DOKUMENTATION

PLCcom für Modbus
Anhang D

PLCcom
by Indi.An Germany

Technische Produktdokumentation

Daten, Register und Diagnose

PLCcom für Modbus · Java

Dokument-ID	IND-MODBUS-JAVA-DATA-DIAGNOSTICS-DE
Dokument-Version	1.0
SDK-Version	9.x
Sprache	Deutsch
Plattform	Java
Herausgeber	Indi.An GmbH

Indi.An GmbH · Flughafenallee 3 · D-28199 Bremen
support@indi-an.com · Tel +49 421 989703-30

PLCcom für Modbus Java — Anhang D: Daten, Register und Diagnose

Dokument-ID: IND-MODBUS-JAVA-DATA-DIAGNOSTICS-DE
Dokumentversion: 1.0
SDK-Version: 9.x
Sprache: Deutsch
Plattform: Java

Inhaltsverzeichnis

1. Über dieses Dokument
2. Modbus-Datenmodell
3. Datentypen
4. Registermodi
5. 16-, 32- und 64-Bit-Register
6. Byteorder
7. Qualität und Result-Objekte
8. Telegrammlogging
9. ReadCollection und Diagnose
10. Gerätebesonderheiten
11. Fragen und Support

1. Über dieses Dokument

Dieser Anhang beschreibt die Themen, die sowohl Master- als auch Slave-Anwendungen betreffen: Datentypen, Registerbreiten, Byteorder, Ergebnisqualität, Logging und typische Diagnoseabläufe.

Modbus ist einfach, aber die korrekte Interpretation der Registerdaten ist gerätespezifisch. Dieser Guide hilft dabei, die relevanten Stellschrauben bewusst zu verwenden.

2. Modbus-Datenmodell

Modbus unterscheidet vier logische Datenbereiche.

Bereich	Zugriff	Grundgröße
Coils	lesen/schreiben	1 Bit
Discrete Inputs	lesen	1 Bit
Input Registers	lesen	16 Bit
Holding Registers	lesen/schreiben	16 Bit

Register sind immer 16 Bit breit. Größere Werte entstehen durch Kombination mehrerer Register. Wie diese Register zu einem Wert zusammengesetzt werden, hängt vom Gerät ab.

2.1 Bereich, Funktion und Adresse

In vielen Gerätehandbüchern werden Modbus-Adressen in einer historischen Schreibweise angegeben, zum Beispiel `40001` für Holding Register oder `30001` für Input Register. Diese Schreibweise ist für Menschen hilfreich, aber nicht identisch mit der Adresse im Modbus-Telegramm.

PLCcom arbeitet mit Funktion und Startadresse. Die Startadresse ist dabei die Adresse, die im Telegramm verwendet wird. Häufig entspricht die Handbuchadresse `40001` deshalb der Startadresse `0` für Holding Register.

Handbuchschreibweise	Typischer Modbus-Bereich	Typische Funktion	Häufige Telegrammadresse
`00001`	Coil	Coil lesen/schreiben	`0`
`10001`	Discrete Input	Discrete Inputs lesen	`0`
`30001`	Input Register	Input Registers lesen	`0`
`40001`	Holding Register	Holding Registers lesen/schreiben	`0`

> **Hinweis:** Nicht jedes Handbuch verwendet dieselbe Adresslogik. Manche Hersteller dokumentieren bereits die 0-basierte Telegrammadresse, andere verwenden die klassische 1-basierte Modbus-Schreibweise. Wenn ein Gerät immer um ein Register "daneben" antwortet, ist eine 0-/1-basierte Adressverwechslung eine der ersten Ursachen, die geprüft werden sollte.

