optonic
vision · automation

Feldbus Schnittstelle

mikado

Version 4.0

Ein Produkt von Optonic
www.mikado-robotics.de

Optonic GmbH | Zollhallenstr. 11 | 79106 Freiburg | Deutschland/Germany
Tel.: +49 761-79095-0 | E-Mail: info@optonic.com
© Copyright Optonic GmbH, 2015-2026

Inhaltsverzeichnis

1 Kurzbeschreibung der Schnittstelle **2**

2 Genereller Ablauf der Kommunikation **3**

3 Status der Kommunikation **4**
3.1 LifeBit . . . 4
3.2 Mikado-Status . . . 5

4 Synchronisieren von Feldbus-Daten und Mikado-Variablen **5**

5 Rotations-Konvention **6**

6 Fehler-Codes **7**

7 Kommandos **10**
7.1 Programm spezifische Kommandos . . . 10
7.2 Allgemeine Kommandos . . . 12

8 Beispielablauf für Select-Zellen **15**

9 Aktualisierungen der Feldbus-Schnittstelle **16**
9.1 Version 3.3.1 . . . 16
9.2 Version 3.3.0 . . . 16
9.3 Version 3.1.2 . . . 17
9.4 Version 3.1.0 . . . 17

10 Feldbus-Schnittstelle allgemein **18**
10.1 Byte-Reihenfolge . . . 18
10.2 Datentypen . . . 18
10.3 Addressbelegung . . . 19

11 Spezifische Konfiguration für PROFINET **20**
11.1 GSDML . . . 20
11.2 Standard-Feldbus-Schnittstelle . . . 21

12 Spezifische Konfiguration für EtherCAT **21**
12.1 Schnittstellenbeschreibung . . . 21
12.2 Standard-Feldbus-Schnittstelle . . . 21

13 Spezifische Konfiguration für EtherNet/IP **21**
13.1 Schnittstellenbeschreibung . . . 21
13.2 Standard-Feldbus-Schnittstelle . . . 22

14 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle **22**
14.1 Datentypen . . . 22
14.2 Konfiguration . . . 22

15 Changelog **24**

---

1 Kurzbeschreibung der Schnittstelle

Über die Feldbus-Schnittstelle können die Kernfunktionen von Mikado aufgerufen werden. Dabei wird unterschieden zwischen allgemeinen Funktionen und Funktionen zur Steuerung eines geladenen Mikado-Programms.

Allgemeine Funktionen

- Durchführen der Hand-Auge-Kalibrierung.

Funktionen zur Steuerung eines Mikado-Programms

- Starten und Stoppen des Mikado-Programms bzw. einer einzelnen Sequenz innerhalb des Mikado-Programms.
- Die Synchronisation von Mikado-Variablen mit der SPS.

2 Genereller Ablauf der Kommunikation

Die Kommunikation wird realisiert mittels Kommandos (siehe *Kommandos*) die von SPS an Mikado gesendet werden. Der Status der Kommunikation wird mittels Status-Bits beschrieben (siehe *Status der Kommunikation*). Die folgende Ablaufbeschreibung gilt für alle Kommandos. Bedingung für die Kommunikation ist die vollständige Betriebsbereitschaft des Mikado-Systems. Das wird durch `OutReadyBit = 1` signalisiert und ist in der Mikado-Oberfläche anhand des **Systemstatus** abzulesen.

Tab. 1: Genereller Ablauf der Kommunikation

Zeit	**SPS**	**Mikado**
$< t_1$	Warte auf `OutReadyBit = 1` Warte auf `OutBusyBit = 0` Parameter setzen	Warten auf Kommando
t_1	`InCmd` setzen Warte auf `OutBusyBit = 1`	

t_2		`OutErrorCode = 0`
		`OutBusyBit = 1` (Feldbus-Adapter verarbeitet Befehl)
		Nur beim Start des Programms / einer Sequenz:
		`OutRunningBit = 1`

t_3	`InCmd` zurücksetzen auf `0` Warte auf `OutBusyBit = 0`	
t_4		`OutErrorCode` schreiben `OutBusyBit = 0` (Feldbus-Adapter hat Befehl verarbeitet)
t_5	Lesen von `OutErrorCode` ***Nur nach Start des Programms / einer Sequenz:*** Warte auf `OutRunningBit = 0`	
t_6	***Nur nach Start des Programms / einer Sequenz:*** Lesen von `OutErrorCode`	***Nur wenn zuvor ein Programms / eine Sequenz gestartet wurde:*** `OutRunningBit = 0` (MIK-Programm endet)
ab hier	ggf. Lesen von Variablen	

