in der HTTP-API auf dasselbe Slot-/Subslot-Modell.

5.2 Ein-/Ausgangsrichtung

Die Richtung wird immer aus Sicht des verbundenen Geräts benannt: Eingang sind Daten, die das Feldgerät erzeugt und Ihre Anwendung liest (ein Sensorwert, ein Statuswort); Ausgang sind Daten, die Ihre Anwendung erzeugt und das Feldgerät verarbeitet (ein Sollwert, ein Befehl, eine Spule). Dies ist das Gegenteil davon, wie ein SPS-Programm seine eigene I/O benennt, und es ist eine einmalige Regel, keine engine-spezifische.



Die Richtung wird immer aus Sicht des GERÄTS benannt — das Gegenteil der Regel eines SPS-Programms, einmal zu prüfen wert

Die Bytes eines Ausgangssubmoduls werden von Ihrer Anwendung (oder, je nach Engine, der feldseitigen SPS/dem Scanner) geschrieben und in der nächsten Zyklusrunde von der verbundenen Seite verarbeitet.

5.3 Tags

Etiketler

Ham veriye (yukarıdaki Canlı IO) ek olarak, bayt/bit konumuna bir AD ve TİP verin — okuma/yazma bundan sonra bu adla yapılabilir. XDB ile eklenen cihazlarda da etiket tanımlanabilir.

AD	SLOT	YÖN	TİP	KONUM	DEĞER	
Sensor_Giris1	2 / 1	Giriş	bool	@0.0	false	<button>Sil</button>
Motor_Calistir	3 / 1	Çıkış	bool	@0.0	true	<button>Yaz</button> <button>Sil</button>

Etiket adı

1/0x0001 — Arayüz (in 0B / out 0B) ▼

Giriş (cihazdan) ▼

Int16 ▼

Ofset
Bit
Uzunluk
Birim

Ekle / Güncelle

Tags-Karte — benennen Sie einen Byte-/Bit-Bereich, damit er nach Namen gelesen oder geschrieben werden kann (Beispiel für PROFINET Controller gezeigt)

Ein Tag ist eine Schicht über den rohen zyklischen Daten — er kopiert sie niemals. Geben Sie einer Byte-/Bit-Position innerhalb eines Slots/Subslots einen Namen und einen Typ; von da an können Sie (oder Ihr Code) sie unter diesem Namen statt in rohem Hex lesen/schreiben. Tags können für jedes Gerät auf jeder Engine definiert werden, einschließlich automatisch hinzugefügter (via .xdb, GSDML oder ESI-Abgleich) — die Modulstruktur liefert nur das Slot-/Subslot-Layout; die Benennung fügen Sie immer selbst hinzu.

Typ	Größe	Hinweise
Bool	1 Bit	Bitnummer 0–7 innerhalb des Bytes; andere Bits bleiben unberührt.
Byte / SByte	1 Byte	Vorzeichenlos/vorzeichenbehaftet, 8 Bit.
Int16 / UInt16	2 Byte	Big-Endian, wie alle drei Feldbusse auf der Leitung.
Int32 / UInt32	4 Byte	Vorzeichenbehaftet/-los, 32 Bit.
Int64 / UInt64	8 Byte	Lesen Sie für den genauen Wert die Rohbytes im .NET-Client (double verliert oberhalb von 53 Bit an Genauigkeit).
Float32 / Float64	4/8 Byte	IEEE 754 (Real/LReal).
String	variabel	Feste Länge, ASCII, mit nachgestelltem Leerzeichen/NUL aufgefüllt/akzeptiert.
Bytes	variabel	Rohes Byte-Array, als Hex-Text gelesen/geschrieben.

Validierungsregeln, bei jedem Hinzufügen/Aktualisieren geprüft (die gesamte Änderung wird abgelehnt, wenn eine Regel fehlschlägt): Der Name darf nicht leer und muss innerhalb des Geräts eindeutig sein; die Bitnummer eines Bool-Tags muss 0–7 sein; die Länge eines String-/Bytes-Tags muss mindestens 1 betragen; zwei Tags dürfen nicht dieselben Bytes in demselben Slot/Subslot und derselben Richtung beanspruchen (zwei Bool-Tags auf verschiedenen Bits desselben Bytes sind die eine Ausnahme — eine gängige Art, jedes Bit eines Statusworts einzeln zu benennen). Ein Tag, der auf einen nicht mehr existierenden Slot/Subslot zeigt oder über die Daten

seines Submoduls hinausreicht, wird mit einer Warnung markiert statt direkt blockiert — harmlos, bis das Gerät mit einem anderen Modullayout neu importiert wird.

Kapitel 6

EtherCAT Master

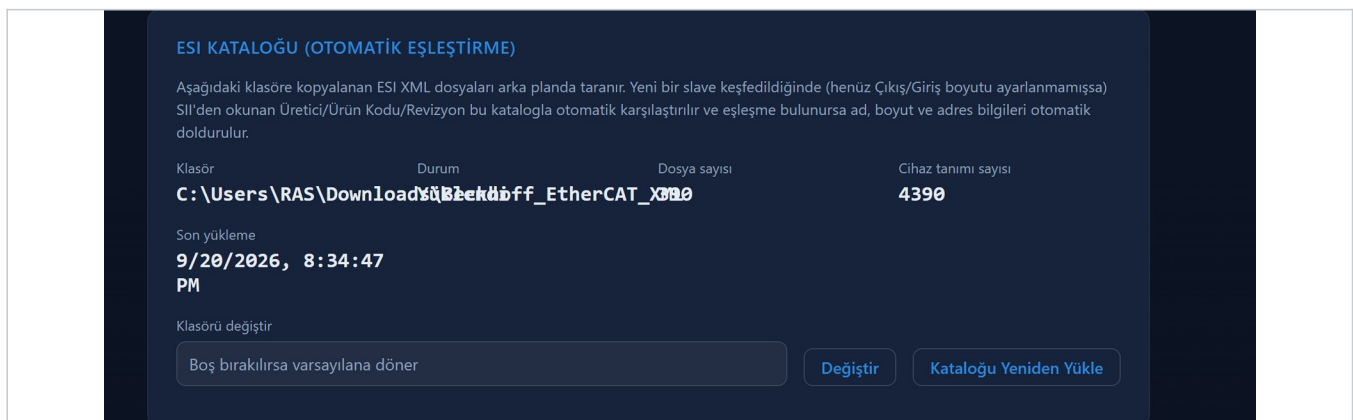
6.1 Überblick

Die EtherCAT-Master-Engine macht den Ethernet-Adapter Ihres PCs zum Master eines EtherCAT-Segments: Sie öffnet den Ring, durchläuft ihn durch die Zustandsmaschine INIT→PREOP→SAFEOP→OP und steuert anschließend den zyklischen Prozessdatenaustausch mit jedem Slave im Ring zur konfigurierten Zykluszeit, vollständig in Software — es wird keine dedizierte EtherCAT-Masterkarte verwendet.

Punkt	Wert
Rolle	Master
Standard-Zykluszeit	4 ms (konfigurierbar)
Slave-Identifikation	ESI-XML-Katalog, automatisch nach Hersteller/Produkt/Revision aus der SII abgeglichen
Gemessener Jitter (typischer PC, 4-ms-Zyklus)	≈ 0,3 ms nach Echtzeit-Feinabstimmung

6.2 ESI-Katalog — automatischer Slave-Abgleich

EtherCAT-Slaves benötigen — anders als PROFINET-Geräte — keine Stellvertreterdatei: Jeder Slave meldet seine Hersteller-ID, seinen Produktcode und seine Revision selbst über die SII (Slave Information Interface), sobald er mit Strom versorgt und in den Ring eingebunden ist. RasProLink liest diese Identität automatisch aus und sucht sie in einem Ordner mit ESI-XML-Dateien — denselben Dateien, die Ihre Slave-Hersteller bereits für TwinCAT, Codesys oder ähnliche Tools veröffentlichen.



ESI-Katalog-Karte — überwachter Ordner für ESI-Dateien, indizierte Gerätesignaturen, letzte Neuladezeit

Kopieren Sie die ESI-XML-Dateien Ihrer Slave-Hersteller in den überwachten Ordner (ein Standardpfad ist vorgegeben und kann über diese Karte geändert werden); neue Dateien werden automatisch erkannt. Ist für einen neu erkannten Slave noch keine Ein-/Ausgangsgröße konfiguriert, vergleicht RasProLink dessen von der SII gemeldete Signatur mit jeder indizierten ESI-Datei und füllt bei einem Treffer automatisch Name, PDO-Größen und Prozessdaten-Offsets aus — kein manuelles Byte-Zählen.

6.3 Prozessdaten-Zuordnung

Die PDO-Zuordnung (Process Data Object) jedes Slaves — sein Eingangs- und Ausgangs-Bit-/Byte-Layout — stammt aus dem abgeglichenen ESI-Eintrag, genau wie die Slot-/Subslot-Tabelle eines PROFINET-Geräts (die Prinzipien aus §5 gelten hier gleichermaßen: Die Richtung wird immer aus Sicht des Slaves benannt, Tags ermöglichen die Benennung eines Byte-/Bit-Bereichs, alle unterstützten Typen von Bool bis Bytes stehen zur Verfügung). Siehe §5 für die gemeinsamen Regeln zu Datenzuordnung und Tags, die alle fünf Engines verwenden.

6.4 .NET-Client-Beispiel

```
using RasProLink.Client;

using var client = new RasProLinkClient("localhost", 8080);
var status = await client.ConnectAsync();

var slaves = await client.GetDevicesAsync();
foreach (var s in slaves)
    Console.WriteLine($"{s.Name} @ {s.State} in={s.InputBytes}B out={s.OutputBytes}B");

// Read one slave's live process data
var io = await client.GetDeviceIoAsync("drive-axis-1");

// Write an output byte range (e.g. a drive control word)
await client.WriteDeviceOutputAsync("drive-axis-1", slot: 1, subslot: 1,
    data: new byte[] { 0x06, 0x00 });
```

Dies ist dieselbe RasProLink.Client-Bibliothek, die von jeder Engine verwendet wird — nur der Zielgerätename ändert sich. Die vollständige API finden Sie in Kapitel 11.

6.5 Fehlerbehebung

Symptom	Wahrscheinliche Ursache
Keine Slaves gefunden	Ring nicht geschlossen (der OUT-Port des letzten Slaves muss zurückgeführt werden, oder eine offene Ring-Topologie konsequent verwendet werden), oder der gewählte Adapter ist nicht der mit dem Ring verkabelte.
Slave bleibt unterhalb von OP	Keine ESI-Übereinstimmung (unbekannte PDO-Größen) — fügen Sie die ESI-Datei des Herstellers dem Katalogordner hinzu und lassen Sie neu scannen, oder ein slave-spezifischer SDO-/Mailbox-Startparameter ist fehlgeschlagen (testen Sie das Objekt manuell mit dem SDO-Werkzeug, §6.7).
Hoher Jitter / gelegentlicher Frameverlust	Eine weitere Anwendung konkurriert um denselben CPU-Kern, oder der Energiesparplan des PCs drosselt — siehe die Echtzeit-Hinweise in §1.3 und §13.4.

6.6 ESI-Dateivorschau und Ketten-Builder — Topologie konfigurieren, bevor sie verdrahtet ist

§6.2 behandelt den automatischen Abgleich: Ein bereits mit Strom versorgter und verdrahteter Slave meldet seine Identität über die SII, und RasProLink schlägt sie im überwachten ESI-Ordner nach. Die Karte ESI-Dateivorschau auf der Slave-Details-Karte geht den umgekehrten Weg — Sie wählen von Hand eine einzelne ESI-XML-Datei aus, RasProLink parst sie im Browser, und Sie können deren Sync-Manager-Größen/-Adressen sowie die CoE-Unterstützung in das Formular des aktuell ausgewählten Slaves übernehmen, oder damit eine ganze Reihe von Slaves aufbauen, bevor überhaupt Hardware am Ring hängt.

Eine einzelne ESI-Datei in der Vorschau ansehen

Wählen Sie zunächst den zu bearbeitenden Slave aus der Slave-Tabelle, dann eine ESI-XML-Datei unter ESI-Dateivorschau. Beschreibt die Datei mehrere Geräte, wählen Sie das richtige aus dem Dropdown Gerät; die Karte zeigt dann Hersteller, Produktcode / Version, ob CoE (Mailbox) unterstützt wird, sowie die von der Datei angegebenen Größen für Ausgänge (Outputs SM) / Eingänge (Inputs SM). Diese Werte in das Formular übernehmen füllt die Ausgangs-/Eingangsgröße und die Adressfelder des ausgewählten Slaves aus diesem ESI-Eintrag — Sie müssen anschließend trotzdem auf Speichern im Formular klicken, genau wie bei manueller Eingabe (§6.3).

Eine Kette aufbauen: Koppler + Klemmen

Modulare IO bei EtherCAT besteht meist aus einem Buskoppler (z. B. Beckhoff EK1100) mit einer Reihe daran angesteckter Klemmen (z. B. EL1008, EL2008); jede Klemme ist ein eigener Slave im Ring, jeweils eine Position hinter der vorherigen. Statt jede einzeln per ESI-Vorschau zu übernehmen und zu speichern, laden Sie die ESI-Datei des Kopplers und klicken auf Zur Kette hinzufügen, dann wiederholen Sie dies für jede Klemme in ihrer physischen Verdrahtungsreihenfolge von links nach rechts — jedes Zur Kette hinzufügen hängt das gerade in der Vorschau angezeigte Gerät an die untenstehende Tabelle Kette (Koppler + Klemmen) an, in der Position, Produktcode / Version, Name, Ausgangs-/Eingangs-Bytezahlen und CoE-Unterstützung aufgeführt sind; eine Zeile kann vor dem Anwenden noch mit Nach oben/unten verschieben umsortiert oder mit Entfernen verworfen werden.

Legen Sie die Start-Ringposition fest (sie wird beim ersten Hinzufügen zu einer leeren Kette standardmäßig auf eine Position hinter dem letzten bereits in der Liste vorhandenen Slave gesetzt) und klicken Sie auf Kette auf Slave-Liste anwenden: RasProLink durchläuft die Kette der Reihe nach, weist jedem Eintrag die Ringposition Startposition + Index sowie die feste Stationsadresse $0x1000 + \text{Ringposition}$ zu und speichert ihn dann mit einem normalen PUT an `/api/ethercat/slaves` — demselben Aufruf, den auch das manuelle Formular verwendet. Belegt bereits ein Slave diese Ringposition, wird dessen Eintrag an Ort und Stelle aktualisiert statt dupliziert, sodass ein erneutes Anwenden einer korrigierten Kette über eine bestehende hinweg gefahrlos ist. Das Ergebnis ist ein ganz normaler Satz von Slave-Konfigurationseinträgen, nicht zu unterscheiden von manuell eingegebenen oder über den ESI-Katalog (§6.2) abgeglichenen — die Kettentabelle selbst ist nur ein Zwischenspeicher und wird nach dem Anwenden geleert.

Tipp: So können Sie einen kompletten Koppler-plus-Klemmen-Aufbau anhand der ESI-Dateien aus dem Herstellerdatenblatt bereits am Schreibtisch konfigurieren, bevor der Schaltschrank verdrahtet ist — nützlich, um eine Konfigurationsdatei vorzubereiten, die mit einer Maschine ausgeliefert wird. Sobald die echte Hardware verdrahtet ist und der Bus öffnet, wird die tatsächliche SII-Identität jedes Slaves genau wie bei jedem anderen Slave (§6.1) gegen die konfigurierte Hersteller-ID/Produktcode/Version geprüft; eine Abweichung (falsche Klemme im falschen Slot) ist sofort in der Spalte ESM-Zustand der Slave-Tabelle sichtbar.

6.7 SDO (CoE) Lesen/Schreiben — manueller CANopen-over-EtherCAT-Zugriff

Neben dem in §6.3 beschriebenen zyklischen PDO-Austausch stellt ein Slave, der CoE (CANopen over EtherCAT) unterstützt, sein Objektverzeichnis auch über die Mailbox bereit — einzelne, über einen 16-Bit-Index und einen 8-Bit-Subindex adressierte Objekte, die unabhängig vom zyklischen Frame einzeln per SDO-Transfer (Service Data Object) gelesen oder geschrieben werden. Dies ist ein manueller, bedarfsgesteuerter Zugriffsweg: Er läuft nicht in jedem Zyklus und ist für Inbetriebnahme und Diagnose gedacht, nicht für Regelkreis-Daten (dafür Tags/PDO-Zuordnung verwenden, §6.3).

Wählen Sie in der Slave-Tabelle einen Slave; die Karte SDO (CoE) Lesen/Schreiben erscheint weiter unten auf der Slave-Details-Karte, unterhalb von Live-IO (§6.9), nur wenn für diesen Slave die aus der ESI übernommene CoE-Unterstützung gesetzt ist — andernfalls zeigt die Karte Dieser Slave unterstützt CoE (Mailbox) nicht, und die Felder bleiben verborgen. Geben Sie Index (hex, z. B. 1018 für das Identity Object) und Subindex ein, klicken Sie dann auf Lesen, um den aktuellen Wert als Hex abzurufen, oder geben Sie Daten (hex) ein und klicken Sie auf Schreiben, um sie zu senden. Beide laufen über die Mailbox des Slaves statt über den zyklischen PDO-Frame, ein Lese-/Schreibvorgang kann daher merklich länger als ein Buszyklus dauern und bei einem Slave, der noch nicht über PREOP hinaus ist, fehlschlagen.

Hinweis: Ein fehlgeschlagener SDO-Transfer meldet zusätzlich zur Fehlermeldung den CANopen-Abbruchcode des Slaves — denselben Abbruchcode, den das Handbuch Ihres Slaves für dieses Objekt dokumentiert (z. B. Objekt existiert nicht, oder Schreibzugriff für einen schreibgeschützten Eintrag verweigert).

6.8 Netzwerkadapter-Auswahl und Bus-Zykluszeit

§6.1 nennt die Standard-Zykluszeit von 4 ms als Tatsache über die Engine; in diesem Abschnitt wird beschrieben, wo sowohl der Adapter als auch diese Zykluszeit tatsächlich eingestellt werden — auf der Karte Netzwerkadapter-Auswahl oben in der Web-Oberfläche.