2.2 Register sind Rohdaten

Ein Modbus-Register enthält zunächst nur 16 Bits. Die Bedeutung entsteht erst durch die Gerätedokumentation und die Anwendung:

- Ist der Wert vorzeichenbehaftet oder vorzeichenlos?
- Ist der Wert ein Zähler, ein Statuswort, ein Messwert oder ein Teil eines größeren Datentyps?
- Muss ein Skalierungsfaktor angewendet werden?
- Gehören mehrere Register zu einem Wert?
- Welche Byteorder verwendet das Gerät?

Viele Geräte liefern Messwerte als Integer mit Faktor. Ein Registerwert `235` kann zum Beispiel `23,5 °C` bedeuten, wenn laut Handbuch ein Faktor `0,1` gilt. Solche fachlichen Skalierungen sind nicht Teil des Modbus-Protokolls und müssen in der Anwendung berücksichtigt werden.

3. Datentypen

PLCcom stellt Datentypen bereit, damit eine Anwendung nicht manuell Bytes und Register zusammensetzen muss.

Typische Datentypen:

Datentyp	Typische Bedeutung
`BOOL`	einzelner binärer Wert
`BYTE` / `SBYTE`	8-Bit-Wert
`SHORT` / `USHORT`	16-Bit-Wert
`INT` / `UINT`	32-Bit-Wert
`LONG` / `ULONG`	64-Bit-Wert
`FLOAT`	32-Bit-Gleitkommawert
`DOUBLE`	64-Bit-Gleitkommawert

Die tatsächlich verfügbaren Datentypen sind in der API-Referenz dokumentiert.

3.1 Datentyp, Registeranzahl und Geräteablage

Der Datentyp beschreibt, wie PLCcom die empfangenen Registerdaten in einen Java-Wert umwandelt. Daraus ergibt sich, wie viele Bits für den Wert benötigt werden. Das bedeutet aber noch nicht automatisch, welchen Registermodus das Gerät verwendet.

Datentypgruppe	Nutzbreite	Normaler Modbus-Fall mit 16-Bit-Registern	Möglicher erweiterter Registermodus
`SHORT` / `USHORT`	16 Bit	1 Register	keiner erforderlich
`INT` / `UINT` / `FLOAT`	32 Bit	2 Register	32-Bit-Registermodus, wenn das Gerät ihn ausdrücklich beschreibt
`LONG` / `ULONG` / `DOUBLE`	64 Bit	4 Register	64-Bit-Registermodus, wenn das Gerät ihn ausdrücklich beschreibt

Für die meisten Geräte gilt: Auch 32-Bit- und 64-Bit-Werte werden als Folge normaler 16-Bit-Register dokumentiert. In diesem Fall bleibt der 16-Bit-Registermodus korrekt. Byteorder und Datentyp bestimmen dann, wie PLCcom die Registerfolge interpretiert.

3.2 Vorzeichen und Wertebereich

Ein häufiger Fehler ist die Verwechslung von vorzeichenbehafteten und vorzeichenlosen Datentypen. Dieselben 16 Bits können als `SHORT` einen negativen Wert und als `USHORT` einen großen positiven Wert ergeben.

Beispiel:

Rohwert	Interpretation als `SHORT`	Interpretation als `USHORT`
`0xFFFF`	`-1`	`65535`
`0x8000`	`-32768`	`32768`

Wenn Werte unerwartet negativ werden oder ungewöhnlich große Zahlen entstehen, sollte zuerst geprüft werden, ob der Datentyp zur Gerätedokumentation passt.

3.3 Skalierung und Einheiten

Viele Geräte übertragen keine Gleitkommazahl, sondern einen Integer mit Skalierung. Das Handbuch enthält dann Angaben wie:

Registerwert	Faktor	Fachlicher Wert
`235`	`0,1`	`23,5`
`1234`	`0,01`	`12,34`
`5000`	`1`	`5000`

PLCcom liest den Rohwert korrekt aus dem Modbus-Register. Die fachliche Umrechnung in Engineering Units, zum Beispiel °C, Volt, Ampere oder Prozent, gehört zur Anwendung oder zu einer darüberliegenden Mapping-Schicht.