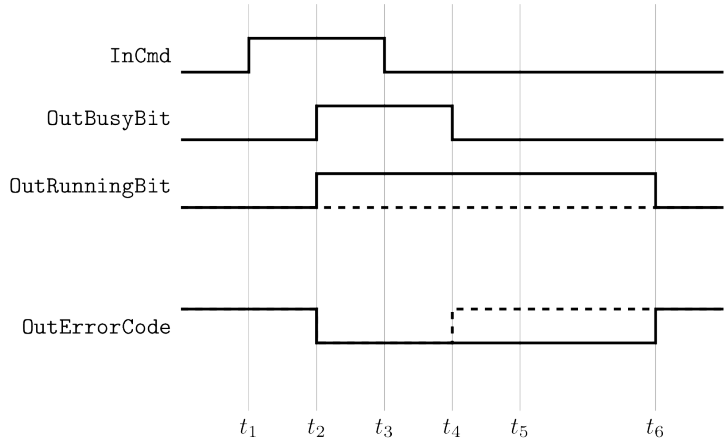


Abb. 1: Signalverlauf allgemein, alternative Verläufe gestrichelt

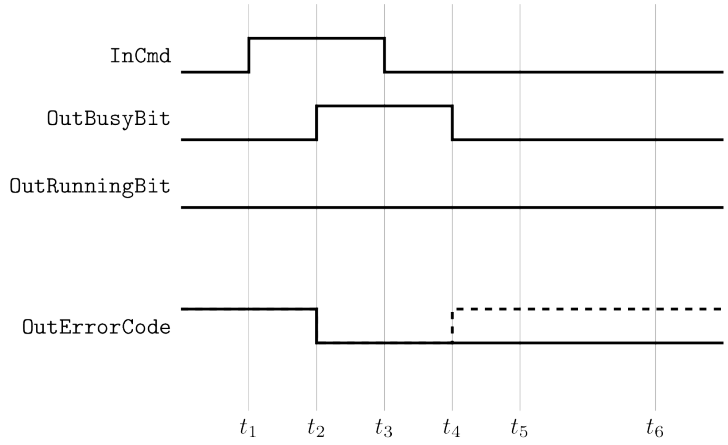


Abb. 2: Signalverlauf für Kommandos wie StartCalib, CaptureCalib, FinishCalib

- Wird ein Kommando gesetzt (`InCmd` $\neq$ `0`), antwortet Mikado mit `OutBusyBit = 1` und setzt zunächst den `OutErrorCode` zurück. Auf `OutBusyBit = 1` antwortet die SPS mit `InCmd = 0` und wartet dann auf `OutBusyBit = 0`.
- Die SPS wertet den `OutErrorCode` aus. Wenn dieser den Wert 0 hat, ist das Kommando erfolgreich ausgeführt worden.
- Nur bei Kommando `RunMikadoProg` und `StartSequence` wird `OutRunningBit = 1`. Das `OutRunningBit` signalisiert, dass Mikado das aktuelle Programm abarbeitet. Sobald `OutRunningBit = 0` wird, können der Fehlercode `OutErrorCode` und ggf. weitere Output-Variablen ausgelesen werden.

3 Status der Kommunikation

3.1 LifeBit

Um den Status der Kommunikation zwischen Mikado und SPS zu überprüfen wird ein LifeBit verwendet. Dabei liest die SPS den Wert von `OutLifeBit` und schreibt diesen unverändert auf `InLifeBit`. Der Feldbusadapter liest den Wert von `InLifeBit` und schreibt den negierten Wert auf `OutLifeBit`. Verändert sich der Wert von `OutLifeBit` nicht regelmäßig, gilt die Kommunikation als unterbrochen und ein entsprechender Fehler wird im Mikado-Status angezeigt.

3.2 Mikado-Status

Der Status von Mikado wird mittels drei verschiedener Bits an die SPS übertragen. Folgende Tabelle fasst die Bedeutung der einzelnen Bits zusammen.