Netzwerkadapter-Auswahl

EtherCAT arbeitet mit rohen Ethernet-Frames statt mit IP, daher muss dem Master mitgeteilt werden, mit welchem Netzwerkadapter der Ring physisch verbunden ist. Das Dropdown Netzwerkadapter listet jeden Adapter auf, den RasProLink sehen kann; belassen Sie es auf (automatisch auswählen), damit der Master selbst wählt, oder wählen Sie einen ausdrücklich aus. Ein optionales Feld MAC (manuell, optional) erlaubt es, den Adapter anhand der MAC-Adresse statt des vom Betriebssystem vergebenen Namens festzulegen — nützlich auf einem PC, bei dem sich Adapternamen nach einem Treiber-Update oder dem erneuten Anstecken einer USB-Dockingstation ändern können. Ein Klick auf Anwenden startet den Bus neu — ein Adapterwechsel betrifft das gesamte Segment und kann daher nicht ohne ein kurzes erneutes Öffnen des Rings angewendet werden, genau wie das Hinzufügen/Entfernen eines Slaves (§6.3).

Bus-Zykluszeit

Das Feld Zyklusintervall (ms) auf derselben Karte legt fest, wie oft der Master den zyklischen Austausch durchführt (die 4 ms in §6.1 sind nur der Standardwert dieses Felds, kein fester Wert). Geben Sie ein neues Intervall ein und klicken Sie auf das daneben liegende Anwenden: Anders als beim Adapter baut eine Änderung der Zykluszeit die FMMU-/Sync-Manager-Konfiguration des Rings nicht neu auf und erzwingt daher keinen Neustart — der Wortlaut der Oberfläche selbst ist eindeutig: Angewendet, der neue Wert wird beim nächsten Öffnen des Busses wirksam, d. h. er wird erst beim nächsten Verbindungszyklus übernommen und nicht mitten im laufenden Betrieb.

Hinweis: "Beim nächsten Öffnen des Busses" tritt in der Praxis früher ein als ein Dienstneustart: Das Stoppen und Starten des Masters über die Statuskarte (§6.1), oder jede Änderung, die ohnehin bereits einen Neustart auslöst

(ein Adapterwechsel wie oben, oder das Hinzufügen/Entfernen eines Slaves, §6.3), öffnet den Ring ebenfalls neu und übernimmt die neue Zykluszeit.

6.9 Live-IO — manuelles Lesen/Schreiben je Slave in der Web-Oberfläche

§6.4 zeigt das Lesen/Schreiben von Prozessdaten über .NET-Client-Code; dieselben Rohbytes sind auch ohne eine Zeile Code direkt über das Panel Live-IO auf der Slave-Details-Karte im Browser erreichbar.

Wählen Sie in der Slave-Tabelle einen Slave, um dessen Slave-Details-Karte zu öffnen; unterhalb der Identitätsfelder (ESM-Zustand, Vendor ID, Produktcode, Version, Seriennummer) zeigt der Block Live-IO den Eingang (vom Slave, schreibgeschützt, hex) — das aktuelle, automatisch aktualisierte und schreibgeschützte Eingangsabbild des Slaves — sowie den Ausgang (zum Slave, hex), ein editierbares Feld: Geben Sie eine Hex-Zeichenfolge ein und klicken Sie auf Senden, um sie im nächsten Zyklus direkt in das Ausgangsabbild dieses Slaves zu schreiben. Dies ist derselbe Ein-/Ausgangs-Byte-Bereich, den Client-Code über `GetDeviceIoAsync/WriteDeviceOutputAsync` (§6.4) erreicht, hier direkt für Labortests ohne Client-Projekt zugänglich gemacht — etwa um beim Verdrahten eines Schaltschranks eine Ausgangsspule zwangsweise zu setzen oder zu bestätigen, dass sich ein Eingangsbyte beim Auslösen eines Sensors ändert, noch bevor überhaupt Tag-Namen definiert wurden (§6.3).

Hinweis: Ein auf diesem Weg gesendeter Wert wird durch den nächsten zyklischen Schreibvorgang eines verbundenen Clients oder durch eine Tag-Schreibung (§6.3) überschrieben — Live-IO hält oder verriegelt keinen Ausgang, es fügt lediglich einen einzelnen Schreibvorgang in den laufenden zyklischen Austausch ein. Es ist für Labortests und Inbetriebnahme gedacht, nicht als Ersatz für eine laufende Client-Anwendung.

Kapitel 7

EtherNet/IP Scanner

7.1 Überblick

Die EtherNet/IP-Scanner-Engine macht Ihren PC zum Originator einer oder mehrerer Klasse-1-CIP-Verbindungen (implizit, zyklisch): Sie öffnet zu jedem konfigurierten Target/Adapter ein Forward Open, tauscht dann Ein- und Ausgangs-Assemblies zum ausgehandelten RPI (Requested Packet Interval) aus und öffnet die Verbindung bei einem Abbruch automatisch erneut.

Punkt	Wert
Rolle	Originator
Verbindungs-klasse	CIP Klasse 1 (zyklisch, UDP), ein Forward Open pro Target
Identifikation	Target-IP + Assembly-Instanznummern (Eingang/Ausgang/Konfiguration)

7.2 Ein Target hinzufügen

Fügen Sie ein Target über die Gerätekarte der Weboberfläche hinzu, indem Sie dessen IP-Adresse und die Assembly-Instanznummern aus der EDS-Datei oder dem Datenblatt (Eingang, Ausgang und ggf. Konfigurations-Assembly) sowie die gewünschte RPI eingeben — oder überspringen Sie die manuelle Suche ganz, indem Sie zuerst eine Geräteerkennung (§7.5) ausführen, die dasselbe Formular vorausfüllen kann. RasProLink öffnet sofort das Forward Open, und die Zeile des Targets zeigt Verbinden → Läuft, sobald der erste zyklische Austausch gelingt — derselbe Statusverlauf wie in der Geräteliste des PROFINET Controller (§9.2).

7.3 .NET-Client-Beispiel

```
using RasProLink.Client;

using var client = new RasProLinkClient("localhost", 8080);
await client.ConnectAsync();

var targets = await client.GetDevicesAsync();
foreach (var t in targets)
    Console.WriteLine($"{t.Name} {t.IpAddress} [{t.State}] in={t.InputBytes}B out={t.OutputBytes}B");

var io = await client.GetDeviceIoAsync("vfd-conveyor-3");
await client.WriteDeviceOutputAsync("vfd-conveyor-3", slot: 0, subslot: 1,
    data: new byte[] { 0x01, 0x00 }); // e.g. run/stop word
```

7.4 Fehlerbehebung

Symptom	Wahrscheinliche Ursache
Forward Open abgelehnt	Assembly-Instanznummern oder -Größen stimmen nicht mit der EDS des Targets überein, oder das Target hat bereits die maximale Anzahl an Verbindungen von einem anderen Scanner/einer SPS

Symptom	Wahrscheinliche Ursache
	geöffnet.
Verbindung läuft wiederholt in Timeout	RPI für den Netzwerkpfad zu aggressiv eingestellt, oder ein Switch/VLAN fügt Latenz hinzu — siehe die Echtzeit-Netzwerkhinweise in §2.

7.5 Geräteerkennung — Targets im Netz finden

Die Geräteerkennungs-Karte sendet einen UDP-ListIdentity-Broadcast und listet jedes antwortende EtherNet/IP-Gerät mit IP-Adresse, Vendor-ID, Produktcode, Version und Produktname auf — das EtherNet/IP-Gegenstück zur DCP-Netzwerksuche des PROFINET Controller (§9.3). Sie läuft für ein festes Zeitfenster (standardmäßig 2 s) und öffnet dabei selbst keine CIP-Verbindung; es handelt sich um eine reine Erkundung, schreibgeschützt.

Jedes Ergebnis trägt eine Spalte EDS-Übereinstimmung: RasProLink gleicht die Vendor-ID/den Produktcode/Gerätetyp/die Version des antwortenden Geräts automatisch mit dem EDS-Katalog ab (§7.6). "Diese IP verwenden" füllt nur das IP-Feld des Formulars zum Hinzufügen eines Targets (§7.2); besteht eine Katalogübereinstimmung, wird stattdessen "Automatisch ausfüllen" angeboten, das zusätzlich die Assembly-Instanznummern und Ein-/Ausgangsgrößen aus dieser übereinstimmenden EDS vorausfüllt — das Target lässt sich so hinzufügen, ohne von Hand etwas nachzuschlagen.

Hinweis: ListIdentity ist ein Broadcast im lokalen Subnetz — ein Target hinter einem Router oder einer VLAN-Grenze erscheint hier nicht, selbst wenn ein Forward Open zu seiner bekannten IP weiterhin erfolgreich wäre. Fügen Sie es in diesem Fall manuell hinzu.

7.6 EDS-Katalog — automatische Zuordnung

Die EDS-Katalog-Karte überwacht einen Ordner mit Hersteller-EDS-Dateien und durchsucht ihn im Hintergrund — das EtherNet/IP-Gegenstück zum PROFINET-GSDML-Katalog (§9.10). Legen Sie die von Ihren Geräteherstellern gelieferten .eds-Dateien in den Ordner (Standard ist ethernetip-scanner-eds, neben den übrigen Konfigurationsdateien von RasProLink), und RasProLink indiziert die Vendor-ID/den Produktcode/Gerätetyp/die Version jeder Gerätedefinition zusammen mit ihrem Assembly-Layout, sodass die Geräteerkennung (§7.5) daraus abgleichen und vorausfüllen kann, ohne dass Sie auch nur eine einzige EDS-Datei von Hand öffnen.

Feld	Bedeutung
Ordner	Das überwachte Verzeichnis; ändern Sie es über die Steuerelemente Ändern/Durchsuchen (ein Ordnerauswahldialog wird angeboten, wenn RasProLink auf einem interaktiven Desktop läuft, andernfalls den Pfad direkt eingeben).
Status / Anzahl Dateien / Anzahl Gerätedefinitionen	Ob der letzte Scan geladen ist oder noch läuft, wie viele .eds-Dateien gelesen wurden und wie viele Gerätedefinitionen daraus extrahiert wurden (eine Datei kann mehrere Gerätevarianten beschreiben).
Katalog neu laden	Erzwingt einen sofortigen erneuten Scan des Ordners, ohne auf den Hintergrundwächter zu warten — direkt nach dem Kopieren neuer EDS-Dateien verwenden.

Hinweis: Eine Datei, die sich nicht parsen lässt, wird übersprungen, nicht als fataler Fehler behandelt — sie wird namentlich in der Fehlerliste des Katalogs gemeldet, sodass Sie sie korrigieren oder ersetzen können, während jede andere Datei im Ordner weiterhin normal geladen wird.

7.7 EDS-Dateivorschau — eine Datei vor dem Hinzufügen eines Geräts prüfen

Die Karte EDS-Dateivorschau erlaubt es, den Rohtext einer einzelnen EDS-Datei einzufügen und sie sofort zu parsen, ohne sie in den EDS-Katalog-Ordner (§7.6) aufzunehmen und ohne ein Target anzulegen. Sie meldet Herstellername und -code, Produktcode/-typ/-name, Version sowie jede in der Datei definierte Assembly-Instanz mit ihrer Bytegröße und Richtung (Eingang/Ausgang) — nützlich, um eine gerade per E-Mail erhaltene Herstellerdatei zu prüfen oder Assembly-Nummern eines Geräts zu lesen, zu dem Sie noch keine Verbindung herstellen wollen.

Lässt sich der [Connection Manager]-Pfad der Datei sauber auflösen, erscheint die Schaltfläche "Diese Werte in das Formular übernehmen" und kopiert die geparsen Assembly-Instanzen und -Größen direkt in das Formular zum Hinzufügen eines Targets (§7.2). Kann die Richtung nicht aus der Datei ermittelt werden (kein verwertbarer Connection-Manager-Pfad), wird diese Schaltfläche ausgeblendet und Sie lesen die Instanznummern manuell aus der Tabelle ab.

Kapitel 8

EtherNet/IP Adapter

8.1 Überblick

Die EtherNet/IP-Adapter-Engine macht Ihren PC zu einem Target: Sie wartet auf ein Forward Open von einem Scanner — typischerweise einer SPS — und tauscht nach dessen Annahme die konfigurierten Ein-/Ausgangs-Assemblies zum vom Scanner angeforderten RPI aus. Dies ist das Spiegelbild der Scanner-Engine: Hier ist Ihr PC das gescannte Gerät, nicht der Scanner selbst.

Punkt	Wert
Rolle	Target/Adapter
Identifikation gegenüber dem Scanner	Konfigurierte Assembly-Instanzen (Eingang/Ausgang/Konfiguration) und ihre Byte-Größen

8.2 Konfiguration der lokalen Assemblies

Definieren Sie in der Weboberfläche das Eingangs-Assembly (Daten, die Ihr PC erzeugt und die vom Scanner gelesen werden) und das Ausgangs-Assembly (Daten, die der Scanner schreibt und die von Ihrem PC verarbeitet werden) mit ihren Instanznummern und Byte-Größen — passend zu dem, was die verbindende Scanner-/SPS-Konfiguration (oder EDS) erwartet. Nach dem Speichern beginnt RasProLink, Forward-Open-Anfragen für diese Instanz anzunehmen.

8.3 .NET-Client-Beispiel

```
using RasProLink.Client;

using var client = new RasProLinkClient("localhost", 8080);
await client.ConnectAsync();

// Read what the Scanner/PLC most recently wrote to us
var io = await client.GetDeviceIoAsync("self");

// Publish new input data for the Scanner/PLC to read next cycle
await client.WriteDeviceOutputAsync("self", slot: 0, subslot: 1,
    data: new byte[] { 0x2A, 0x00, 0x00, 0x00 });
```

8.4 Fehlerbehebung

Symptom	Wahrscheinliche Ursache
Scanner kann keine Verbindung öffnen	Assembly-Instanz-/Größenkonflikt zwischen der Konfiguration des Scanners und den konfigurierten Assemblies dieser Engine, oder das Assembly ist noch nicht gespeichert/gestartet.

8.5 Verbindungsstatus und Statistiken

Zwei Karten auf der Weboberfläche des Adapters geben Ihnen einen Live-Einblick in die Verbindung, die ein Scanner mit dieser Engine hält, ohne dass ein REST-Aufruf oder eine Paketaufzeichnung nötig wäre — eine separate Geräteliste wie bei der Scanner-Engine (§7) gibt es hier nicht, da der Adapter stets nur die eine Verbindung hat, die ein Scanner zu ihm öffnet.

Verbindungsstatus

Zeigt, ob aktuell ein Scanner verbunden ist, und falls ja, dessen IP-Adresse sowie den Zeitpunkt des Verbindungsaufbaus ("Verbunden seit"), zusätzlich die beiden vom Forward Open ausgehandelten CIP-Verbindungs-IDs — Input Connection ID (die O→T-Verbindung, die trägt, was diese Engine empfängt) und Output Connection ID (die T→O-Verbindung, die trägt, was sie sendet) — sowie den Zeitstempel des zuletzt vom Scanner empfangenen Eingangs, nützlich, um einen Scanner zu bemerken, der verstummt ist, ohne die Verbindung formell zu schließen.

Statistiken

Meldet Forward_Open-Versuche (jede von dieser Engine behandelte akzeptierte oder abgelehnte Forward-Open-Anfrage, einschließlich solcher vor der aktuellen Verbindung), Verbindungsverluste (wie oft eine bestehende Verbindung abgebrochen ist), Gesendete Frames und Empfangene Frames (zyklische I/O-Frame-Zähler seit dem Start der Engine) sowie Jitter, dargestellt auf dieselbe Weise wie das Jitter-Diagramm auf Geräteebe des PROFINET Controller (§9.4, §13.4) — ein hoher oder wachsender Jitter-Wert deutet auf dieselben Netzwerkpfad-Ursachen hin, die in der Fehlerbehebungstabelle des §7.4 behandelt werden.

Hinweis: Es gibt weder einen REST-Endpunkt noch eine UI-Liste, die andere Scanner/SPS aufzählt, die eventuell ebenfalls für eine Verbindung zu diesem Adapter konfiguriert sind — ein Adapter-Target bedient stets nur das eine Forward Open, das es gerade hält, sodass "wer ist verbunden" genau das ist, was die obige Karte Verbindungsstatus bereits zeigt.

8.6 I/O-Überwachung / Simulation — Labortest ohne Scanner

Die Karte I/O-Überwachung / Simulation erlaubt es, die I/O dieser Engine ohne angeschlossenen echten Scanner oder Feldhardware zu betreiben. Sie zeigt das aktuelle Eingangs-Assembly schreibgeschützt als Hex an (die zuletzt von einem verbundenen Scanner geschriebenen Daten, oder komplett Null, wenn keiner verbunden ist) und bietet ein Hex-Feld sowie eine Senden-Schaltfläche für das Ausgangs-Assembly, mit denen Sie manuell die Bytes einspeisen können, die sonst ein echter Scanner schreiben würde — nützlich, um Tag-Definitionen (§5) oder eine Integration mit dieser Engine zu testen, bevor ein SPS-Programm existiert.

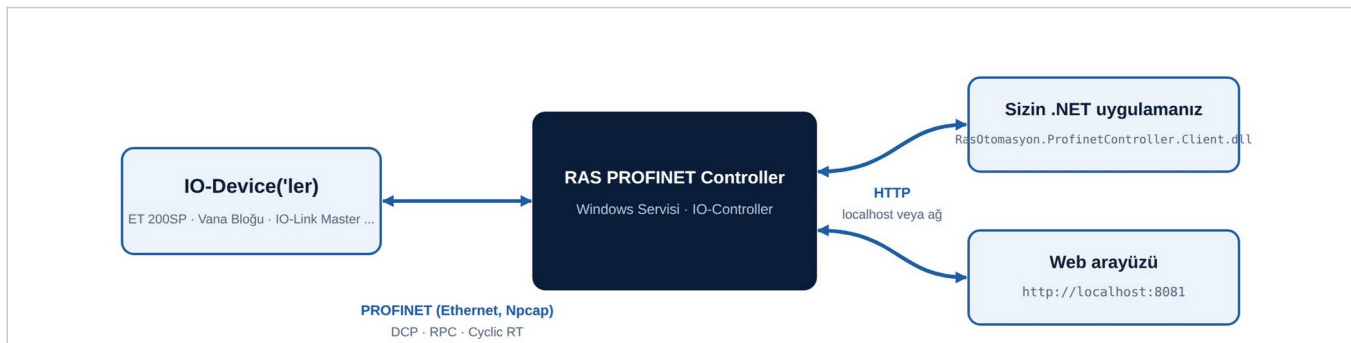
Tipp: "Ausgang" und "Eingang" folgen hier der Sichtweise des Scanners, passend zu den Feldnamen aus §8.2: Was Sie in dieser Karte eingeben und per Senden übertragen, ist genau die Daten, die der verbundene Scanner als sein Eingangs-Assembly liest.