4. Registermodi

Registermodi beschreiben, welche Registerbreite ein Gerät für einen Wert erwartet. Sie sind nicht identisch mit dem Java-Datentyp, sondern beschreiben die Ablage des Wertes im Modbus-Registermodell.

Der normale Modbus-Standard arbeitet mit 16-Bit-Registern. Deshalb unterstützen fast alle herkömmlichen Modbus-Geräte den 16-Bit-Registermodus. 32-Bit- oder 64-Bit-Registermodi sind Geräte- oder Herstellererweiterungen und werden nur von einzelnen Geräten unterstützt.

> **Wichtig:** Verwenden Sie 32-Bit- oder 64-Bit-Registermodi nur dann, wenn die Dokumentation des Zielgeräts diesen Modus ausdrücklich beschreibt. Wenn das Gerät nur normale 16-Bit-Register unterstützt und die Anwendung dennoch einen größeren Registermodus verwendet, passen Request-Aufbau und Geräteerwartung nicht zusammen. Die Kommunikation liefert dann falsche Werte, Modbus-Exceptions oder schlägt vollständig fehl.

PLCcom unterstützt 16-Bit-, 32-Bit- und 64-Bit-Registermodi, damit auch Geräte mit erweiterten Registerlayouts angebunden werden können. Die Auswahl muss immer aus der Gerätedokumentation abgeleitet werden.

Registermodus	Typischer Einsatz	Geräteunterstützung
16-Bit	Standard-Modbus-Register, einzelne `SHORT` / `USHORT`-Werte und klassische Registerlisten	praktisch alle Modbus-Geräte
32-Bit	Geräte, die Werte als 32-Bit-Registereinheiten oder spezielle Doppelregister ablegen	nur wenn vom Gerät dokumentiert
64-Bit	Geräte, die 64-Bit-Werte als entsprechende Registereinheiten bereitstellen	nur wenn vom Gerät dokumentiert

Ein breiter Registermodus ist nicht automatisch besser. Er ist nur dann korrekt, wenn das Gerät ihn erwartet und der verwendete Datentyp dazu passt.

4.1 Entscheidungshilfe

Die folgende Entscheidungshilfe ist für die Inbetriebnahme oft praktischer als die reine Typbeschreibung:

Frage	Konsequenz
Beschreibt das Handbuch normale 16-Bit-Registeradressen?	16-Bit-Registermodus verwenden
Beschreibt das Handbuch einen 32-Bit-Registermodus oder 32-Bit-Registereinheiten?	32-Bit-Registermodus prüfen und nur für passende 32-Bit-Datentypen verwenden
Beschreibt das Handbuch einen 64-Bit-Registermodus oder 64-Bit-Registereinheiten?	64-Bit-Registermodus prüfen und nur für passende 64-Bit-Datentypen verwenden
Ist die Dokumentation unklar?	zunächst 16-Bit-Modus verwenden und mit bekanntem Testwert validieren
Kommt eine Modbus-Exception wegen ungültiger Adresse oder Länge zurück?	Registermodus, Startadresse und Anzahl prüfen

4.2 Was passiert bei falschem Registermodus?

Ein falscher Registermodus ist kein kleiner Darstellungsfehler. Er kann den kompletten Request verändern:

- Die adressierte Registermenge passt nicht zur Geräteerwartung.
- Das Gerät liefert eine Modbus-Exception, zum Beispiel wegen ungültiger Adresse.
- Die Antwortlänge passt nicht zur erwarteten Datentypkonvertierung.
- Die Anwendung liest zwar Bytes, interpretiert aber einen falschen Wert.
- Schreibzugriffe können an der falschen Stelle oder mit falscher Länge landen.

Deshalb sollte der Registermodus nicht “ausprobiert” werden, wenn Schreibzugriffe auf produktive Geräte erfolgen. Für Schreiboperationen zuerst die Gerätedokumentation prüfen und möglichst an einem ungefährlichen Testregister validieren.