Tab. 2: Bedeutung der einzelnen Status-Bits

Name	Erläuterung	Wert
`OutBusyBit`	Status des Feldbus-Adapters.	True = Kommando wird verarbeitet. False = Adapter ist bereit neues Kommando entgegenzunehmen
`OutReadyBit`	System-Status von Mikado.	True = Mikado betriebsbereit False = Mikado nicht bereit
`OutRunningBit`	Programm-Status von Mikado	True = Mikado-Programm läuft (entspricht in Mikado: Grün oder Gelb) False = Programm läuft nicht (entspricht in Mikado: Orange, Rot oder Grau)

Bemerkung

Neue Kommandos werden nur entgegengenommen, wenn `OutBusyBit` = False

4 Synchronisieren von Feldbus-Daten und Mikado-Variablen

Feldbus-Daten können auf Variablen eines Mikado-Programms abgebildet werden und umgekehrt. Hierdurch ist es möglich, Daten aus einem Mikado-Programmablauf heraus auf den Feldbus zu schreiben bzw. von dort zu lesen.

Die Zuordnung zwischen Feldbus-Daten und Mikado-Variablen kann im Mikado-Assistant unter „Schnittstellen“ konfiguriert werden. Für die verschiedenen, unterstützten Kommunikationsprotokolle (ADS, EtherCAT, EtherNet/IP, Profinet) stehen entsprechende Einrichtungsseiten zu Verfügung.

In der Tabelle der *Adressbelegung* kann im Feld „Variablenname in Mikado” für jedes Element der Feldbus-Schnittstelle der Name einer Mikado-Variablen angegeben werden. Dadurch wird dieses Element der Schnittstelle automatisch mit der entsprechenden Variable synchronisiert. Die Elemente der Standard-Adressbelegung sind mit den Präfixen `FI` (kurz für „fieldbus input“ = Feldbuseingang) bzw. `FO` (kurz für „fieldbus output“ = Feldbusausgang) benannt.

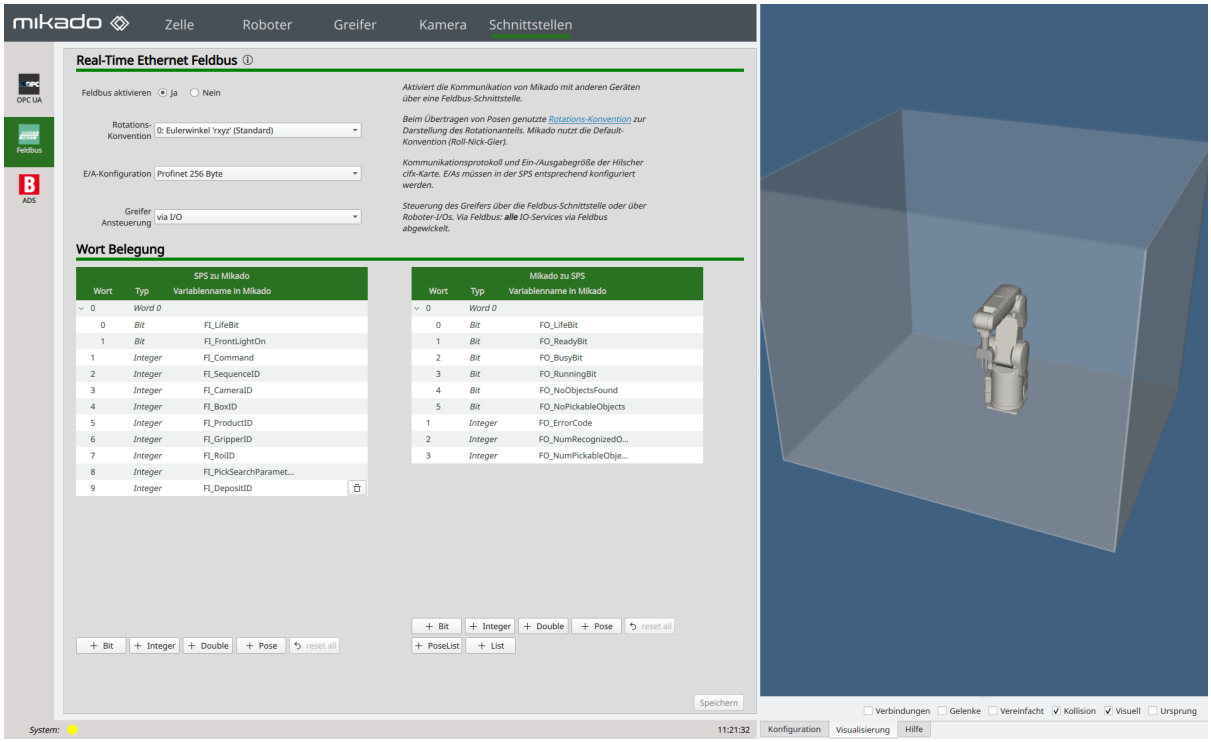


Abb. 3: Assistant zum Anpassen der Adressbelegung und synchronisierten Variablen

5 Rotations-Konvention

Die Rotations-Konvention legt fest, wie die Ausrichtung von Posen auf die Schnittstelle übertragen bzw. von der Schnittstelle gelesen wird. Die Ausrichtung wird dabei immer in Form von Eulerwinkeln angegeben, welche jedoch je nach Konvention auf unterschiedliche Weise interpretiert werden können (siehe unten).