Kapitel 9

PROFINET Controller

9.1 Überblick

Die PROFINET-Controller-Engine verwandelt Ihren Windows-PC in einen echten PROFINET-IO-Controller — die Rolle, die normalerweise eine SPS innehat. Über einen echten Netzwerkadapter, mit denselben Protokollen, die eine SPS verwendet — DCP zur Erkennung, RPC zum Verbindungsaufbau, zyklische Echtzeitrahmen für den laufenden Datenaustausch — verbindet sie sich direkt mit einem oder mehreren PROFINET-IO-Geräten von Drittanbietern: Remote-I/O-Blöcke, Ventilinseln, IO-Link-Master, jede Art von GSDML-beschriebenem Gerät. Eine SPS muss nicht dazwischen sitzen.

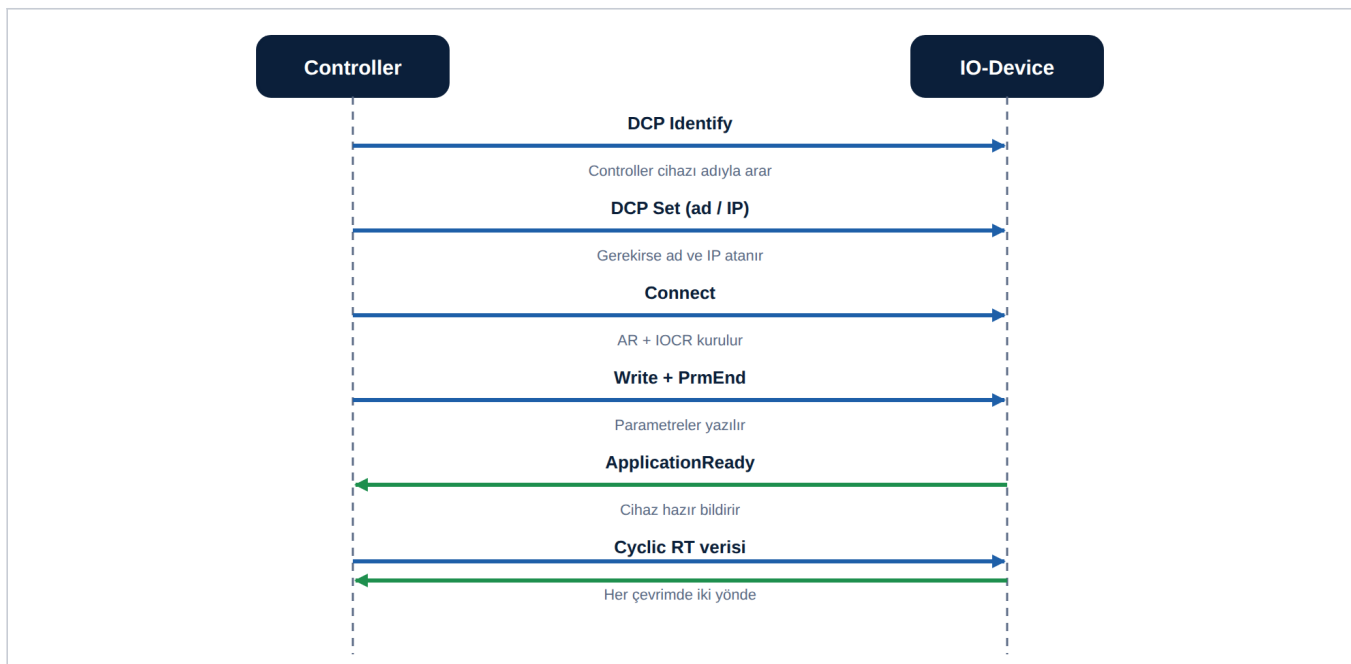


IO-Devices auf der einen Seite, Ihre Anwendung und die Weboberfläche auf der anderen — der Controller-Dienst steht in der Mitte

Eine einzelne Controller-Instanz ist nicht auf ein Gerät beschränkt: Sie öffnet einen Netzwerkadapter und einen RPC-Socket, über die jedes konfigurierte IO-Device bedient wird — die Geräteanzahl ist eine Lizenz-/praktische Frage, keine architektonische.

9.2 Verbindungsablauf

Jedes Gerät durchläuft dieselbe Sequenz, die in der Geräteliste als Status angezeigt wird: Resolving (DCP Identify/Set) → Connecting → Parametrizing → WaitingApplicationReady → Running.



DCP Identify/Set, Connect, Write+PrmEnd, ApplicationReady, dann zyklische RT-Daten in beide Richtungen bei jedem Zyklus

9.3 IO-Devices hinzufügen

IO-DEVICE'LER

TIA Portal'dan dışa aktarılan XDB dosyasını buraya yükleyin; içindeki her cihaz otomatik olarak listeye eklenir ve etkinse hemen bağlanmaya çalışılır.

XDB dosyasını buraya sürükleyin ya da seçmek için tıklayın

CİHAZ	IP	DURUM	GİRİŞ / ÇIKIŞ	ÇEVİRİM	ETKİN	
ET 200SP Giriş/Çıkış	192.168.0.21	Çalışıyor	1B / 1B	4 ms	☒	Sil
Vana Blok 2	192.168.0.22	Hata	0B / 2B	8 ms	☒	Sil
IO-Link Master 3	192.168.0.23	Kapalı	32B / 32B	16 ms	☐	Sil

IO-Devices-Karte — legen Sie eine .xdb-Datei ab, um alle darin enthaltenen Geräte in einem Schritt hinzuzufügen

Es gibt zwei unabhängige Wege, ein IO-Device hinzuzufügen, und nur einer davon benötigt TIA Portal. Eine Netzwerksuche findet jedes PROFINET-Gerät im Netz per DCP und gleicht für jedes dessen Hersteller-/Geräte-ID automatisch mit dem GSDML-Katalog ab — ganz ohne .xdb (§9.10). Die folgenden Schritte behandeln den zweiten Weg: den Import einer aus TIA Portal exportierten .xdb-Datei.

Der Controller benötigt für diesen Weg keine GSDML-Datei direkt — stattdessen liest er eine Geräte-Stellvertreterdatei (.xdb), die TIA Portal bereits für ein in einem Projekt konfiguriertes Gerät exportiert. Diese Datei enthält bereits alles, was der Controller wissen muss: den Stationsnamen des Geräts, die IP-Konfiguration und das vollständige Slot-/Subslot-Layout jedes Moduls mit Ein-/Ausgangs-Bytelängen und Parametern — nichts davon muss manuell neu eingegeben werden. Siehe §9.10 für den alternativen Weg, der TIA Portal und die .xdb-Datei vollständig überspringt.

Spalte	Bedeutung
Gerät	Anzeigename (fällt auf den Stationsnamen zurück); bei einem Gerät mit Modul-/Submodul-Abweichung erscheint darunter zusätzlich eine kleine Warnung „Modulabweichung“ (§9.4).
IP	Die eigene IP-Adresse des Geräts.
Status	Läuft (zyklische Daten fließen), Fehler (letzter Versuch fehlgeschlagen) oder Aus (deaktiviert / nie versucht).
Ein / Aus	Gesamte Ein- und Ausgangsbyteanzahl über alle Submodule des Geräts.
Zyklus	Konfigurierte Aktualisierungszeit.
Aktiv	Deaktivieren, um die Verbindung zu stoppen, ohne die Gerätekonfiguration zu entfernen.

9.4 Gerätedetails, Live-IO und Tags

CİHAZ AYRINTISI

Modül Yapısı

SLOT	MODÜL	ALT SLOT	ALT MODÜL	GİRİŞ	ÇIKIŞ	PARAMETRE
1	IM 155-6 PN HF (Kafa Modülü) (0x00000001)	0x0001	Arayüz (0x00000001)	0B	0B	0
2	8DI DC24V (0x0000AB12)	0x0001	8DI DC24V (0x0000AB12)	1B	0B	3
3	8DO DC24V/0.5A (0x0000AB20)	0x0001	8DO DC24V/0.5A (0x0000AB20)	0B	1B	2

Canlı IO

SLOT	MODÜL	VERİ (HEX)	DURUM
2	8DI DC24V	9C	OK
3	8DO DC24V/0.5A	3F Yaz	OK

Etiketler

Ham veriye (yukarıdaki Canlı IO) ek olarak, bayt/bit konumuna bir AD ve TİP verin — okuma/yazma bundan sonra bu adla yapılabilir. XDB ile eklenen cihazlarda da etiket tanımlanabilir.

AD	SLOT	YÖN	TİP	KONUM	DEĞER	
Sensor_Giris1	2 / 1	Giriş	bool	@0.0	false	Sil
Motor_Calistir	3 / 1	Çıkış	bool	@0.0	true Yaz	Sil

1/0x0001 — Arayüz (in 0B / out 0B)

Giriş (cihazdan)

Int16

İstatistik

Gönderilen 812940	Alınan 812935	Alarm 0	Bağlantı Denemesi 1
Bağlantı Kaybı 0	Jitter 0.280 ms		

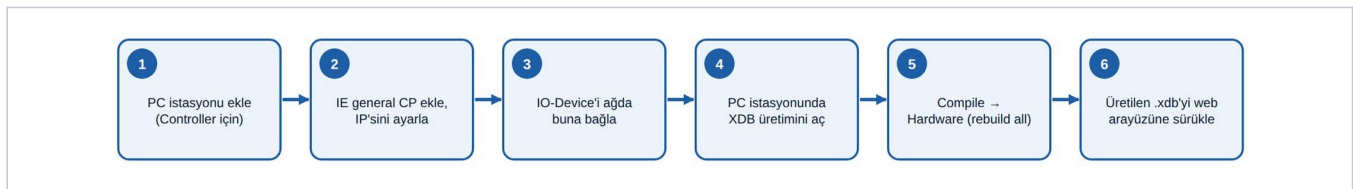
Gerätedetails — Modulstruktur, Live-IO mit manuellem Hex-Schreibfeld, Tags und Verbindungsstatistiken inklusive Jitter

Ein Klick auf ein Gerät in der Liste öffnet drei Bereiche: Modulstruktur (das aus der .xdb importierte oder aus dem GSDML-Katalog konfigurierte Slot-/Subslot-Layout, hier schreibgeschützt), Live-IO (die aktuellen Rohbytes jedes Submoduls, mit einem Hex-Feld und einer Schreiben-Schaltfläche an Ausgangssubmodulen für schnelle Labortests) und Tags (benennen Sie einen Byte-/Bit-Bereich, damit er nach Namen statt nach rohem Hex gelesen oder geschrieben werden kann — Regeln in §5). Die Statistiken unten melden gesendete/empfangene Rahmen, Alarmer, Verbindungswiederholungen, Verbindungsverluste und Zyklus-Jitter (§13.4) — der Jitter erhält weiter unten auf der Seite außerdem ein eigenes Live-Diagramm, das laufendes Min/Max/Durchschnitt statt nur die Momentanzahl in der Statistikzeile verfolgt.

Bei jedem Connect meldet das Gerät seine tatsächlichen Modul- und Submodul-Idents pro Slot/Subslot zurück, und der Controller vergleicht sie mit dem konfigurierten Layout. Ein Slot ohne bestücktes Modul, mit falsch bestücktem Modul oder mit einer nicht passenden Submodul-Identität kennzeichnet das Gerät in der Liste mit einer kleinen Warnung „Modulabweichung“ unter seinem Namen (§9.3) — das Gerät kann trotz vorliegender Abweichung dennoch Läuf erreichen, da manche Submodulzustände (z. B. ein Ersatzwert) den Datenaustausch nicht blockieren. Das Öffnen der Gerätedetailansicht zeigt dann eine Zeile „Modulabweichung“, die jeden vom Gerät tatsächlich gemeldeten Slot und Subslot zusammen mit seinem Zustand auflistet (auf Modulebene kein Modul / falsches Modul / richtiges Modul / Ersatzmodul, bzw. je Submodul ok / Ersatz / falsch / kein) — damit lässt sich finden, welches physische Modul getauscht, weggelassen oder an den falschen Slot verdrahtet wurde, ohne den Schaltschrank abzulaufen.

9.5 TIA Portal — Export einer Geräte-Stellvertreterdatei

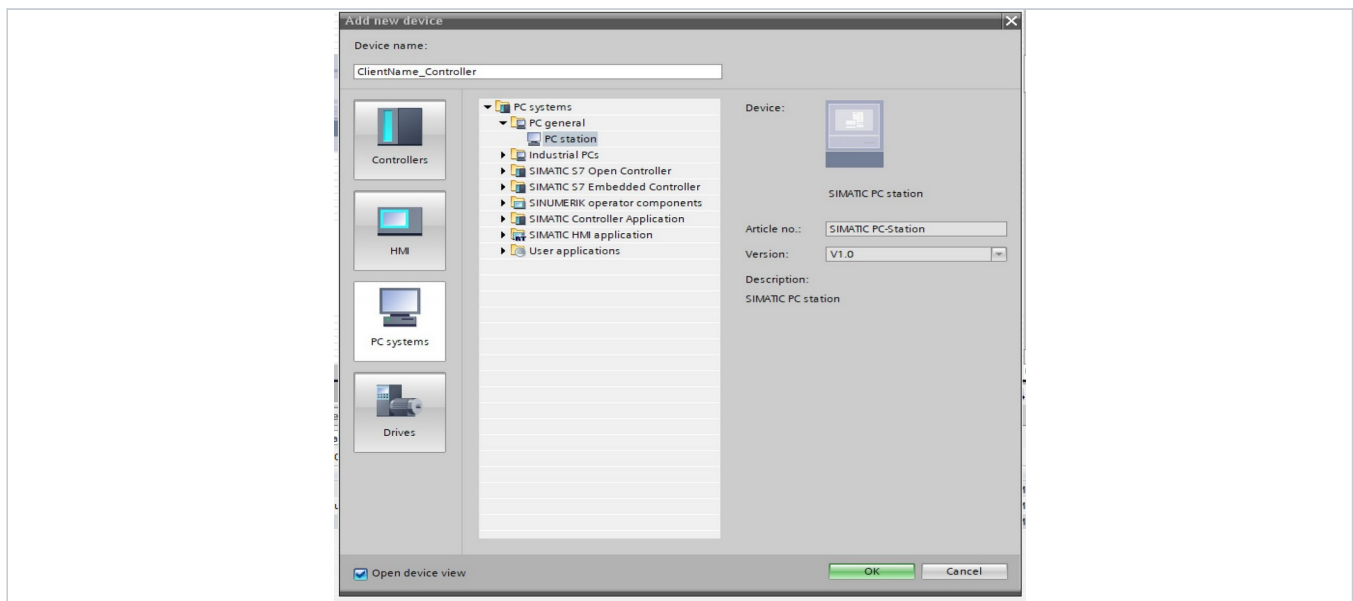
Sitzt das IO-Device bereits in einem echten SPS-Projekt (S7-1200/1500 usw.), klicken Sie es mit der rechten Maustaste an und exportieren Sie seine Geräte-Stellvertreterdatei direkt. In den meisten Aufbauten gibt es jedoch überhaupt keine physische SPS — diese Rolle übernimmt der Controller selbst —, sodass das TIA-Projekt eine Station benötigt, die ihn repräsentiert. Die folgenden Schritte zeigen mit Screenshots aus einem echten TIA-Portal-Projekt diesen gängigen Weg.



Gesamtablauf — Details zu jedem Schritt unten

1. PC-Station für den Controller hinzufügen

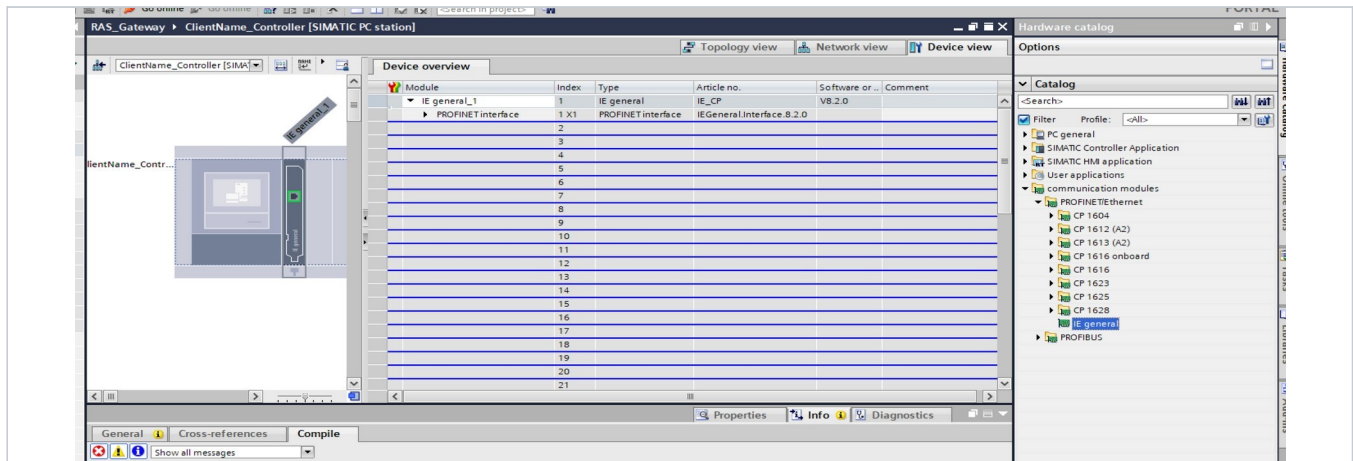
Öffnen Sie im Projektbaum Add new device, wählen Sie PC systems → PC general → PC station und vergeben Sie einen aussagekräftigen Namen (z. B. <Kunde/Standort>_Controller).