5. 16-, 32- und 64-Bit-Register

5.1 16-Bit-Registermodus

Der 16-Bit-Registermodus ist der normale Modbus-Fall. Ein Register besteht aus 16 Bit beziehungsweise zwei Bytes. Die meisten Gerätehandbücher beschreiben ihre Registertabellen auf dieser Grundlage.

Typische 16-Bit-Werte sind:

- `SHORT`
- `USHORT`
- einzelne Status- oder Fehlerworte
- klassische Holding- und Input-Register

Auch größere Werte können in normalen Modbus-Registertabellen über mehrere 16-Bit-Register verteilt sein. In diesem Fall bleibt der Registermodus trotzdem 16 Bit; die Zusammensetzung des größeren Wertes erfolgt über Datentyp und Byteorder.

Beispiel für einen 32-Bit-Wert im normalen 16-Bit-Modbusmodell:

Adresse	Inhalt
`100`	High Word oder Low Word des Wertes
`101`	zweites Word des Wertes

Der Wert belegt zwei Register, aber das Gerät arbeitet weiterhin mit normalen 16-Bit-Registern.

5.2 32-Bit-Registermodus

Der 32-Bit-Registermodus ist für Geräte vorgesehen, die Werte nicht als einzelne 16-Bit-Register, sondern als 32-Bit-Registereinheiten beschreiben. Das ist nicht der allgemeine Modbus-Standard, sondern eine gerätespezifische Ablageform.

Typische Kandidaten sind:

- `INT`
- `UINT`
- `FLOAT`

Der 32-Bit-Registermodus ist im SDK an passende 32-Bit-Datentypen gebunden. Eine Kombination mit 16-Bit-Datentypen ist keine sinnvolle Sonderform, sondern eine unzulässige Kombination und wird von PLCcom unterbunden.

Wenn ein Gerät normale 16-Bit-Register erwartet, muss deshalb der 16-Bit-Registermodus verwendet werden. Auch ein 32-Bit-Wert kann in diesem Standardmodus korrekt gelesen werden, wenn er laut Gerätedokumentation aus zwei normalen 16-Bit-Registern besteht. In diesem Fall bestimmen Datentyp und Byteorder die Zusammensetzung des Wertes.

Typische Hinweise in Gerätedokumentationen können Formulierungen sein wie “32-bit register”, “double register”, “DWord register”, “register pair as one value” oder herstellerspezifische Begriffe. Entscheidend ist nicht der Begriff selbst, sondern ob das Gerät die Adressierung und Länge wirklich anders erwartet.

5.3 64-Bit-Registermodus

V9 erweitert PLCcom um 64-Bit-Registerunterstützung. Sie ist für Geräte gedacht, die Werte als 64-Bit-Registereinheiten bereitstellen oder erwarten.

Typische Kandidaten sind:

- `LONG`
- `ULONG`
- `DOUBLE`

Der 64-Bit-Registermodus ist im SDK an passende 64-Bit-Datentypen gebunden. Kombinationen mit 16-Bit- oder 32-Bit-Datentypen sind unzulässig und werden von PLCcom unterbunden.

Wenn ein Gerät einen 64-Bit-Wert als Folge normaler 16-Bit-Register beschreibt, bleibt der 16-Bit-Registermodus korrekt. Wenn ein Gerät dagegen ausdrücklich eine 64-Bit-Registereinheit beschreibt, kann der 64-Bit-Registermodus verwendet werden. Entscheidend ist also nicht allein die Breite des Java-Datentyps, sondern die vom Gerät dokumentierte Registerablage.

64-Bit-Werte sind besonders empfindlich gegenüber falscher Byteorder, weil vier 16-Bit-Register beteiligt sein können. Ein einziger vertauschter Word-Block reicht aus, damit ein `DOUBLE` vollständig unplausibel wird.