Die Rotations-Konvention wird jeweils angegeben als eine Kombination aus vier Buchstaben: **r** (**r**elativ, intrinsisch, gedrehtes Koordinatensystem) oder **s** (**s**tatisch ,extrinsisch, feststehendes Koordinatensystem) gefolgt von der Reihenfolge der drei Rotationsachsen **x/y/z**. Es ergeben sich somit theoretisch 24 verschiedene Kombinationen, in denen die Ausrichtung von Posen in Eulerwinkeln dargestellt werden kann.

In der nachfolgenden Tabelle sind die Rotations-Konventionen für einige Roboterhersteller aufgeführt. Die Konvention muss passend zum verwendeten Roboter im Mikado Assistant unter dem Reiter „Schnittstellen“ eingestellt werden, damit die Ausrichtung von Posen korrekt interpretiert werden kann.

Mikado verwendet als Standard die Konvention *rxyz* (Roll-Nick-Gier / RPY). Bei Auswahl dieser Konvention werden die Mikado-Posen unverändert an die Schnittstelle weitergegeben. Bei Auswahl einer anderen Einstellung wird der Rotationsanteil einer Pose von der Schnittstelle in das entsprechende Format konvertiert.

Eine besondere Stellung nimmt die Rotations-Konvention *uvw* ein, welche von Universal Robots nativ verwendet wird. Anstatt diese Roboterseitig in Eulerwinkel umzurechnen, kann diese durch das Wählen der entsprechenden Konvention direkt verwendet werden. Die Parameter **u**, **v** und **w** stellen dabei einen Vektor dar, um welchen gedreht wird. Die Art der Rotationsbeschreibung entspricht der *Axis–Angle Representation*.

Tab. 3: Unterstützte Konventionen

ID	Konvention	Roboterhersteller	R1	R2	R3
0	*rxyz*	(Standard) Stäubli	*X* (Roll)	*Y‘* (Nick/Pitch)	*Z“* (Gier/Yaw)
1	*sxyz*	Fanuc Yaskawa Motoman Universal Robots Fruitcore Mitsubishi Techman	*X* (A)	*Y* (B)	*Z* (C)
2	*rzyx*	Kuka ABB Nachi	*Z* (A)	*Y‘* (B)	*X“* (C)
3	*rzyz*	Doosan Kawasaki Omron	*Z*	*Y‘*	*Z“*
4	*uvw*	Universal Robots	*U*	*V*	*W*

6 Fehler-Codes

Das Ausgangs-Wort `OutErrorCode` kodiert Fehlermeldungen. Die folgende Tabelle fasst alle möglichen Fehlercodes zusammen und erläutert jeweils ihre Bedeutung sowie empfohlene Maßnahmen zur Behebung.

Tab. 4: Mögliche Fehlermeldungen

Code	Fehlername	Beschreibung	Empfohlene Maßnahme
0	`NoError`	Ausgangszustand. Es liegt kein Fehler vor	-
>0	Fehler im Mikado-Programm	Projektspezifischer Fehler im Mikado-Programmablauf	Details siehe Mikado-Programm bzw. projektspezifische Dokumentation
-1	`ServiceCallFailed`	Aufruf eines Kommandos ist durch einen internen Fehler gescheitert.	Mikado Support kontaktieren
-4	`NoReplyValue`	Keinen Rückgabe-Wert erhalten (interner Fehler)	Mikado Support kontaktieren
-5	`TooManyReplyValues`	Anzahl der Rückgabewerte höher als erwartet (interner Fehler)	Mikado Support kontaktieren
-6	`FatalErrorMikadoProgram`	Mikado-Programm hat einen fatalen Fehler. Die Ausführung des Programms kann von dieser Stelle aus nicht fortgesetzt werden.	Mikado-Programm zurücksetzen und Programm / Sequenz erneut starten
-8	`UnknownCommand`	Unbekanntes Kommando aufgerufen	`InCmd` überprüfen. Gültige Kommandos siehe *Kommandos*.
-9	`LifebitClientMissing`	Lifebit des Clients wurde nicht erkannt.	Feldbus-Verbindung zwischen Mikado und dem Kommunikationspartner und Konfiguration des `LifeBit` überprüfen
-11	`TriggeringFailed`	Aufruf einer Mikado-Sequenz fehlgeschlagen. Dies kann beispielsweise auftreten, wenn Mikado nicht betriebsbereit ist oder bereits ein Programm ausgeführt wird.	Sicherstellen, dass Mikado betriebsbereit ist (`ReadyBit = true`) und dass nicht bereits ein Programm ausgeführt wird (`RunningBit = false`)
-12	`ReadingVariableFailed`	Interner Fehler beim Lesen von Daten der Feldbus-Schnittstelle	Mikado Support kontaktieren
-13	`WritingVariableFailed`	Interner Fehler beim Schreiben von Daten zur Feldbus-Schnittstelle	Mikado Support kontaktieren