Add new device → PC systems → PC general → PC station

2. PROFINET-Schnittstelle hinzufügen (IE general)

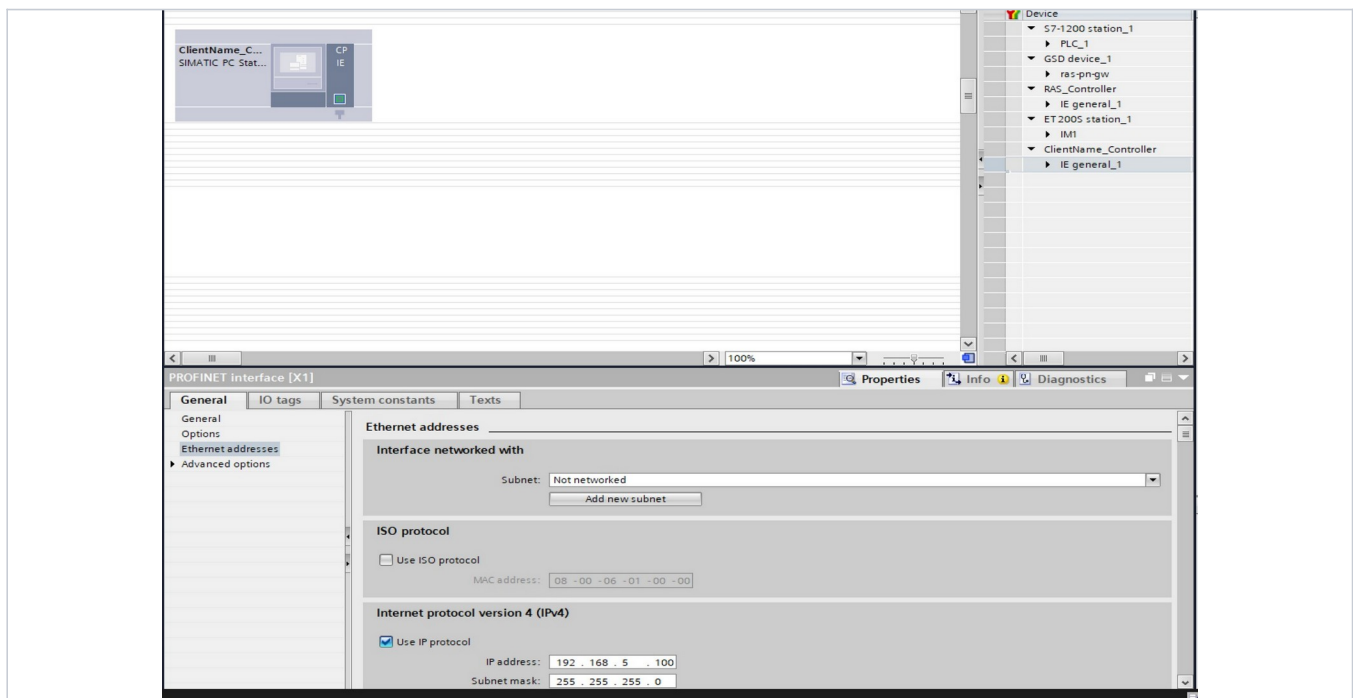
Ziehen Sie aus dem Hardwarekatalog Communication modules → PROFINET/Ethernet → IE general auf die Baugruppenträger der PC-Station — dieses Modul repräsentiert den PROFINET-seitigen Netzwerkadapter des Controllers.



IE general-Modul zur PC-Station hinzugefügt, Hardwarekatalog rechts

3. IP-Adresse des Controllers festlegen

Geben Sie auf der Seite Properties → Ethernet addresses der neuen Schnittstelle die IP-Adresse und Subnetzmaske ein, die dieser PC tatsächlich verwendet (bei Bedarf ein neues Subnetz hinzufügen) — sie muss im selben Subnetz wie die IO-Devices liegen.

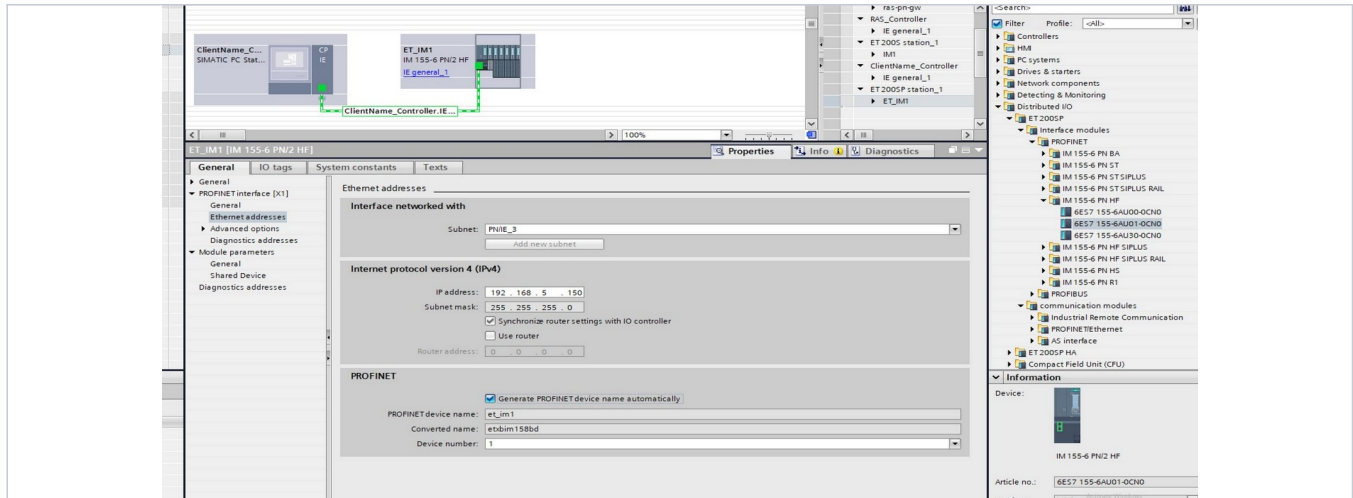


PROFINET-Schnittstelle → Ethernet addresses: IP und Subnetz

Hinweis: Die hier eingegebene IP ist nur die interne Referenz von TIA — TIA kann sie keiner echten Netzwerkkarte zuweisen. Dass diese IP der Netzwerkkarte des Controller-PCs TATSÄCHLICH zugewiesen ist (und in den eigenen Einstellungen des Controllers, §9.6, dieselbe IP erscheint), bleibt Ihre Verantwortung.

4. IO-Device hinzufügen und im Netzwerk verbinden

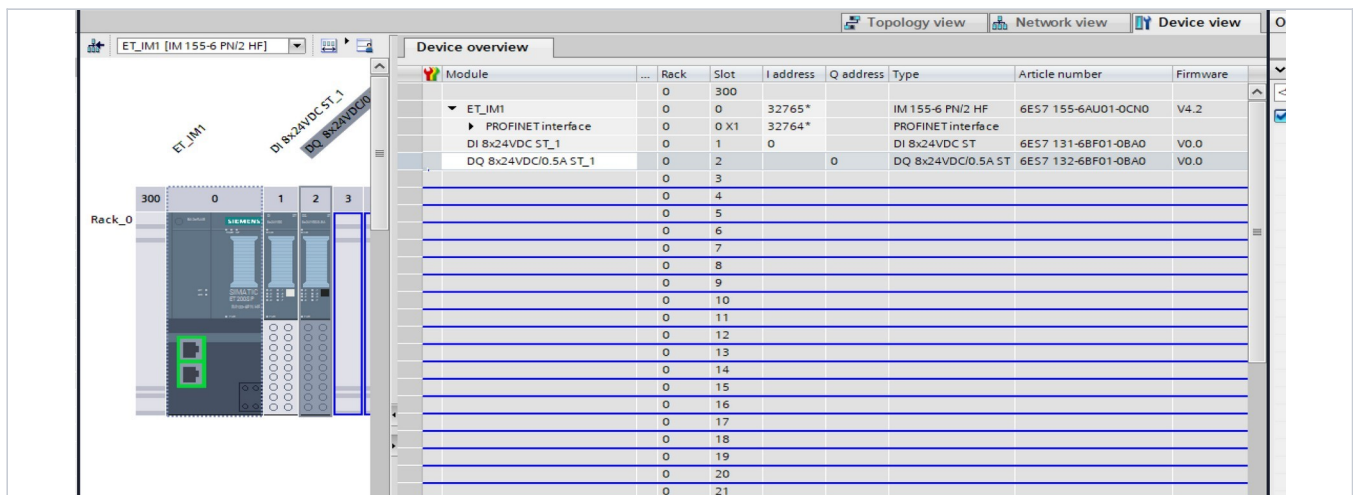
Fügen Sie das IO-Device dem Projekt hinzu (importieren Sie zuerst dessen GSDML, falls der Hardwarekatalog es noch nicht enthält), zeichnen Sie dann in der Network view mit der Maus eine PROFINET-Verbindung von der Schnittstelle der PC-Station zur Schnittstelle des IO-Devices — dadurch kommen beide in dasselbe IO-System. Legen Sie den eigenen Stationsnamen und die IP des IO-Devices (gleiches Subnetz wie der Controller) auf dessen Seite Ethernet addresses fest.



Network view: PC-Station ↔ IO-Device-Verbindung; die Ethernet-addresses-Seite des IO-Devices

5. Module des IO-Devices konfigurieren

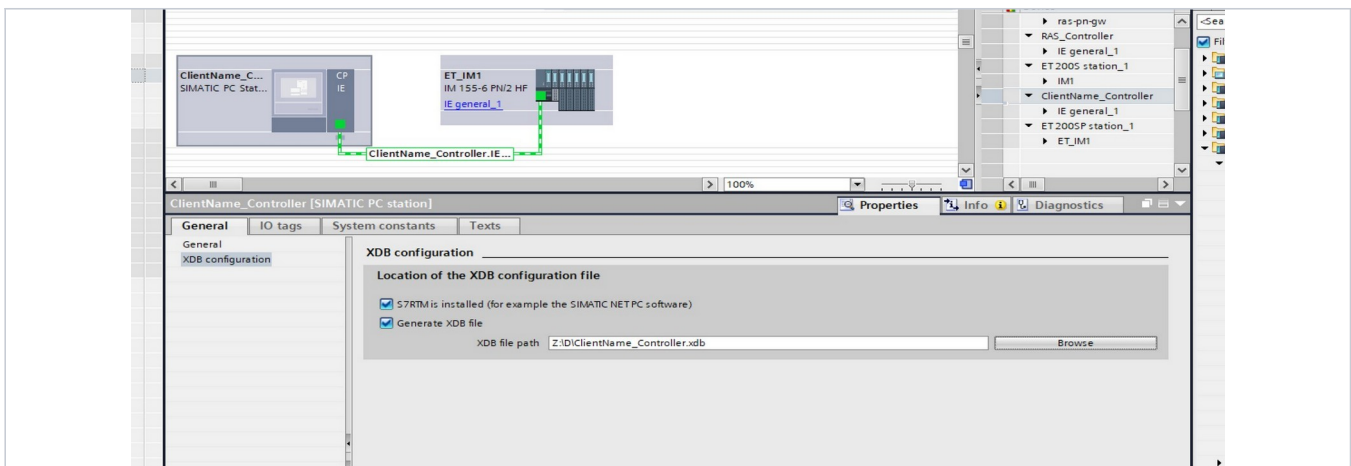
Fügen Sie in der eigenen Geräteansicht des IO-Devices die physisch installierten Module (DI, DQ, analog usw.) genau wie bestückt ein — dies bestimmt das Slot-/Subslot-Layout und die Ein-/Ausgangslängen, die letztlich in der .xdb-Datei landen.



Geräteansicht des IO-Devices: installierte Module (z. B. DI 8x24VDC, DQ 8x24VDC/0.5A)

6. XDB-Generierung an der PC-Station aktivieren

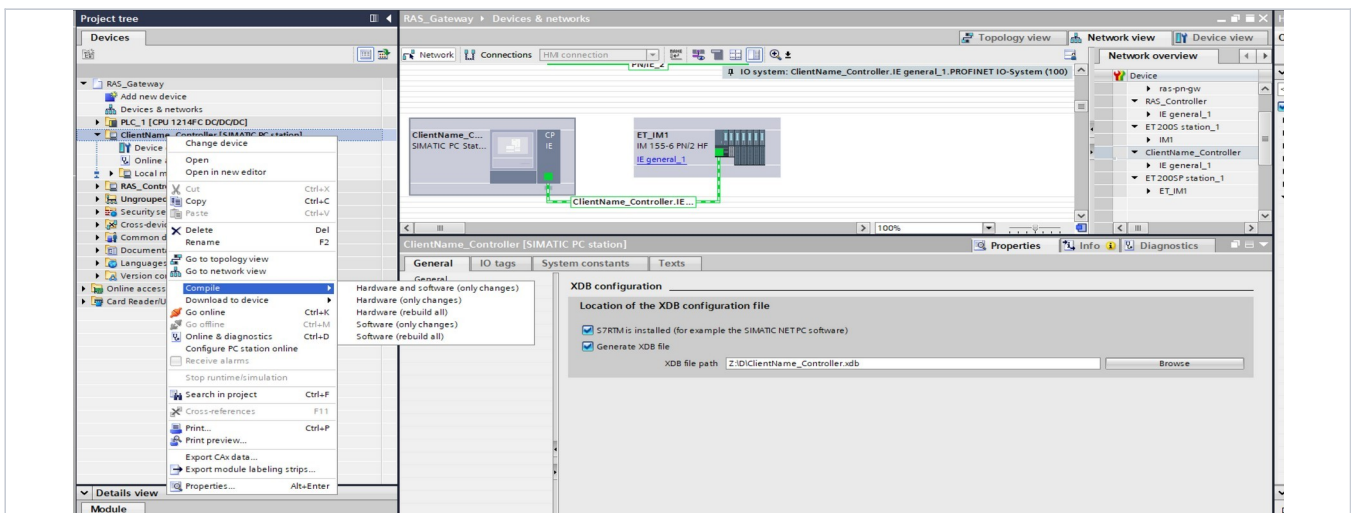
Öffnen Sie an der PC-Station (nicht am IO-Device) Properties → General → XDB configuration, aktivieren Sie Generate XDB file und legen Sie den Speicherpfad fest — einen Ort, der vom Controller-Dienst-PC aus erreichbar ist.



PC-Station → Properties → XDB configuration: „Generate XDB file“ und der Dateipfad

7. Kompilieren, um die .xdb-Datei zu erzeugen

Klicken Sie die PC-Station mit der rechten Maustaste an und wählen Sie Compile → Hardware (rebuild all) (für spätere kleine Änderungen genügt Hardware (only changes)). TIA schreibt die .xdb-Datei während der Kompilierung an den konfigurierten Pfad.



Rechtsklick auf PC-Station → Compile → Hardware (rebuild all)

Tipp: Sind an derselbe PC-Station mehrere IO-Devices angebunden, enthält EINE .xdb-Datei bereits alle — der Import in den Controller fügt alle Geräte in einem Schritt hinzu.

8. Die .xdb in den Controller importieren

Legen Sie die erzeugte Datei im Upload-Bereich der Controller-Weboberfläche ab (§9.3). Die Importantwort listet auf, wie viele Geräte importiert wurden, sowie etwaige Warnungen — z. B. ein Slot, den der Importer nicht vollständig interpretieren konnte.

9.6 Controller-Einstellungen und Netzwerkadapter

Controller-Einstellungen — Stationsname, IP/Subnetz/Gateway und die Option, die IP auch auf den Windows-Netzwerkadapter anzuwenden

Dies ist die eigene Identität des Controllers im PROFINET-Netzwerk: Stationsname und IP/Subnetz/Gateway. Diese IP ändert sich nicht — es ist die Adresse, unter der der Controller DCP und RPC ausführt. Da RPC eine echte Adresse auf der Leitung benötigt, aktivieren Sie „Die IP auch auf den Windows-Netzwerkadapter anwenden“, damit die Adresse auf der gewählten Karte wirklich existiert (als sekundäre Adresse hinzugefügt; bestehende Adressen werden nicht entfernt).

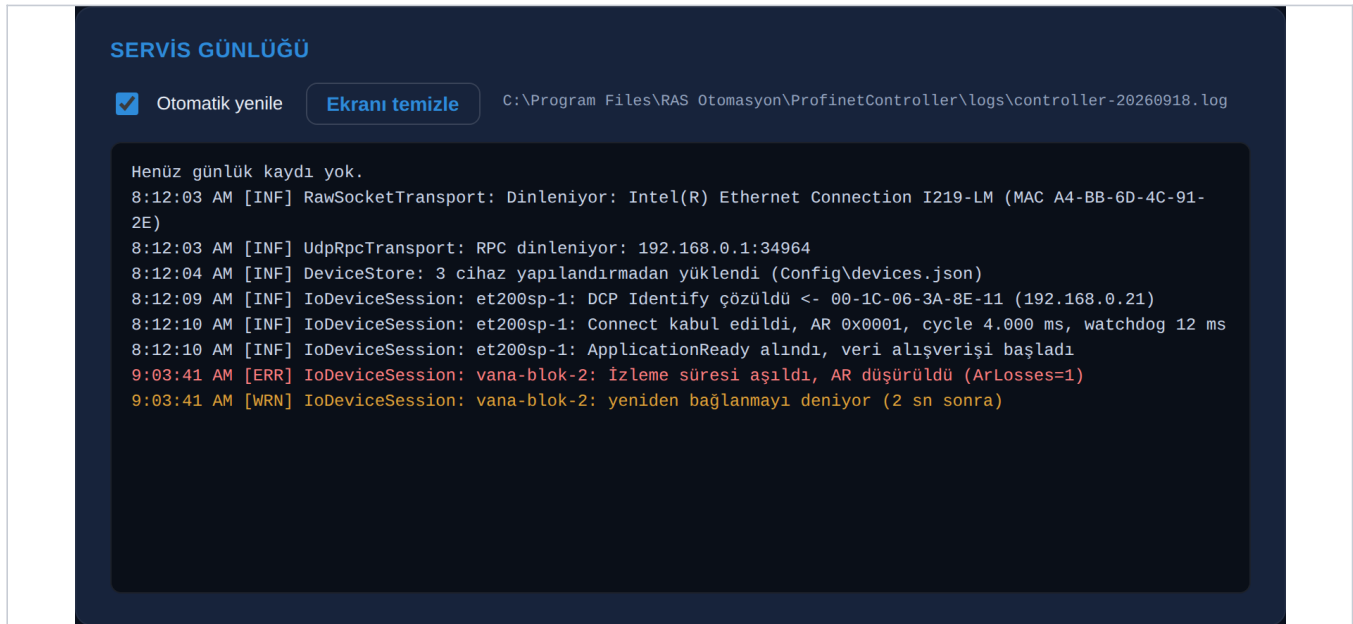
Hinweis: Das Ändern der IP hier startet die PROFINET-Schicht neu — Geräte verbinden sich automatisch neu, aber es kommt kurz zu einem Abbruch. Führen Sie dies in einem geplanten Zeitfenster durch, nicht bei laufender Produktion.