Empfohlene Vorgehensweise:

1. Gerätedokumentation prüfen.
2. Prüfen, ob das Gerät nur normale 16-Bit-Register oder zusätzlich 32-/64-Bit-Registermodi beschreibt.
3. Registeradresse und Funktion bestimmen.
4. Datentyp bestimmen.
5. Registermodus passend zur Gerätedokumentation wählen.
6. Byteorder anhand eines bekannten Testwertes validieren.

6. Byteorder

Byteorder beschreibt die Reihenfolge, in der Bytes und Register zu einem mehrbyteigen Wert zusammengesetzt werden. Das Thema wird relevant, sobald ein Wert mehr als ein Byte oder mehr als ein 16-Bit-Register verwendet.

Ein einzelnes Modbus-Register besteht aus zwei Bytes. Bei einem `USHORT` muss also bereits klar sein, welches Byte das höherwertige Byte ist. Bei 32-Bit- oder 64-Bit-Werten kommt zusätzlich die Reihenfolge der Register hinzu. Genau hier unterscheiden sich Gerätehersteller.

Beispiel: Ein 32-Bit-Wert besteht aus zwei 16-Bit-Registern:

Register	Bytes
Register 0	A B
Register 1	C D

Je nach Gerät kann derselbe logische Wert als `A B C D`, `C D A B`, `B A D C` oder in einer weiteren herstellerspezifischen Reihenfolge übertragen werden. PLCcom kann diese Reihenfolge über die Byteorder-Einstellung anpassen.

Typische Varianten:

- Big Endian
- Little Endian
- Word Swap
- Byte Swap
- kombinierte Word-/Byte-Swaps

Die richtige Byteorder muss aus der Gerätedokumentation oder durch einen Test mit bekanntem Wert bestimmt werden. Besonders hilfreich ist ein Register, dessen Wert eindeutig bekannt ist, zum Beispiel eine Firmware-Version, ein konstanter Testwert oder ein Messwert, der am Gerätedisplay abgelesen werden kann.

6.1 Beispiel mit 32-Bit-Wert

Angenommen, ein Gerät liefert den 32-Bit-Rohwert `0x12345678`. Dieser Wert besteht aus vier Bytes:

```
A = 0x12
B = 0x34
C = 0x56
D = 0x78
```

Je nach Byteorder können die übertragenen Register unterschiedlich interpretiert werden:

Reihenfolge	Bedeutung
`A B C D`	High Word zuerst, Bytes innerhalb der Words normal
`C D A B`	Word Swap
`B A D C`	Byte Swap innerhalb der Words
`D C B A`	vollständige Umkehrung

Wenn die Anwendung statt eines erwarteten Wertes völlig unplausible Zahlen liest, ist ein solcher Reihenfolgefehler sehr wahrscheinlich.

6.2 Beispiel mit 64-Bit-Wert

Ein 64-Bit-Wert besteht aus acht Bytes. Wenn er über normale 16-Bit-Register übertragen wird, sind vier Register beteiligt.

Beispielwert:

```
0x1122334455667788
```

Dieser Wert kann in Bytes dargestellt werden als:

```
A = 0x11
B = 0x22
C = 0x33
D = 0x44
E = 0x55
F = 0x66
G = 0x77
H = 0x88
```

Eine typische Big-Endian-Ablage über vier 16-Bit-Register sieht so aus:

Register	Bytes
Register 0	A B
Register 1	C D
Register 2	E F
Register 3	G H

Andere Geräte können die vier Words in anderer Reihenfolge ablegen oder zusätzlich die Bytes innerhalb der Words tauschen.

Reihenfolge	Bedeutung
`A B C D E F G H`	High Word zuerst, Bytes innerhalb der Words normal
`G H E F C D A B`	Word-Reihenfolge der vier 16-Bit-Register umgekehrt
`B A D C F E H G`	Byte Swap innerhalb jedes 16-Bit-Registers
`H G F E D C B A`	vollständige Umkehrung

Bei 64-Bit-Gleitkommawerten (`DOUBLE`) ist eine falsche Byteorder besonders auffällig. Das Ergebnis ist dann oft nicht nur leicht falsch, sondern vollständig unplausibel.