Tab. 5: Mögliche Fehlermeldungen, Fortsetzung

Code	Fehlername	Beschreibung	Empfohlene Maßnahme
-15	`CancelCalibrationFailed`	Abbrechen einer bereits gestarteten Kalibrierung fehlgeschlagen	Sicherstellen, dass Kalibrierung zuvor gestartet wurde (*StartCalib*) und kein Programm ausgeführt wird (`RunningBit = false`)
-16	`CaptureForCalibrationFailed`	Kommando *CaptureCalib* ist fehlgeschlagen. Aufnahme für laufende Kalibrierung nicht erfolgreich	Sicherstellen, dass Kalibrierung zuvor gestartet wurde (*StartCalib*) und kein Programm ausgeführt wird (`RunningBit = false`)
-17	`FinishCalibrationFailed`	Kommando *FinishCalib* ist fehlgeschlagen. Kalibrierung berechnen und speichern war nicht erfolgreich	Sicherstellen, dass Kalibrierung zuvor gestartet wurde (*StartCalib*) und kein Programm ausgeführt wird (`RunningBit = false`)
-18	`StartCalibrationFailed`	Kommando *StartCalib* ist fehlgeschlagen. Kalibrierung konnte nicht gestartet werden	Sicherstellen, dass nicht bereits eine Kalibrierung läuft und kein Programm ausgeführt wird (`RunningBit = false`)
-19	`SequenceDoesNotExist`	Kommando *StartSequence* ist fehlgeschlagen. Mikado Sequenz mit der übergebenen ID existiert nicht	`SequenceID` überprüfen und Kommando *StartSequence* mit einer gültigen ID aufrufen
-20	`MissingConfigurationItem`	Ungültige Feldbus-Konfiguration. Ein erforderlicher Eintrag ist nicht vorhanden.	Mikado Support kontaktieren
-60	`RetryErrorMikadoProgram`	Mikado-Programm hat einen Fehler aber die Ausführung kann wiederholt werden. Dies kann auftreten, wenn die Verbindung zum Roboter oder zur Kamera kurzzeitig unterbrochen wird, während Mikado gerade eine Aktion ausführen möchte.	Kommando *RunMikadoProg* ausführen. In vielen Fällen ist die Ursache des Fehlers bis zum Wiederholversuch bereits automatisch behoben worden und der Ablauf kann erfolgreich fortgesetzt werden. Falls nicht, wird erneut ein Fehler ausgelöst.
-61	`RetryActionErrorMikadoProgram`	Roboterbewegung fehlgeschlagen. Ein Fahrbefehl von Mikado konnte nicht erfolgreich vom Roboter ausgeführt werden. Dies kann auftreten, wenn die Verbindung zum Roboter verloren geht oder dieser unerwartet gestoppt wird, beispielsweise durch einen Fehler oder einen Sicherheits-Stopp (Not-Aus)	Falls vorhanden, zunächst die Ursache des unerwarteten Stopps beheben und dann Kommando *RunMikadoProg* ausführen. Der Ablauf wird fortgesetzt. Wenn die Bewegung weiterhin nicht erfolgreich ausgeführt werden kann, wird erneut ein Fehler ausgelöst.
-100	`UnexpectedError`	Unerwarteter Fehler aufgetreten	Mikado Support kontaktieren

7 Kommandos

Alle Kommandos haben mindestens `OutErrorCode` als Rückgabewert. Einige Kommandos haben zusätzliche Eingabe- und Rückgabewerte. In der Detailbeschreibung der Kommandos sind die zusätzlichen Eingabe- und Rückgabewerte aufgeführt, sofern existent.