Die PROFINET-Schicht manuell stoppen und starten

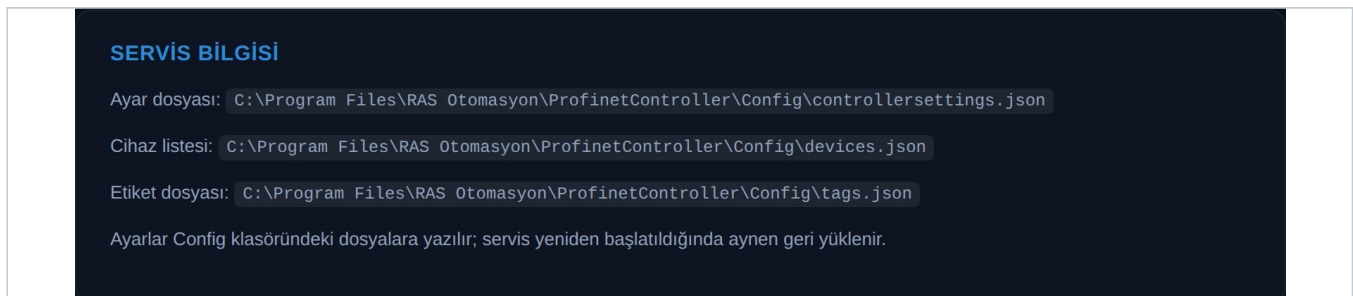
Dieselbe Karte enthält eine einzelne Stoppen/Starten-Umschaltfläche, unabhängig vom Aktiv-Kontrollkästchen (§9.3) eines einzelnen Geräts: Stoppen beendet jede laufende Gerätesitzung und schließt danach den Netzwerkadapter und den RPC-Socket vollständig (POST /api/stop), während Starten beides wieder öffnet und jedes aktivierte Gerät neu verbindet (POST /api/start). Das Bestätigen von Stoppen unterbricht die Verbindung zu allen IO-Devices sofort, und es wird kein automatischer Wiederverbindungsversuch unternommen, bis erneut auf Starten geklickt wird — die Schicht bleibt so lange unten, bis ein Bediener sie bewusst neu startet. Verwenden Sie dies vor dem Umverkabeln oder Aus-/Einschalten des PROFINET-Segments, damit der Controller diese Zeit nicht mit dem erfolglosen Wiederholen von Verbindungsversuchen verbringt.

Warnung: Stoppen ist ein harter Schnitt, keine Pause: Es wartet nicht auf einen sicheren Punkt im Zyklus der IO-Devices. Behandeln Sie es wie das Ziehen des PROFINET-Kabels — stellen Sie vor dem Drücken sicher, dass die Logik der angeschlossenen Anlage einen plötzlichen Ausgangsverlust verträgt.

9.7 Dienstprotokoll und Pfade



Dienstprotokoll-Karte — live scrollende Ansicht von Verbindungsversuchen, DCP-Auflösung, Fehlern und Wiederverbindungen



Dienstinfo-Karte — genaue Pfade zu den Einstellungs-, Geräteliste- und Tag-Dateien auf der Festplatte

Die automatische Aktualisierung fragt das Protokoll alle zwei Sekunden ab; deaktivieren Sie sie, um die Ansicht während des Lesens einzufrieren (dies löscht nur die Bildschirmliste, niemals die Protokolldatei auf der Festplatte). Die Dienstinfo-Karte ist nützlich, wenn Sie die Protokolldatei an eine Support-Anfrage anhängen oder devices.json manuell bearbeiten.

9.8 .NET-Client-Beispiel

```
using RasProLink.Client;

using var client = new RasProLinkClient("localhost", 8080);
var status = await client.ConnectAsync();
Console.WriteLine($"{status.StationName}: {status.DeviceCountRunning}/{status.DeviceCount}
devices running");

var detail = await client.GetDeviceAsync("et200sp-1");
var io = await client.GetDeviceIoAsync("et200sp-1");

await client.WriteDeviceOutputAsync("vana-blok-2", slot: 1, subslot: 1,
    data: new byte[] { 0x01 });

client.DevicesChanged += (s, list) => Console.WriteLine($"{list.Count} devices");
```

```
await client.StartPollingAsync();
```

9.9 Fehlerbehebung

Symptom	Prüfung
Weboberfläche lädt nie	Läuft der Dienst (services.msc)? Wird Port 8080 bereits von etwas anderem verwendet?
Keine Netzwerkadapter aufgelistet	Npcap nicht installiert, oder es wurde nach dem Start des Dienstes installiert — installieren und den Dienst neu starten.
Gerät verlässt Resolving nie	Keine DCP-Identify-Antwort — Verkabelung prüfen und ob der gewählte Adapter (§4.3) wirklich damit verbunden ist.
Gerät wechselt zwischen Läuft → Fehler → Läuft	Überwachungszeit wird erreicht — prüfen Sie den Jitter in der Gerätedetailkarte (§9.4) gegen die konfigurierte Aktualisierungszeit; meist ein ausgelastetes Netzwerksegment.
XDB-Import abgelehnt	Kein Geräte-Stellvertreterexport, oder von einer inkompatiblen TIA-Portal-Version exportiert — erneut exportieren und wiederholen.

9.10 GSDML-Katalog — Geräte ohne TIA Portal oder .xdb hinzufügen

Jedes entdeckte oder aus dem Katalog hinzugefügte Gerät durchläuft weiterhin denselben DCP/RPC/zyklischen Austausch wie in §9.2 — der GSDML-Katalog ändert nur, wie das Slot-/Subslot-Layout eines Geräts in den Controller gelangt, und ersetzt den in §9.3, §9.5 beschriebenen Umweg über TIA Portal + .xdb durch einen Ordner mit den eigenen GSDML-Dateien des Herstellers.

Netzwerksuche (DCP)

Netzwerksuche sendet einen DCP-Identify-All-Broadcast und listet jede antwortende PROFINET-Station auf — IO-Devices von Drittanbietern, andere Controller, sogar die eigene Schnittstelle des Controllers — ohne dass dafür eine XDB oder GSDML nötig ist, ähnlich der Funktion Erreichbare Geräte von TIA Portal. Jede Zeile zeigt Name, IP, MAC und Hersteller-/Geräte-ID des Geräts; die Spalte GSDML-Übereinstimmung zeigt die beste Katalogübereinstimmung für diese Hersteller-/Geräte-ID mit den Ein-/Ausgangsbyte-Summen, oder Keine Katalogübereinstimmung, wenn nichts passt. Ein bereits konfiguriertes Gerät zeigt statt Hinzufügen die Option Neu konfigurieren.

GSDML-Katalogordner

Der Katalog durchsucht im Hintergrund einen Ordner nach .xml-/gsdml-Dateien — standardmäßig Config\profinet-controller-gsdml, neben den anderen auf der Dienstinfo-Karte (§9.7) angezeigten Dateien, automatisch beim ersten Start des Controllers angelegt. Legen Sie die GSDML-Datei(en) des Herstellers dort ab und klicken Sie auf Katalog neu laden (oder warten Sie auf die nächste automatische Neuabastung) — ein Neustart des Dienstes ist nicht nötig. Die GSDML-Katalog-Karte meldet die Dateianzahl, wie viele Gerätedefinitionen geparkt wurden, wann zuletzt geladen wurde, sowie etwaige dateibezogene Parse-Fehler. Ein anderer Ordner kann über dieselbe Karte festgelegt werden (Pfad eingeben, oder Durchsuchen — die Ordnerauswahl funktioniert nur, wenn der Dienst über eine interaktive Desktop-Sitzung verfügt; die manuelle Pfadeingabe funktioniert immer) und bleibt über Neustarts hinweg gespeichert.

Ein Gerät direkt aus dem Katalog hinzufügen

Ein Klick auf Hinzufügen neben einer GSDML-Übereinstimmung — entweder aus den Ergebnissen der Netzwerksuche oder aus der eigenen Geräteliste des Katalogs — öffnet ein mit diesem Gerätetyp vorausgefülltes Formular: Geben Sie den Stationsnamen, IP/Subnetz/Gateway ein und weisen Sie die optionalen Steckmodule des Katalogs Slots zu (Slot 0 ist immer der feste DAP selbst). Beim Absenden wird sofort eine vollständige Gerätekonfiguration erstellt; die Verbindungsparameter, die normalerweise eine .xdb mitbringen würde (AR, IOCRs, Alarm CR), werden mit denselben Standardwerten gefüllt, die der Controller für jedes manuell erstellte Gerät verwendet, sodass sich das neue Gerät genau wie ein per XDB importiertes verbindet. Ein kompaktes Gerät ohne Steckmodule überspringt die Slot-Tabelle vollständig.

Hinweis: Der GSDML-Katalog liefert ausschließlich das Modul-/Slot-Layout — er überschreibt nie ein bereits per .xdb hinzugefügtes Gerät, und beide Wege lassen sich frei mischen: manche Geräte aus einer .xdb importiert, andere direkt aus dem Katalog hinzugefügt, nebeneinander in derselben IO-Devices-Liste (§9.3).

Kapitel 10

PROFINET Device

10.1 Überblick

Die PROFINET-Device-Engine ist das Spiegelbild des Controllers: Sie macht Ihren PC zu einem PROFINET-IO-Device — einem Slave — gegenüber einer echten SPS. Sie simuliert ein Gerät, dessen Modul-/Slot-Layout mit der GSDML-Datei übereinstimmt, die mit Ihrer RasProLink-Device-Lizenz ausgeliefert wird; sobald Ihre Slot-Konfiguration diese GSDML widerspiegelt, ist die PLC-IO-Kommunikation ohne SPS-seitige Programmierung über den normalen Import der Hardwarekonfiguration hinaus sofort einsatzbereit.

Punkt	Wert
Rolle	IO-Device (Slave)
Beschrieben durch	Eine mit der Device-Lizenz ausgelieferte GSDML-Datei (extern zu RasProLink gepflegt, nicht von ihm erzeugt)
Von der SPS erkannt über	DCP (Stationsname, bei Bedarf IP)

10.2 GSDML — was es ist und wie es verwendet wird

GSDML (General Station Description Markup Language) ist der XML-Dialekt, den jedes PROFINET-Gerät zur Selbstbeschreibung veröffentlicht: seine Identität (Vendor ID, Device ID), seinen Modulkatalog und das Ein-/Ausgangsdatenlayout jedes Moduls. Ein SPS-Ingenieur importiert diese eine Datei in TIA Portal (oder ein beliebiges anderes PROFINET-Engineering-Tool), und das Gerät erscheint im Hardwarekatalog genau wie ein echtes Siemens-/Herstellerteil — auf das Netzwerk ziehbar, mit konfigurationsbereiten Modulen.

RasProLink erzeugt oder exportiert keine GSDML-Datei — die Datei ist ein feststehendes Artefakt, das außerhalb des Produkts gepflegt und mit Ihrer RasProLink-Device-Lizenz ausgeliefert wird. Was RasProLink tut: Es lässt Sie eine Slot-Konfiguration aufbauen, die die Modulliste dieser Datei exakt widerspiegeln muss — der untenstehende Modulkatalog ist auf RasProLink-Seite fest vorgegeben und nur nutzbar, wenn er mit dem übereinstimmt, was dieselben Module in der von TIA Portal (oder einem anderen Engineering-Tool) importierten GSDML beschreiben. Eine wirklich neue Modulgröße hinzuzufügen ist keine Aktion auf dem Konfigurationsbildschirm — dazu muss zuerst die GSDML-Datei selbst aktualisiert und erneut in TIA Portal importiert werden; erst danach wird die passende Größe hier auswählbar.

Modul	Eingang / Ausgang
8 Byte In/Out	8 Byte Eingang, 8 Byte Ausgang
16 Byte In/Out	16 Byte Eingang, 16 Byte Ausgang
32 Byte In/Out	32 Byte Eingang, 32 Byte Ausgang
64 Byte In/Out	64 Byte Eingang, 64 Byte Ausgang

Diese vier symmetrischen Module bilden den gesamten Katalog — Slot 0 ist immer der feste DAP selbst, und bis zu 8 Datenslots (Slots 1–8) können jeweils eine dieser vier Größen aufnehmen. Das deckt die meisten Punkt-zu-Punkt-Datenaustauschbedürfnisse sofort ab; wählen Sie für jeden Slot die passende Größe. Benötigt Ihre

Anwendung asymmetrische oder größere Blöcke, ist das eine GSDML-Änderung und muss — wie oben beschrieben — auf GSDML-Ebene beantragt und ausgerollt werden, nicht allein in RasProLink konfiguriert.

10.3 Inbetriebnahme gegenüber einer echten SPS

6. Besorgen Sie die GSDML-Datei für Ihre RasProLink-Device-Lizenz (siehe §10.2) und importieren Sie sie in TIA Portal, falls sie noch nicht im Hardwarekatalog ist: Options → Manage general station description files (GSD).
7. Legen Sie in der Weboberfläche von RasProLink den Stationsnamen des Devices fest und wählen Sie für jeden Slot die Modulgröße, die dem entsprechenden Slot in der importierten GSDML entspricht — siehe Modulkatalog in §10.2.
8. Ziehen Sie im SPS-Projekt das Gerät aus dem Hardwarekatalog auf das Netzwerk, verbinden Sie es mit der PROFINET-Schnittstelle der SPS und weisen Sie denselben Stationsnamen sowie eine IP im Subnetz der SPS zu.
9. Laden Sie die Hardwarekonfiguration in die SPS. Sie löst das Device per DCP anhand des Stationsnamens auf, verbindet sich, und zyklische Daten beginnen zu fließen — sichtbar im eigenen Gerätestatus von RasProLink, genauso wie ein IO-Device beim Controller erscheint (§9.2).

10.4 .NET-Client-Beispiel

```
using RasProLink.Client;

using var client = new RasProLinkClient("localhost", 8080);
await client.ConnectAsync();

// Read what the PLC most recently wrote to us (its outputs, our inputs)
var io = await client.GetDeviceIoAsync("self");

// Publish new input data for the PLC to read next cycle
await client.WriteDeviceOutputAsync("self", slot: 1, subslot: 1,
    data: new byte[] { 0x00, 0x2A });
```

10.5 Fehlerbehebung

Symptom	Wahrscheinliche Ursache
SPS findet das Gerät nie	Stationsname stimmt nicht überein (Groß-/Kleinschreibung beachten, nur DNS-verträgliche Zeichen), oder SPS und Device liegen nicht im selben Subnetz.
Modulkonfliktalarm in der SPS	Das in RasProLink konfigurierte Slot-Layout stimmt nicht mehr mit der in das SPS-Projekt importierten GSDML überein — jeden Slot anhand der Modulliste der GSDML-Datei (§10.2) erneut prüfen und die abweichenden Slots korrigieren.

10.6 Diagnose

Die Diagnose-Karte ist ein Werkzeug für Inbetriebnahme und Verdrahtungsprüfung: Sie können bekannte Testmuster in die Daten einspeisen, die das Device an die SPS sendet, ein einzelnes Tag oder Byte auf einen

festen Wert fixieren, einen Modulfehler simulieren und die laufenden Zyklusstatistiken zurücksetzen — alles ohne das SPS-Programm anzufassen. Diese Werkzeuge beeinflussen, was die SPS tatsächlich empfängt; behandeln Sie sie daher als Labor-/Inbetriebnahmehilfen: Solange ein Force oder ein Testmuster aktiv ist, werden die von Ihrer eigenen Anwendung geschriebenen Werte überschrieben, und RasProLink schaltet sie beim Schließen der Seite nicht automatisch ab.

Testmuster

Um von der SPS-Seite aus zu prüfen, dass Daten wirklich Ende-zu-Ende fließen, wählen Sie ein Muster, einen Slot (oder belassen Sie es beim ersten verfügbaren Slot) und ein Aktualisierungsintervall und wenden es an: Zähler erhöht das erste Byte der Eingangsdaten des Slots bei jedem Intervall; Rampe schreibt Byte $n = \text{Zähler} + n$, was zusätzlich eine Offset-Verschiebung aufdeckt, falls die Zuordnung auf SPS-Seite falsch ist; Wechsel schaltet das gesamte Submodul zwischen 0x00 und 0xFF um; Zufällig schreibt zufällige Bytes und dient rein dazu, die Transparenz des Datenpfads zu bestätigen (kein Byte bleibt unterwegs hängen oder wird maskiert). Es läuft immer nur ein Muster gleichzeitig; ein Wechsel auf Aus stellt die normalen Anwendungsdaten wieder her.

Signal-Force

Force fixiert ein gewähltes Eingangs-Tag auf einen von Ihnen eingegebenen Wert (als Hex), unabhängig vom Testmuster und von dem, was Ihre eigene Anwendung darunter schreibt. Der zugrunde liegende Wert wird nicht gelöscht — er wird weiterhin geschrieben und verfolgt, nur auf der Leitung überschrieben —, sodass das Aufheben des Force (pro Tag oder Alle aufheben) den wahren Wert sofort wieder erscheinen lässt, ohne dass etwas abgeglichen werden muss. Die Tabelle der aktiven Forces listet für jedes Slot, Offset, Tag-Namen und geforcten Wert mit einer Aufheben-Aktion; damit bestätigen Sie, dass ein bestimmtes nachgelagertes Signal (ein Alarmbit, ein Analogwert) das erwartete SPS-Verhalten auslöst, bevor die echte Feldverdrahtung angeschlossen ist.