6.3 Byteorder bei Integern und Gleitkommazahlen

Bei Integern fällt eine falsche Byteorder oft durch sehr große oder negative Werte auf. Bei Gleitkommazahlen ist das Fehlerbild noch deutlicher: Ein falsch zusammengesetzter `FLOAT` oder `DOUBLE` ergibt häufig extrem kleine, extrem große oder `NaN`-ähnliche Werte.

Deshalb sollten Gleitkommawerte bei der Inbetriebnahme immer mit einem bekannten Referenzwert geprüft werden. Ein Wert vom Gerätedisplay, ein statischer Simulationswert oder ein Hersteller-Testregister eignet sich dafür deutlich besser als ein schwankender Prozesswert.

Empfohlene Vorgehensweise:

1. Gerätedokumentation nach Begriffen wie Byte Order, Word Order, Endianness, High Word, Low Word oder Swap prüfen.
2. Registeradresse, Datentyp und Registeranzahl bestimmen.
3. Registermodus festlegen.
4. Byteorder passend zur Dokumentation einstellen.
5. Einen bekannten Wert lesen und prüfen, ob das Ergebnis plausibel ist.

Wenn ein Wert um Größenordnungen falsch ist, negative Werte unerwartet auftreten oder Gleitkommazahlen völlig unplausibel sind, ist häufig nicht die Kommunikation selbst defekt, sondern Registermodus, Datentyp oder Byteorder passen nicht zur Geräteablage.

6.4 Typische Fehlerbilder

Beobachtung	Mögliche Ursache
Wert ist um Faktor 10, 100 oder 1000 falsch	Skalierungsfaktor aus Gerätedokumentation fehlt
Wert ist negativ, obwohl nur positive Werte erwartet werden	vorzeichenbehafteter statt vorzeichenloser Datentyp oder umgekehrt
Integerwert ist extrem groß	Byteorder oder Datentyp falsch
`FLOAT` / `DOUBLE` ist völlig unplausibel	Byteorder, Wordorder oder Registermodus falsch
Modbus-Exception bei ansonsten korrekter Verbindung	Adresse, Funktionscode, Registeranzahl oder Registermodus passt nicht
Lesen funktioniert, Schreiben nicht	Register ist read-only oder Schreibfunktion passt nicht zum Bereich

7. Qualität und Result-Objekte

PLCcom liefert Operationsergebnisse mit Qualitätsstatus zurück. Eine Anwendung sollte diesen Status immer prüfen, bevor sie Nutzdaten verwendet.

```
ReadResult result = master.read(request);

if (result.getQuality() == OperationResult.eQuality.GOOD)
{
    // Werte verwenden
}
else
{
    // Meldung und Diagnose auswerten
}
```

Fehler können aus verschiedenen Ebenen stammen:

Ebene	Beispiel
Transport	Verbindung nicht möglich, Timeout
Protokoll	Modbus Exception Code
Konvertierung	Datentyp passt nicht zur Registeranzahl
Sicherheit	TLS- oder Zertifikatsfehler
Lizenz	ungültige Lizenzdaten

8. Telegrammlogging

Telegrammlogging ist ein wichtiges Werkzeug bei Inbetriebnahme und Support. Es zeigt, welche Requests gesendet und welche Antworten empfangen wurden.

Für HMI- und QHMi-Logfunktionen ist wichtig, dass auch zusammengefasste Operationen wie ReadCollection nachvollziehbaren Telegrammverkehr erzeugen.

Empfehlungen:

- Log-Level in der Anwendung konfigurierbar machen.
- Telegrammlogging bei Fehleranalyse aktivieren.
- Logs im Dauerbetrieb begrenzen.
- sensible Daten und Lizenzinformationen nicht unnötig protokollieren.