Simulierter Fehler und Statistik-Reset

„Modulfehler simulieren (IOPS = Bad)“ erzwingt das IOPS-Statusbyte des Submoduls auf Bad, ohne etwas zu trennen — so prüfen Sie, ob das SPS-Programm tatsächlich auf eine schlechte Datenqualitätsanzeige reagiert (z. B. ein Modulfehler-OB, ein Qualitätsbit, das nachgelagerte Logik sperrt), statt blind zu vertrauen, welche Bytes gerade im Prozessabbild stehen. Zum Wiederherstellen des normalen (Good-)Status wieder ausschalten. Statistik zurücksetzen nullt die an anderer Stelle in der Oberfläche angezeigten Zyklusähler (gesendete/empfangene Rahmen, Jitter usw.), was direkt vor einem zeitgesteuerten Testlauf nützlich ist, damit frühere Anlauftransienten die Zahlen nicht verfälschen.

Hinweis: Die Diagnose-Statuskacheln (Geforctes Bytes, Testmuster) oben in der Karte geben einen schnellen Überblick darüber, was gerade überschrieben ist — prüfen Sie diese, bevor Sie eine Station übergeben, damit nichts während der Inbetriebnahme Geforctes in der Produktion aktiv bleibt. Die zugrunde liegenden REST-Endpunkte (GET/POST /api/diagnostics und Unterrouuten) sind in Anhang A aufgeführt.

Kapitel 11

.NET-Clientbibliothek

RasProLink.Host kann in einem einzigen Prozess jede der fünf Engines (EtherCAT Master, EtherNet/IP Scanner, EtherNet/IP Adapter, PROFINET Controller, PROFINET Device) hosten, aber jeweils nur eine ist aktiv — ein Wechsel über die Protokollauswahl der Weboberfläche startet den Prozess mit dieser Engine neu (§4.4). Die REST-Oberfläche der aktiven Engine unterscheidet sich von Engine zu Engine, am deutlichsten darin, wie ein Tag-Schreibvorgang angibt, an welches Ziel (Gerät/Slave/Scanner-Ziel) er geht:

Engine	Tag-Schreib-Endpunkt
EtherCAT Master	POST /api/ethercat/slaves/{id}/tags/write — {id} ist die interne GUID des Slave.
EtherNet/IP Scanner	POST /api/ethernetip/scanner/targets/{id}/tags/write — {id} ist die interne GUID des Ziels.
PROFINET Controller	POST /api/devices/{name}/tags/write — {name} ist der PROFINET-Stationsname, keine GUID.
EtherNet/IP Adapter	POST /api/ethernetip/adapter/tags/write — kein Ziel/keine ID, ein fest vorgegebenes lokales IO-Set.
PROFINET Device	POST /api/io/write-tags — kein Ziel/keine ID, ein fest vorgegebenes lokales IO-Set.

RasProLink.Client verbirgt diesen Unterschied vollständig. Jede gemeinsame Methode — einschließlich WriteTagAsync — behält exakt dieselbe Signatur, unabhängig davon, welche Engine aktiv ist: intern ermittelt ConnectAsync (GET /api/engine), welche Engine aktiv ist, und jeder Aufruf wird auf den echten REST-Pfad dieser Engine übersetzt (die Name→GUID-Zuordnung für EtherCAT/Scanner wird einmal über GetTargetsAsync ermittelt und zwischengespeichert). Eine Zeile wie

```
await client.WriteTagAsync("IM1", "Motor_Run", true);
```

funktioniert unverändert weiter, egal welche Engine RasProLink.Host gerade ausführt (bei EtherNet/IP Adapter und PROFINET Device wird der Zielname "IM1" einfach ignoriert, da diese beiden Engines ein fest vorgegebenes lokales IO-Set haben). Eine engine-spezifische Fähigkeit, die nicht in diesen gemeinsamen Ablauf passt (z. B. roher EtherCAT-SDO-Zugriff, PROFINET-.xdb-Import), wird stattdessen als separate, engine-spezifische Methode hinzugefügt — sie ändert nie das eigene Verhalten von WriteTagAsync/ReadTagsAsync.

Über dieses Design hinaus ist die Bibliothek ein kleines, abhängigkeitsfreies Paket, das Ihre eigene .NET-Anwendung referenziert, um sich mit dem Dienst zu verbinden, Engine-Status und Ziele zu lesen und im Code Tags zu lesen/schreiben — anstatt die Weboberfläche zu bedienen. Sie wird als zwei DLLs ausgeliefert:

Assembly	Inhalt
RasProLink.Client.Abstractions.dll	Gemeinsame, engine-unabhängige Datentypen (EngineKind, EngineStatus, TargetSummary, TagDefinition, TagValue ...) und ein kleiner, eigenständiger JSON-Lese-/Schreibmechanismus.
RasProLink.Client.dll	Der eigentliche Client (RasProLinkClient), aufgebaut auf HttpClient.

Hinweis: Keine der beiden DLLs hat eine NuGet-Abhängigkeit, und beide zielen auf netstandard2.0 und net8.0 ab — dieselbe Bibliothek funktioniert unverändert von einer alten .NET-Framework-4.7.2-WinForms-Anwendung bis zu einem modernen .NET-8-Dienst, ohne Binding-Redirect-Überraschungen.

11.1 Einrichtung und Schnellstart

Referenzieren Sie beide DLLs aus Ihrem Projekt (kein NuGet-Feed erforderlich — einfache Dateiverweise, oder kompilieren Sie die Projekte aus dem Quellcode und fügen Sie eine Projektreferenz hinzu).

```
using RasProLink.Client;
using RasProLink.Client.Abstractions;

using var client = new RasProLinkClient("localhost", 8080);

var active = await client.ConnectAsync();           // GET /api/engine — which engine
is active
Console.WriteLine($"Active engine: {active}");

var status = await client.RefreshStatusAsync();
Console.WriteLine($"{status.RoleDisplayName}: enabled={status.Enabled}
connected={status.Connected}");

var targets = await client.GetTargetsAsync();
foreach (var t in targets)
    Console.WriteLine($"{t.Name} @ {t.IpAddress} [{t.State}] in={t.InputBytes}B
out={t.OutputBytes}B");
```

ConnectAsync muss nicht explizit aufgerufen werden — jede andere Methode ruft es bei Bedarf beim ersten Gebrauch automatisch auf. Es selbst aufzurufen ist nur nützlich, um die aktive Engine vorab zu kennen/protokollieren; danach lesen IsConnected und ActiveEngine das zwischengespeicherte Ergebnis, ohne eine weitere Anfrage.

11.2 Engine-Status und Ziele

```
var info = await client.GetEngineInfoAsync();       // full engine-selection list, not
just the active one
Console.WriteLine($"Active: {info.Active}");
foreach (var opt in info.Available)
    Console.WriteLine($" {opt.DisplayName} ({opt.Kind})
productionReady={opt.ProductionReady}");

var status = await client.RefreshStatusAsync();     // normalized snapshot of the
ACTIVE engine
Console.WriteLine(status.Message ?? "(no message)");
```

GetEngineInfoAsync liefert dieselbe Liste, die die Protokollauswahl der Weboberfläche zeigt (§4.4) — jede Engine, zu der RasProLink.Host wechseln könnte, nicht nur die aktive; nutzen Sie es, um eine eigene Engine-Auswahl zu bauen oder ProductionReady zu prüfen, bevor ein Wechsel angeboten wird. RefreshStatusAsync dagegen beschreibt immer nur die derzeit aktive Engine, normalisiert in die gemeinsame EngineStatus-Form: Enabled zeigt, ob die Engine bewusst gestoppt wurde (siehe §11.8); Connected bedeutet je nach Engine etwas anderes (Bus/PROFINET-Schicht aktiv bei EtherCAT/PROFINET, eine bestehende Scanner-Verbindung beim

Adapter) — es ist eine Zusammenfassung der gesamten Engine, nicht eines einzelnen Ziels. Der Status pro Ziel steht stattdessen in `TargetSummary.State`.

`GetTargetsAsync` liefert die Geräte-/Slave-/Scanner-Ziel-Liste der Engine, normalisiert in `TargetSummary` — Name ist der Wert, den Sie als `target`-Argument an die Tag-Methoden unten übergeben. EtherNet/IP Adapter und PROFINET Device haben keine echte Ziel-Liste (ein fest vorgegebenes lokales IO-Set), daher liefert `GetTargetsAsync` für sie stattdessen eine einzelne synthetische Zeile.

11.3 Tags schreiben

```
// "IM1" is a target Name from GetTargetsAsync (§11.2). On EtherNet/IP Adapter
// and PROFINET Device it is accepted but ignored – there is only one target.
await client.WriteTagAsync("IM1", "Motor_Run", true);

// Write several output tags on one target in a single round trip
await client.WriteTagsAsync("IM1", new Dictionary<string, object?>
{
    ["Motor_Run"] = true,
    ["Speed_Setpoint"] = 1500,
});
```

`WriteTagAsync` schreibt ein einzelnes Tag; es ist ein dünner Wrapper über `WriteTagsAsync` mit einem einzigen Eintrag. Bevorzugen Sie `WriteTagsAsync`, wenn Sie mehr als ein Tag auf demselben Ziel setzen — eine Anfrage statt mehrerer bedeutet sowohl weniger Netzwerkverkehr als auch kein Risiko, dass die Werte in unterschiedlichen Engine-Zyklen landen. Es dürfen nur Tags mit Ausgangsrichtung geschrieben werden, genau wie die eigene Regel der Weboberfläche (§5.3) — das Schreiben eines Eingangs-Tags oder eines für dieses Ziel nicht existierenden Tags wirft eine `RasProLinkClientException` (§11.6).

11.4 Live-Updates — Polling

```
client.StatusChanged += (s, status) => Console.WriteLine($"status: {status}");
client.TargetsChanged += (s, list) => Console.WriteLine($"{list.Count} targets");
client.PollError += (s, ex) => Console.WriteLine($"poll error: {ex.Message}");

client.PollIntervalMs = 500; // default 1000
await client.StartPollingAsync();
// ...
client.StopPolling();
```

Der Dienst bietet nur REST, keinen Push-Kanal, daher pollt die Clientbibliothek: `StartPollingAsync` lässt eine Hintergrundschleife laufen, die in Ihrem Namen alle `PollIntervalMs` `RefreshStatusAsync` und `GetTargetsAsync` aufruft und nach jeder Runde `StatusChanged` und `TargetsChanged` auslöst — dieselbe Information, nach der auch die Weboberfläche selbst pollt. Eine fehlgeschlagene Runde (Dienst kurzzeitig nicht erreichbar) löst `PollError` aus, statt die Schleife zu stoppen; das Polling versucht es beim nächsten Tick erneut. Tag-Werte sind nicht Teil des Pollings — rufen Sie `ReadTagsAsync` selbst auf, in dem Rhythmus, den Ihre Tags brauchen, da welche Tags wichtig sind und wie oft anwendungsspezifisch ist.

11.5 Tags aus dem Code

```
var defs = await client.GetTagDefinitionsAsync("IM1");
```

```
await client.AddOrUpdateTagAsync("IM1", new TagDefinition
{
    Name = "Sensor_Input1",
    Direction = IoDirection.Input,
    Type = TagDataType.Bool,
    Offset = 0,
    BitOffset = 0
});

await client.RemoveTagAsync("IM1", "Obsolete_Tag");

// One round trip fills a local table for GetTag/ReadTag<T> below
var values = await client.ReadTagsAsync("IM1");

bool running = client.ReadTag<bool>("IM1", "Motor_Run");
TagValue? raw = client.GetTag("IM1", "Sensor_Input1");
Console.WriteLine(raw?.Text);
```

GetTagDefinitionsAsync, AddOrUpdateTagAsync, SetTagDefinitionsAsync (ersetzt in einem Aufruf die gesamte Tag-Tabelle eines Ziels) und RemoveTagAsync spiegeln den Tag-Editor der Weboberfläche (§5.3) wider — Slot/Subslot sind nur bei den PROFINET-Engines relevant und werden andernorts bei ihren Standardwerten (0/1) belassen.

ReadTagsAsync ist der einzige Aufruf, der für Werte das Netzwerk erreicht; er füllt eine lokale, nach Ziel und Tag-Name indizierte Tabelle. GetTag und das generische ReadTag<T> lesen dann aus dieser lokalen Tabelle — sie stellen nie eine Anfrage, rufen Sie also ReadTagsAsync erneut auf, wann immer Sie einen aktuelleren Wert benötigen. ReadTag<T> wandelt die von TagValue gespeicherte Zahl/den Text/die Rohbytes in bool, string, byte[] oder einen numerischen Typ um; für Int64/UInt64-Tags lesen Sie stattdessen direkt TagValue.Raw, da ein double den vollen 64-Bit-Bereich nicht verlustfrei fassen kann.

11.6 Fehlerbehandlung

Jeder fehlgeschlagene Aufruf wirft eine RasProLinkClientException. Antwortet der Dienst mit einem strukturierten Fehlertext, ist dessen Meldung der tatsächliche Grund; kann ein an eine Tag-Methode übergebener Zielname nicht aufgelöst werden (nur EtherCAT Master / EtherNet/IP Scanner — siehe §11.2), meldet der Client dies direkt, ohne den Dienst zu kontaktieren; ist der Dienst überhaupt nicht erreichbar, sagt die Meldung dies und umschließt die zugrunde liegende Transportausnahme.

11.7 Ohne die Bibliothek — einfaches HTTP

Die Bibliothek ist ein dünner Wrapper — jede Operation ist ein einfacher HTTP-Aufruf, aus jeder Sprache erreichbar. Vollständige Endpunktliste in Anhang A; hier derselbe „Geräte auflisten“-Aufruf in curl und Python für einen PROFINET Controller — der zugrunde liegende Pfad unterscheidet sich bei den anderen vier Engines genau so, wie am Anfang dieses Kapitels zusammengefasst.

```
curl http://localhost:8080/api/devices
```

```
import requests
devices = requests.get("http://localhost:8080/api/devices").json()
for d in devices:
    print(d["name"], d["ipAddress"], d["state"])
```

Der mit RasProLink ausgelieferte examples-Ordner enthält einen vollständigen 16-Kanal-Walking-Bit-Test-Client in fünf Sprachen — C#, VB.NET, Python, PowerShell und Node.js — jeweils eigenständig mit einer eigenen Kopie der Clientbibliothek, bereit zum eigenständigen Erstellen und Ausführen.

11.8 Engine stoppen und starten

```
await client.StopAsync(); // same effect as the web interface's Stop control – halts ALL
targets
// ...
await client.StartAsync(); // resumes it
```

StopAsync und StartAsync sind das clientseitige Gegenstück zur Stopp/Start-Steuerung, die auf der eigenen Statusseite der aktiven Engine in der Weboberfläche angezeigt wird — sie halten oder setzen die gesamte aktive Engine fort, alle ihre Ziele auf einmal, nicht ein einzelnes Tag oder Ziel. Nach StopAsync wird EngineStatus.Enabled von RefreshStatusAsync false; auch Connected sinkt, da nun kein zyklischer Datenaustausch mehr stattfindet. Es gibt keine Client-Methode, um ein einzelnes Ziel zu stoppen, zu deaktivieren oder zu entfernen — diese Granularität wird über den eigenen Geräte-/Ziel-Editor jeder Engine in der Weboberfläche konfiguriert, nicht über diese Bibliothek.

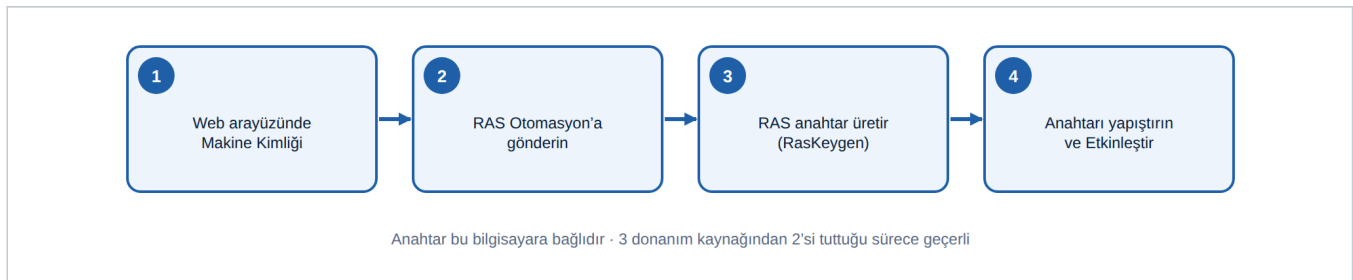
Kapitel 12

Lizenzierung

12.1 Testphase

RasProLink läuft auf jeder der fünf Engines 30 Tage lang ab dem ersten Start ohne eingegebene Lizenz voll funktionsfähig — jede Engine, jedes Gerät, unbegrenzt. Endet die Testphase ohne Schlüssel, stoppt die aktive Protokollschicht (Geräte trennen sich), aber die Weboberfläche funktioniert weiter, sodass noch ein Schlüssel eingefügt werden kann.

12.2 Lizenz beziehen und aktivieren



Der Schlüssel ist an diesen Computer gebunden — gültig, solange 2 von 3 Hardware-Fingerabdrücken übereinstimmen

10. Öffnen Sie die Lizenzkarte (§4.2) und kopieren Sie die Maschinen-ID.
11. Senden Sie sie zusammen mit Ihrer Bestell-/Kundenreferenz an RAS Automation.
12. RAS Automation stellt einen für diese Maschinen-ID signierten Schlüssel aus (über RasKeygen — ein internes Tool, das RAS Automation nie verlässt).
13. Fügen Sie den Schlüssel in die Lizenzkarte ein und klicken Sie auf Aktivieren.

Eine Lizenz kann über dieselbe Lizenzkarte oder per DELETE /api/license auch wieder entfernt werden — dies löscht den gespeicherten Schlüssel und versetzt den Dienst zurück in den Testphasenstatus (§12.1). Dies dient Support- und Übergabefällen (z. B. Übergabe des PCs an eine andere Person) und ist für die routinemäßige Aktivierung nicht erforderlich.

12.3 Woran die Lizenz gebunden ist

Ein Schlüssel ist ein Fingerabdruck über mehrere unabhängige Hardware-/Betriebssystem-Kennungen des jeweiligen PCs. Er bleibt gültig, solange 2 von 3 dieser Quellen noch übereinstimmen — tolerant gegenüber einer routinemäßigen Änderung (etwa einer alleinigen Neuinstallation des Betriebssystems), ohne pauschal auf eine andere Maschine übertragbar zu sein.

Eine Lizenz aktiviert jede Engine auf diesem PC — es gibt keine protokollbezogene Lizenz: eine einmalige Aktivierung auf der Maschinen-ID deckt EtherCAT Master, EtherNet/IP Scanner, EtherNet/IP Adapter, PROFINET Controller und PROFINET Device gleichermaßen ab, unabhängig davon, welche Engine aktiv geschaltet ist.

12.4 Lizenzfehlermeldungen

Meldung	Bedeutung
Schlüssel gehört nicht zu dieser Maschine	Der Schlüssel wurde für eine andere Maschinen-ID erstellt — prüfen Sie

Meldung	Bedeutung
	erneut, was bei der Anfrage übermittelt wurde.
Lizenzdatei konnte nicht geschrieben werden	Das Dienstkonto kann nicht in seinen Config-Ordner schreiben — prüfen Sie die Ordnerberechtigungen.
Leerer Schlüssel	Vor dem Klick auf Aktivieren wurde nichts in das Feld eingefügt.

Kapitel 13

Leistung und Echtzeitverhalten

13.1 Warum es wichtig ist

Windows ist kein Echtzeitbetriebssystem: Eine naive zyklische Schleife auf Basis eines generischen Timers kann unter normaler Desktop-Zeitplanung um mehrere Millisekunden von ihrer beabsichtigten Periode abweichen — oft mehr als die Zykluszeit selbst bei einem schnellen Feldbus. Die fünf Engines von RasProLink teilen sich eine gemeinsame, auf jede von ihnen identisch angewendete Reihe von Techniken, um diese Abweichung auf gewöhnlicher PC-Hardware auf einen Bruchteil der Zykluszeit zu senken — kein Echtzeit-Betriebssystem-Patch oder dedizierte Hardware-Timerkarte erforderlich.

13.2 Die vier Techniken

- Dedizierter Thread mit hoher Priorität — der zyklische Austausch läuft auf einem eigenen Thread mit Höchster Priorität, statt sich den Thread-Pool mit allem anderen zu teilen, was der Prozess tut.
- Windows-Multimedia-Timer-Auflösung — der systemweite Scheduler-Tick wird für die Lebensdauer der zyklischen Schleife vorübergehend auf 1 ms angehoben (vom Standard $\approx 15,6$ ms), sodass ein grobes Warten tatsächlich nahe am beabsichtigten Zeitpunkt aufwacht.
- CPU-Kernaffinität — der zyklische Thread wird auf den letzten logischen CPU-Kern festgelegt, sodass das Betriebssystem ihn nie mitten im Zyklus verschiebt; jede Verschiebung setzt Cache-/TLB-Zustand zurück und kann genau dann, wenn Timing am wichtigsten ist, mehrere hundert Mikrosekunden Jitter hinzufügen.
- Keine Heap-Zuweisung pro Zyklus — IO-Puffer und Gerätelisten werden einmal zugewiesen und in jedem Zyklus wiederverwendet (an Ort und Stelle geleert und neu geschrieben), wodurch Garbage-Collector-Pausen aus dem zeitkritischen Pfad entfernt werden.

13.3 Was zu erwarten ist

Auf einem typischen Windows-PC mit diesen vier angewendeten Techniken wurde der EtherCAT-Master-Jitter bei einem 4-ms-Zyklus mit maximal etwa 0,3 ms gemessen — gegenüber 9–10 ms vor der Feinabstimmung. Die Ergebnisse hängen vom jeweiligen PC ab (CPU-Generation, Kernanzahl, Energiesparplan) und davon, was sonst noch läuft, aber dieselben vier Techniken gelten identisch für alle fünf Engines, sodass eine vergleichbare Verbesserung auch bei PROFINET- und EtherNet/IP-Zyklen zu erwarten ist.

13.4 Jitter in der Oberfläche ablesen

Der Zyklus-Jitter, angezeigt in den Statistiken jedes Geräts (§9.4 für PROFINET Controller; dasselbe Feld existiert in den Gerätedetails jeder Engine), misst, wie weit die tatsächliche zyklische Zeitsteuerung eines Geräts von der konfigurierten Aktualisierungszeit abgewichen ist. Nähert sich der Jitter dem konfigurierten Watchdog/Timeout, wird ein gelegentlicher Verbindungsabbruch wahrscheinlicher — die Lösung ist meist eine längere Aktualisierungszeit an diesem Gerät (mehr Spielraum pro Zyklus) oder die Reduzierung von Überlastung im feldseitigen Netzwerksegment (ein hub-artiger Shared-Switch mit weniger Geräten, oder ein Wechsel zu einem ordentlich QoS-fähigen Switch).

Kapitel 14

Allgemeine Fehlerbehebung

Protokollspezifische Probleme werden im jeweiligen Kapitel jeder Engine behandelt (§6.5, §7.4, §8.4, §9.9, §10.5). Die folgenden Punkte gelten für alle fünf Engines gemeinsam.

Symptom	Prüfung
Weboberfläche von einem anderen PC aus nicht erreichbar	Windows-Firewall blockiert eingehendes TCP 8080 — zulassen (§3.3).
Engine-Wechsel scheint hängen zu bleiben	Der Dienst startet bei einem Wechsel intern neu — einige Sekunden warten, dann aktualisieren; dauert es länger als etwa 30 s, services.msc prüfen.
Lizenz meldet abgelaufene Testphase direkt nach einer sauberen Neuinstallation	uninstall.bat wurde mit -RemoveConfig ausgeführt und hat den Testzähler zusammen mit allem anderen gelöscht — wenden Sie sich statt einer erneuten Testphase an RAS Automation für eine Lizenz.
Diagnosedaten an eine Support-Anfrage anhängen	Verwenden Sie die Pfade der Dienstinfo-Karte (§9.7), um die aktuelle Protokolldatei und devices.json anzuhängen.

Anhang A

HTTP-API-Referenz

Jeder Endpunkt unten ist ein einfacher JSON-REST-Aufruf unter der Basis-URL der Engine (standardmäßig `http://<pc>:8080`), mit camelCase-Feldnamen und ohne integrierte Authentifizierung — der Dienst ist für den Betrieb in einem vertrauenswürdigen lokalen Netzwerk konzipiert. Muss er aus einem nicht vertrauenswürdigen Netzwerk erreichbar sein, stellen Sie ihn hinter einen Reverse-Proxy, der eine Authentifizierung hinzufügt. Die Endpunktmenge ist über Neustarts und Engine-Wechsel hinweg identisch; nur der Abschnitt Geräte unterscheidet sich je Engine entsprechend dem eigenen Gerätemodell des jeweiligen Protokolls.

A.1 Gemeinsame Endpunkte — alle fünf Engines

Endpunkt	Zweck
GET /api/status	Engine-Identität, aktives Protokoll und zusammenfassende Zähler.
GET /api/host/info	Produktname, Build-Version, welche Engine aktiv und produktionsreif ist, Startzeit des Dienstes — unabhängig davon, welche Engine läuft (§4).
GET /api/engine POST /api/engine	Alle fünf Engines mit ihrer Produktionsreife auflisten, oder die aktive wechseln; ein Wechsel speichert die Wahl in <code>enginesettings.json</code> und startet den Dienst innerhalb weniger Sekunden mit der neuen Engine neu (§4).
GET/POST /api/settings/language	Die UI-/Protokollsprache (tr/en/de) lesen oder setzen; auf jeder Engine identisch vorhanden und synchron mit der in der eigenen Konfigurationsdatei der Engine gespeicherten Einstellung gehalten.
GET /api/nics	Netzwerkadapter auflisten, den ausgewählten anzeigen.
POST /api/nics	Einen Netzwerkadapter auswählen (per MAC, optional validiert).
GET /api/health	Einfache Lebendigkeitsprüfung — <code>{"status":"ok"}</code> .
GET/POST/DELETE /api/license	Lizenzstatus lesen, einen Schlüssel aktivieren oder entfernen. Eine Lizenz deckt alle fünf Engines ab (§12.3).
GET /api/logs?after={id}	Dienstprotokoll-Einträge, die neuer als die angegebene ID sind.
GET /api/paths	Konfigurations- und Protokolldateipfade auf der Festplatte (§B).
GET /api/devices/{name}/tags PUT /api/devices/{name}/tags POST /api/devices/{name}/tags DELETE /api/devices/{name}/tags/{tagName} GET /api/devices/{name}/tags/values POST /api/devices/{name}/tags/write	Tags auflisten / vollständig ersetzen / hinzufügen-oder-aktualisieren / entfernen; aktuelle Werte lesen; einen oder mehrere Tags mit Ausgangsrichtung schreiben (§5.3).

Hinweis zu `/api/logs` und `/api/paths` oben: Nur PROFINET Controller und PROFINET Device stellen sie unter diesen nackten Pfaden bereit. EtherCAT Master und beide EtherNet/IP-Rollen stellen dieselben Informationen stattdessen unter einer Engine-präfixierten Route bereit — `/api/ethercat/logs` und `/api/ethercat/paths` bei

EtherCAT Master (§A.2), /api/ethernetip/logs und /api/ethernetip/paths bei sowohl EtherNet/IP Scanner als auch Adapter (§A.3, §A.4). PROFINET Device hat zudem überhaupt keinen /api/paths-Endpunkt — nur /api/logs.

A.2 EtherCAT Master — Geräte

Jede Route in diesem Abschnitt liegt unter dem Präfix /api/ethercat/ — die EtherCAT-Engine verwendet nicht die generische /api/devices-Form.

Endpunkt	Zweck
GET /api/ethercat/status GET /api/ethercat/bus	Zusammenfassung des Engine-Status (läuft/gestoppt, Meldung), oder der vollständige Bus-Snapshot (NIC, Slaves, Zykluszeit).
POST /api/ethercat/start POST /api/ethercat/stop	Den EtherCAT-Bus/die Slave-Kommunikation manuell starten oder stoppen — die Weboberfläche fragt vor dem Stoppen nach Bestätigung (§6).
GET /api/ethercat/nics GET /api/ethercat/nic POST /api/ethercat/nic	Verfügbare Netzwerkadapter auflisten; lesen oder auswählen, welcher vom Ring verwendet wird (per Name, mit optionalem MAC-Override) (§6).
POST /api/ethercat/bus/settings	Das zyklische Austauschintervall (Zykluszeit, Millisekunden) festlegen (§6).
GET /api/ethercat/slaves PUT /api/ethercat/slaves DELETE /api/ethercat/slaves/{id}	Die konfigurierten Slaves auflisten, einen hinzufügen-oder-aktualisieren (Upsert), oder einen aus der Konfiguration entfernen.
GET /api/ethercat/slaves/status	Live-Status je Slave: ESM-Status und Statistiken für jeden konfigurierten Slave.
GET /api/ethercat/slaves/{id}/input POST /api/ethercat/slaves/{id}/output	Aktuelle rohe Eingangs-PDO-Bytes des Slaves lesen (hex), oder dessen Ausgangs-PDO-Bytes schreiben.
GET /api/ethercat/slaves/{id}/tags PUT /api/ethercat/slaves/{id}/tags POST /api/ethercat/slaves/{id}/tags DELETE /api/ethercat/slaves/{id}/tags/{tagName} PUT /api/ethercat/slaves/{id}/tags/{tagName} GET /api/ethercat/slaves/{id}/tags/values POST /api/ethercat/slaves/{id}/tags/write	Die Tags eines Slaves auflisten / vollständig ersetzen / hinzufügen-oder-aktualisieren; ein Tag entfernen oder umbenennen-und-aktualisieren; aktuelle Werte lesen; einen oder mehrere Tags mit Ausgangsrichtung schreiben (§5.3).
POST /api/ethercat/slaves/{id}/sdo/read POST /api/ethercat/slaves/{id}/sdo/write	Einen CoE-Objektverzeichniseintrag (Index/Subindex, Hex-Nutzlast) lesen oder schreiben — das für Startparameter und Diagnose verwendete SDO-Werkzeug (§6).
GET /api/ethercat/esi-catalog POST /api/ethercat/esi-catalog/reload POST /api/ethercat/esi-catalog/directory POST /api/ethercat/esi-catalog/browse-folder	Überwachten ESI-Ordner, Anzahl indizierter Signaturen und letzte Neuladezeit melden; Neuscan erzwingen; den überwachten Ordner ändern; eine Ordnerauswahl öffnen (nur interaktive Desktop-Sitzungen) — der ESI-Ketten-Builder (§6).
GET /api/ethercat/logs GET /api/ethercat/paths	Engine-präfixierte Entsprechungen der in §A.1 beschriebenen gemeinsamen /api/logs und /api/paths, einschließlich des ESI-Katalogordners.

A.3 EtherNet/IP Scanner — Geräte

Jede Route in diesem Abschnitt liegt unter dem Präfix `/api/ethernetip/scanner/` (Target-Verwaltung, Verbindungen, IO) oder `/api/ethernetip/` (Discovery, EDS-Katalog) — die Scanner-Engine verwendet nicht die generische `/api/devices`-Form.

Endpunkt	Zweck
GET <code>/api/ethernetip/status</code>	Rollenidentität und Statusmeldung ("EtherNet/IP Scanner", läuft).
GET <code>/api/ethernetip/scanner/targets</code> PUT <code>/api/ethernetip/scanner/targets</code> DELETE <code>/api/ethernetip/scanner/targets/{id}</code>	Konfigurierte Targets auflisten; eines hinzufügen-oder-aktualisieren (IP, T→O-/O→T-Assembly-Instanzen und -Größen, RPI — §7.2); eines entfernen, wobei zuerst <code>Forward_Close</code> gesendet wird.
GET <code>/api/ethernetip/scanner/connections</code> GET <code>/api/ethernetip/scanner/engine-status</code>	Verbindungsstatus, RPI, Assembly-Größen und Statistiken je Target; oder das Enabled/Disabled-Flag der gesamten Engine (getrennt von der Verbindung eines einzelnen Targets).
POST <code>/api/ethernetip/scanner/stop</code> POST <code>/api/ethernetip/scanner/start</code>	Die Kommunikation mit jedem konfigurierten Target manuell stoppen oder starten.
GET <code>/api/ethernetip/scanner/targets/{id}/input</code> POST <code>/api/ethernetip/scanner/targets/{id}/output</code>	Aktuelle O→T-Assembly-Bytes des Targets lesen (hex), oder T→O-Assembly-Bytes schreiben.
POST <code>/api/ethernetip/scanner/discover?seconds={n}</code>	Eine UDP-ListIdentity-Discovery für 0,5-10 s senden; jedes gefundene Gerät wird mit dem EDS-Katalog abgeglichen, um nach Möglichkeit ein "Als Target hinzufügen"-Formular vorzufüllen.
GET <code>/api/ethernetip/eds-catalog</code> POST <code>/api/ethernetip/eds-catalog/reload</code> POST <code>/api/ethernetip/eds-catalog/directory</code> POST <code>/api/ethernetip/eds-catalog/browse-folder</code> POST <code>/api/ethernetip/eds/parse</code>	Status des Katalogordners; Neuscan erzwingen; den überwachten Ordner ändern; eine Ordnerauswahl öffnen (nur interaktive Desktop-Sitzungen); oder die Assemblies einer einzelnen hochgeladenen .eds-Datei direkt auswerten (ohne Katalogordner).
GET <code>/api/ethernetip/scanner/targets/{id}/tags</code> PUT <code>/api/ethernetip/scanner/targets/{id}/tags</code> POST <code>/api/ethernetip/scanner/targets/{id}/tags</code> DELETE <code>/api/ethernetip/scanner/targets/{id}/tags/{tagName}</code> PUT <code>/api/ethernetip/scanner/targets/{id}/tags/{tagName}</code> GET <code>/api/ethernetip/scanner/targets/{id}/tags/values</code> POST <code>/api/ethernetip/scanner/targets/{id}/tags/write</code>	Die Tags eines Targets auflisten / vollständig ersetzen / hinzufügen-oder-aktualisieren; ein Tag entfernen oder umbenennen-und-aktualisieren; aktuelle Werte lesen; einen oder mehrere Tags mit Ausgangsrichtung schreiben (§5.3).

A.4 EtherNet/IP Adapter — Assemblies

Jede Route in diesem Abschnitt liegt unter dem Präfix `/api/ethernetip/adapter/`. Anders als der Scanner hat der Adapter keinen Endpunkt, der verbundene Scanner auflistet — es gibt genau eine Input-/Output-/Config-Assembly-Menge, daher wird sein eigener Status (läuft/gestoppt, letzte Aktivität) stattdessen von `/api/ethernetip/adapter/status` gelesen.

Endpunkt	Zweck
GET /api/ethernetip/status GET /api/ethernetip/adapter/status	Rollenidentität/Statusmeldung, oder Adapter-spezifischer Status (ob aktuell ein Scanner verbunden ist, letzte Aktivität).
GET /api/ethernetip/adapter/config PUT /api/ethernetip/adapter/config	Die lokalen Input-/Output-/Config-Assembly-Instanzen und -Größen lesen oder festlegen, die diese Engine einem Scanner anbietet (§8.2).
POST /api/ethernetip/adapter/stop POST /api/ethernetip/adapter/start	Die TCP/UDP-Listener, die die Verbindung eines Scanners annehmen, manuell stoppen oder starten.
GET /api/ethernetip/adapter/output POST /api/ethernetip/adapter/input	Die zuletzt von einem Scanner empfangenen Output-Assembly-Bytes lesen, oder die Input-Assembly-Bytes schreiben (Banktest ohne angeschlossenen echten Scanner).
GET /api/ethernetip/adapter/tags PUT /api/ethernetip/adapter/tags POST /api/ethernetip/adapter/tags DELETE /api/ethernetip/adapter/tags/{tagName} PUT /api/ethernetip/adapter/tags/{tagName} GET /api/ethernetip/adapter/tags/values POST /api/ethernetip/adapter/tags/write	Den (einzigen) Tag-Satz dieses Adapters auflisten / vollständig ersetzen / hinzufügen-oder-aktualisieren; ein Tag entfernen oder umbenennen-und-aktualisieren; aktuelle Werte lesen; einen oder mehrere Tags mit Ausgangsrichtung (T→O) schreiben (§5.3).