9. ReadCollection und Diagnose

ReadCollection wird häufig für zyklische Abfragen genutzt. Dabei ist Diagnose besonders wichtig, weil ein einzelner Zyklus mehrere Requests enthalten kann.

Eine gute Anwendung beantwortet bei Fehlern folgende Fragen:

- Welcher Einzelrequest ist fehlgeschlagen?
- Welcher Slave war betroffen?
- Welche Funktion und Adresse wurden verwendet?
- Gab es einen Timeout oder eine Modbus-Exception?
- Welches Telegramm wurde gesendet?
- Welche Antwort kam zurück?

PLCcom stellt dafür Result- und Logginginformationen bereit.

10. Gerätebesonderheiten

Viele Modbus-Geräte verhalten sich leicht unterschiedlich. Typische Besonderheiten:

- Wartezeit nach TCP-Verbindungsaufbau erforderlich
- Registeradressen in Dokumentation sind 1-basiert, API erwartet 0-basierte Adresse
- ungewöhnliche Byteorder
- begrenzte maximale Registeranzahl pro Request
- optionale Funktionen nicht implementiert
- langsame Antwortzeiten bei bestimmten Registern

`DelayAfterConnect` kann verwendet werden, wenn ein Gerät nach TCP-Verbindungsaufbau eine kurze Pause benötigt.

```
master.getConnector().setDelayAfterConnect(250);
```

11. Checkliste zur Dateninterpretation

Wenn Werte falsch aussehen, sollte die Interpretation von unten nach oben geprüft werden:

1. Wird der richtige Modbus-Bereich verwendet: Coil, Discrete Input, Input Register oder Holding Register?
2. Ist die Adresse die vom Gerät erwartete Telegrammadresse?
3. Passt der Function Code zum Bereich?
4. Ist der Registermodus für das Gerät korrekt?
5. Ist der Datentyp laut Handbuch signiert, unsigned oder floating point?
6. Stimmt die Byteorder?
7. Muss der Wert skaliert werden?
8. Stellt das Gerät den Wert nur in einem bestimmten Betriebszustand bereit?

Mehrere Einstellungen gleichzeitig zu ändern, macht die Inbetriebnahme schwer nachvollziehbar. Besser ist es, jeweils genau eine Einstellung zu ändern, erneut zu lesen und das Telegrammlog aufzubewahren.

11.1 Unsigned Datentypen in Java

Java-Integer-Primitiven sind signiert. PLCcom stellt deshalb eigene unsigned Datentypen für Modbus-Werte bereit, die den signierten Java-Bereich überschreiten. Verwenden Sie die unsigned Klassen und Builder, wenn das Gerätehandbuch Werte wie `USHORT`, `UINT` oder `ULONG` beschreibt.

Das ist besonders wichtig für Zähler, Masken, Statuswörter und Kennungen. Ein Statuswort kann als signiertes `short` negativ aussehen, obwohl die Rohbits korrekt sind.

11.2 Rohwerte und Engineering Values trennen

Der rohe Modbus-Wert und der fachliche Engineering-Wert sollten getrennt betrachtet werden. PLCcom liest und konvertiert den Protokollwert. Die Anwendung ergänzt danach typischerweise Skalierung, Einheit und fachlichen Namen.

Beispiel:

Ebene	Beispiel
Rohregister	`235`
Datentyp	`USHORT`
Skalierung	Faktor `0.1`
Engineering-Wert	`23.5 degC`

Diese Trennung erleichtert Logs und Supportfälle, weil der Rohwert weiterhin mit dem Gerätehandbuch verglichen werden kann.

---

12. Fragen und Support

Für technische Fragen, Lizenzthemen und Support wenden Sie sich an Indi.An GmbH bzw. an die in Ihrem Lizenz- oder Wartungsvertrag genannten Ansprechpartner.