A.5 PROFINET Controller — Geräte

Endpunkt	Zweck
GET /api/devices	Alle IO-Devices mit Statuszusammenfassung auflisten.
GET /api/devices/{name}	Volldetail: Slots, Subslots, Statistiken.
POST /api/stop POST /api/start	Die PROFINET-Protokollschicht für alle IO-Devices auf einmal manuell stoppen oder starten — die Weboberfläche fragt vor dem Stoppen nach Bestätigung (§9.6).
POST /api/ip	Den eigenen Stationsnamen und die IP-/Subnetz-/Gateway-Einstellungen des Controllers anwenden; startet die PROFINET-Schicht neu, damit sie wirksam werden.
GET /api/devices/{name}/io POST /api/devices/{name}/io	Aktuelle rohe IO-Bytes je Submodul (hex); auf ein Ausgangs-Submodul schreiben.
POST /api/devices/{name}/enabled	Ein Gerät aktivieren/deaktivieren.
DELETE /api/devices/{name}	Ein Gerät aus der Konfiguration entfernen.
POST /api/devices/xdm	Eine .xdm-Geräte-Stellvertreterdatei importieren (multipart/form-data) — fügt alle darin enthaltenen Geräte in einem Schritt hinzu (§9.3).
POST /api/devices/discover	DCP-Netzwerksuche — listet jede gefundene PROFINET-Station mit ihrer automatischen GSDML-Katalogübereinstimmung, falls vorhanden (§9.10).
GET /api/gsdml-catalog GET /api/gsdml-catalog/devices GET /api/gsdml-catalog/devices/{vendorId}/{deviceId}/{dapId}/m	Status des Katalogordners (Datei-/Geräteanzahl, letzte Suche, Fehler); alle DAPs im Katalog auflisten; die Steckmodule eines DAP auflisten (§9.10).

Endpunkt	Zweck
modules	
POST /api/gsdml-catalog/reload POST /api/gsdml-catalog/directory POST /api/gsdml-catalog/browse-folder	Den Katalogordner jetzt neu durchsuchen; den durchsuchten Ordner ändern; eine Ordnerauswahl öffnen (nur interaktive Desktop-Sitzungen) (§9.10).
POST /api/devices/gsdml	Ein neues Gerät direkt aus einem GSDML-Katalogeintrag (DAP + Slot-/Modulzuweisungen) erstellen und hinzufügen — ganz ohne .xdb (§9.10).

A.6 PROFINET Device — eigene Konfiguration

Endpunkt	Zweck
GET /api/device GET /api/slots	Die eigene Identität dieses Devices (Stationsname, Vendor-/Device-ID) und das aktuelle Slot-/Subslot-Layout.
GET /api/modules POST /api/modules	Das eigene Slot-/Subslot-Modullayout dieses Devices lesen oder festlegen — muss mit dem übereinstimmen, was dem IO-Controller (über GSDML) beschrieben und hier physisch simuliert wird (§10.3).
GET /api/io POST /api/io	Aktuelle Eingangsbytes des Devices, wie vom Controller gesehen, lesen oder schreiben (Simulation eines Sensors); Ausgangsbytes kommen vom Controller und sind hier schreibgeschützt. Die GET-Antwort trägt zusätzlich die aktuelle AR-Phase und ggf. den Stationsnamen des verbundenen Controllers — einen eigenen /api/ar-Endpunkt gibt es nicht.
GET /api/diagnostics POST /api/diagnostics/test-pattern POST/DELETE /api/diagnostics/force DELETE /api/diagnostics/forces POST /api/diagnostics/provider-status POST /api/diagnostics/reset-statistics	Den Zustand der Diagnose-Karte lesen; ein wiederholendes Testmuster auf einem Slot fahren; einen einzelnen Byte-Bereich oder den simulierten Wert eines Tags erzwingen (oder löschen); alle aktiven Forces auf einmal löschen; einen Modulfehler simulieren (IOPS = Bad) oder aufheben; die Rahmen-/Fehlerzähler zurücksetzen (§10).

Anhang B

Konfigurationsdateien

Jede Engine hält ihre Konfiguration unter ihrem eigenen Config\Ordner neben der Dienst-Executable als einfaches, menschenlesbares JSON. Dateien werden normalerweise über die Weboberfläche bearbeitet; manuelles Bearbeiten bei gestopptem Dienst funktioniert ebenfalls, aber ein laufender Dienst muss neu gestartet werden, um manuelle Änderungen zu übernehmen.

B.1 Config\enginesettings.json

Eine kleine Datei auf Host-Ebene, die genau eine Sache enthält: welche der fünf Engines aktiv ist. Wird geschrieben, wenn Sie über die Kopfleiste oder den Engine-Wähler die Engine wechseln (§4); der Dienst liest sie einmal beim Start und startet sich nach einem Wechsel innerhalb weniger Sekunden mit der neu gewählten Engine neu (§4). Sie enthält NICHT die NIC-Auswahl, den Stationsnamen, IP/Subnetz/Gateway oder sonstige engine-spezifische Einstellungen — diese liegen in der eigenen Einstellungsdatei jeder Engine, aufgeführt in §B.2 unten.

B.2 Ayars und Geräte-/Target-Liste je Engine

Jede Engine hält ihre eigenen Identitäts-/Netzwerkeinstellungen und, sofern sie mehr als eine entfernte Station verwaltet, ihre Geräte-/Slave-/Target-Liste unter einem für diese Engine spezifischen Dateinamen — nicht unter einem gemeinsamen generischen Namen. Bei Engines mit mehreren Stationen trägt die Liste Adresse und I/O-Form jeder einzelnen (PROFINET-Controller-Geräte werden normalerweise durch Import einer .xdb-Datei befüllt, §9.3, und tragen Stationsname, IP und vollständiges Slot-/Subslot-Layout; EtherCAT-Master-Slaves tragen ihre per ESI abgeglichenen PDO-Map, §6.2; EtherNet/IP-Scanner-Targets tragen IP, Assembly-Instanzen/-Größen und RPI, §7.2). PROFINET Device und EtherNet/IP Adapter stellen eine einzelne Station dar, daher gibt es keine separate Listendatei — ihre eine Einstellungsdatei deckt sowohl Identität als auch I/O-Form ab.

Engine	Einstellungsdatei	Liste (nur bei mehreren Stationen)
PROFINET Controller	Config\controllersettings.json	Config\devices.json
PROFINET Device	Config\deviceconfig.json	— (einzelne Station)
EtherNet/IP Scanner	Config\ethernetip-scanner-targets.json (enthält auch die Target-Liste)	dieselbe Datei
EtherNet/IP Adapter	Config\ethernetip-adapter-config.json	— (einzelne Station)
EtherCAT Master	Config\ethercat-master-config.json (enthält auch die Slave-Liste)	dieselbe Datei

Bearbeitet über die Kopfleiste und die eigenen Einstellungs-/Geräte-Karten jeder Engine; der Dienst muss neu gestartet werden, um eine manuell vorgenommene Änderung anzuwenden, während er gestoppt ist.

B.3 Tag-Datei je Engine

Etikettdefinitionen (§5.3) liegen ebenfalls unter einem für die aktive Engine spezifischen Dateinamen — ein Wörterbuch in der Form {"Geltungsbereich": [Tags], ...}, wobei "Geltungsbereich" bei Engines mit mehreren Stationen ein Geräte-/Slave-/Target-Name ist oder bei PROFINET Device und EtherNet/IP Adapter ein einzelner fester Schlüssel. Bearbeitet über die Tags-Karte der Weboberfläche oder die Tag-Methoden des .NET-Clients; das

Entfernen eines Geräts/Targets aus der Liste einer Multi-Stationen-Engine löscht auch dessen Tags hier automatisch.

Engine	Tag-Datei
PROFINET Controller	Config\tags.json
PROFINET Device	Config\tags.json
EtherNet/IP Scanner	Config\ethernetip-scanner-tags.json
EtherNet/IP Adapter	Config\ethernetip-adapter-tags.json
EtherCAT Master	Config\ethercat-master-tags.json

PROFINET Controller und PROFINET Device verwenden aus einem unabhängigen Grund zufällig denselben Dateinamen (Config\tags.json) — da immer nur eine Engine gleichzeitig läuft, existieren die beiden Dateien nie gleichzeitig.

B.4 Weitere Dateien

Config\lisans.json enthält die aktivierte Lizenz (entfernt durch uninstall.bat -RemoveConfig oder durch Entfernen der Lizenz über die Weboberfläche). Es ist bei jeder Engine dasselbe Dateiformat und dieselbe Speicherortkonvention, da eine Lizenz alle fünf abdeckt (§12.3). Beachten Sie, dass der Dateiname lisans.json lautet, nicht license.json. Der Ordner logs\ enthält rollierende tägliche Protokolldateien mit demselben Inhalt wie der Live-Feed der Dienstprotokoll-Karte. Engines mit einem Gerätekatalog halten diesen zusätzlich in einem eigenen Unterordner — Config\profinet-controller-gsdml\ (GSDML), Config\ethercat-master-esi\ (ESI), Config\ethernetip-scanner-eds\ (EDS) — gemeldet über den /api/paths-Endpunkt dieser Engine (§A).

Anhang C

Glossar

C.1 PROFINET

Begriff	Definition
AR	Application Relation — die logische Verbindung zwischen einem Controller und einem IO-Device.
IOCR	IO Communication Relation — der als Teil eines AR eingerichtete zyklische Datenkanal.
DCP	Discovery and Configuration Protocol — dient dem Auffinden eines Geräts anhand des Stationsnamens und ggf. der Zuweisung von Name/IP.
RPC	Das eigentliche Verbindungsmanagement-Protokoll von PROFINET über UDP/IP (Port 34964) — verwendet für Connect, Write/PrmEnd und Release.
DAP	Device Access Point — das DeviceAccessPointItem in der GSDML eines Geräts, das das Gerät selbst repräsentiert. Es ist immer Slot 0, fest (kann nicht entfernt oder getauscht werden), und seine in der GSDML deklarierten Steckmodule füllen die übrigen Slots des Geräts (§9.10).
IOPS / IOCS	IO Provider Status / IO Consumer Status — je ein Statusbyte, das den zyklischen Daten jedes Submoduls in jeder Richtung angehängt wird: IOPS gibt an, ob die gerade vom Provider gesendeten Daten gut oder schlecht sind, IOCS gibt an, ob der Consumer sie tatsächlich nutzt. Die Funktion "Modulfehler simulieren" der Device-Diagnosekarte erzwingt für ein Submodul IOPS = Bad (§10).
PrmEnd	Der RPC-Aufruf, der die Parametrierung beendet und das Gerät anweist, seine Konfiguration anzuwenden.
GSDML	General Station Description Markup Language — die vom Gerätehersteller veröffentlichte XML-Datei, die dessen Fähigkeiten, Module und Parameter beschreibt.
XDB	Geräte-Stellvertreterdatei — der von TIA Portal kompilierte Export der Projektkonfiguration einer Controller-PC-Station; der Controller liest sie direkt, anstelle der rohen GSDML eines Geräts (§9.5).
IO-Controller	Die Master-Station — dieses Produkt im Betrieb der PROFINET-Controller-Engine.
IO-Device	Eine entfernte Station mit modularer I/O — entweder echte Feldhardware oder dieses Produkt im Betrieb der PROFINET-Device-Engine.
Watchdog	Die Zeit, die ein Controller auf einen zyklischen Rahmen wartet, bevor er ein AR als verloren erklärt; auch Aktualisierungszeit-/Timeout-Multiplikator genannt.

C.2 EtherCAT

Begriff	Definition
ESI	EtherCAT Slave Information — die vom Slave-Hersteller veröffentlichte XML-Datei, die dessen PDOs, Parameter und unterstützte Modi beschreibt; wird automatisch mit der SII-Signatur jedes Slaves abgeglichen (§6.2).
SII	Slave Information Interface — ein kleines EEPROM, das jeder Slave trägt und das seine Hersteller-ID, seinen Produktcode und seine Revision über den Ring selbst meldet.

Begriff	Definition
PDO	Process Data Object — das zyklische Eingangs-/Ausgangs-Bit-/Byte-Layout, das ein Slave in jedem Zyklus austauscht; aus dem abgeglichenen ESI-Eintrag gelesen.
SDO	Service Data Object — ein azyklischer Mailbox-Lese-/Schreibzugriff auf das CoE-Objektverzeichnis eines Slaves, verwendet für Startparameter und Diagnose.
ESM	EtherCAT State Machine — die INIT→PREOP→SAFEOP→OP-Sequenz, die jeder Slave durchläuft, bevor zyklische Daten zu fließen beginnen.
SM	Sync Manager — der slave-seitige Puffermechanismus, der zyklische und Mailbox-Daten zwischen dem Rahmen des Masters und der Anwendung des Slaves übergibt.
FMMU	Fieldbus Memory Management Unit — bildet den lokalen physischen Speicher eines Slaves in das einzelne logische Prozessabbild des Masters ab.
CoE	CANopen over EtherCAT — die Objektverzeichnis-Konvention, die EtherCAT für SDO-Parameter und Diagnose von CANopen übernimmt.

C.3 EtherNet/IP

Begriff	Definition
CIP	Common Industrial Protocol — das objektorientierte Nachrichtenmodell, das EtherNet/IP über TCP (explizite Nachrichten) und UDP (zyklische I/O) trägt.
Forward Open	Der CIP-Dienst, der eine Class-1-(zyklische)-Verbindung zwischen einem Scanner und einem Target herstellt und dabei RPI und Assembly-Größen aushandelt.
Forward_Close	Der CIP-Dienst, der eine Class-1-Verbindung abbaut. Diese Engine sendet ihn immer dann, wenn die zyklische Austauschschleife eines Targets endet — auch dann, wenn ein Target aus der Liste des Scanners gelöscht wird, unmittelbar bevor es entfernt wird (§A.3).
Class 1 / Class 3	CIP-Verbindungstypen nach Zweck: Class 1 ist der zyklische, verbundene UDP-Verkehr, den ein Forward Open für IO-Assemblies einrichtet; Class 3 ist verbundene Request/Response-TCP-Nachrichtenübermittlung (explizite Nachrichten), die für einmalige Lese-/Schreibvorgänge wie das Parsen eines EDS oder das Lesen der Geräteidentität verwendet wird.
RPI	Requested Packet Interval — die zyklische Aktualisierungsrate, mit der ein Scanner ein Target betreiben lassen möchte.
Originator / Scanner	Die Station, die die Verbindung öffnet und den Austausch steuert — die EtherNet/IP-Scanner-Engine dieses Produkts, in einem realen System typischerweise eine SPS.
Target / Adapter	Die Station, die ein Forward Open annimmt und Assemblies bereitstellt — die EtherNet/IP-Adapter-Engine dieses Produkts, in einem realen System typischerweise ein Feldgerät.
Assembly	Eine benannte Bytegruppe, die CIP als eine Einheit austauscht — Input, Output oder Config, jeweils mit eigener Instanznummer und Größe.
ListIdentity	Der CIP/UDP-Broadcast-Discovery-Befehl (Port 44818) — das EtherNet/IP-Äquivalent zu PROFINETs DCP Identify. Die Discovery-Funktion des Scanners sendet ihn als Broadcast und liest von jedem antwortenden Gerät Vendor ID, Product Code, Device Type und Revision zurück, gleicht diese Identität anschließend mit dem EDS-Katalog ab (§A.3).
EDS	Electronic Data Sheet — das vom Hersteller eines Targets veröffentlichte Dateiformat, das dessen Assemblies und Parameter beschreibt (das EtherNet/IP-Äquivalent zu GSDML/ESI).

C.4 Gemeinsame Begriffe

Begriff	Definition
Engine	Eine der fünf mit diesem Produkt ausgelieferten Protokollimplementierungen — EtherCAT Master, EtherNet/IP Scanner, EtherNet/IP Adapter, PROFINET Controller, PROFINET Device. Es läuft jeweils nur eine.
Tag	Ein einem Byte-/Bit-Bereich innerhalb der rohen IO-Daten eines Geräts gegebener Name, sodass er über die Weboberfläche oder den .NET-Client per Namen statt per rohem Offset gelesen oder geschrieben werden kann (§5.3).
Cycle time	Das Intervall, in dem eine Engine ihren zyklischen Austausch antreibt — je Engine konfigurierbar, typischerweise einstellige Millisekunden.
Jitter	Wie weit die tatsächliche Zeitsteuerung eines Zyklus von der konfigurierten Zykluszeit/RPI/Aktualisierungszeit abweicht; durch die in Kapitel 13 beschriebene Echtzeit-Feinabstimmung im Sub-Millisekundenbereich gehalten.
Machine ID	Die auf der Lizenzkarte angezeigte und zur Aktivierung eines Schlüssels auf diesem PC verwendete Kennung. Sie leitet sich aus zwei von drei unabhängigen Hardware-/Betriebssystemquellen ab (SMBIOS-System-UUID, Windows-Installations-GUID, Datenträger-Volumeseriennummer); jede wird gesalzen und gehasht, sodass die Rohwerte den Rechner nie verlassen. Eine Lizenz bleibt gültig, solange mindestens zwei der drei weiterhin übereinstimmen, sodass routinemäßige Wartung (Festplattentausch, NIC-Wechsel, OS-Neuinstallation) sie nicht stillschweigend bricht (§12.3).
Trial period	Das 30-tägige Evaluierungsfenster, bevor ein Lizenzschlüssel erforderlich ist. Sein Startdatum wird redundant an drei Orten erfasst (Installationsordner, %ProgramData% und Registry), jeder Eintrag per HMAC an die Machine ID dieses PCs gesiegelt, sodass er nicht von Hand bearbeitet werden kann, um ihn zu verlängern; beim Lesen gewinnen der älteste erfasste Start und der am weitesten fortgeschrittene "zuletzt gesehen"-Zeitpunkt aller drei; ein Zurückspringen der Systemuhr wird als abgelaufene Testphase gewertet (§12.3).