Bemerkung

Die Eingabewerte müssen **vor** dem Setzen des Kommandos am Feldbus anliegen damit sie berücksichtigt werden können. Die Rückgabewerte liegen am Feldbus an, sobald `OutBusyBit == 0` ist und die Konfigurationsdatei die entsprechenden Einträge enthält (siehe *Synchronisieren von Feldbus-Daten und Mikado-Variablen*).

Folgende Tabelle listet alle verfügbaren Kommandos auf:

Tab. 6: Verfügbare Kommandos

`InCmd`	Kommandoname	Beschreibung
0	`Idle`	Wartezustand
Programm spezifische Kommandos		
1	*RunMikadoProg*	Mikado-Programm ab dem aktuellen Task starten/fortsetzen
2	*StartSequence*	Eine bestimmte Sequenz des Mikado-Programms starten. Hierfür muss SequenceID zuvor gesetzt sein
3	*PauseMikadoProg*	Ablauf nach Ausführung des aktuellen Tasks anhalten
4	*StopMikadoProg*	Mikado-Programm stoppen
5	*ResetMikadoProg*	Mikado-Programm stoppen und alle Variablen und Zustände im Mikado zurücksetzen
Allgemeine Kommandos		
8	*StartCalib*	Hand-Auge-Kalibrierung starten
9	*CaptureCalib*	Löst eine Bildaufnahme und Messung des Kalibriermusters aus
10	*FinishCalib*	Hand-Auge-Kalibrierung berechnen und speichern
11	*readPoseAndPublishTool0*	Setzt die Pose des Roboterflanschs in Mikado
14	*CancelCalib*	Hand-Auge-Kalibrierung abbrechen
20	*Capture*	Löst eine Bildaufnahme aus
21	*StartPickSearch*	Startet die Objekt- und Griffsuche
22	*GetBestPick*	Liest den besten verfügbaren Griff aus

7.1 Programm spezifische Kommandos

Programm spezifische Kommandos beziehen sich auf das aktuelle Mikado-Programm und können nur ausgeführt werden, wenn ein Programm geladen ist. Diese Kommandos haben keine zusätzlichen Rückgabewerte. Die gewünschten Ergebnisse hängen von der Implementierung im Mikado-Programm ab und werden individuell über die synchronisierten Variablen kommuniziert (siehe hierzu *Synchronisieren von Feldbus-Daten und Mikado-Variablen*).

Andersherum kann das Verhalten des Mikado-Programms über den Feldbus beeinflusst werden, indem synchronisierte Variablen als Inputs für Tasks im Mikado-Programm verwendet werden. Dabei ist zu beachten, dass das Auslesen einer im Mikado-Programmablauf als Task-Input verwendeten synchronisierten Variable zum Zeitpunkt der Task-Ausführung geschieht. Deshalb sollten synchronisierte Variablen auf SPS-Seite nur geändert werden, wenn kein Mikado-Programm läuft.

Bemerkung

Um Fehlverhalten im Programmablauf zu vermeiden sollten synchronisierte Variablen auf SPS-Seite nur geändert werden, wenn kein Mikado-Programm läuft (`OutRunningBit = 0`).

RunMikadoProg

Mit diesem Kommando wird das aktuelle Mikado-Programm gestartet oder fortgesetzt. Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = True` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, können die synchronisierten Variablen ausgewertet werden.

Eingabewerte	Beschreibung
keine	

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode

StartSequence

Mit diesem Kommando wird die Mikado-Sequenz mit der in Wort `SequenceID` hinterlegten ID ausgeführt. Die ID einer Sequenz lässt sich ab Programm-Version 3 in dem Textfeld rechts neben dem Sequenz-Namen einstellen. In früheren Programm-Versionen muss der Sequenz-Name gleich der gesendeten ID sein, beispielsweise die Ganzzahl ‚42'.

Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = 1` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, können die synchronisierten Variablen ausgewertet werden.

Eingabewerte	Beschreibung
`InSequenceID`	ID der zu startenden Sequenz

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode

PauseMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms pausiert.

Eingabewerte	Beschreibung
keine	

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode

StopMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt.

Eingabewerte	Beschreibung
keine	

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode

ResetMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt und der Zustand des Programms und aller Variablen werden zurückgesetzt.

Eingabewerte	Beschreibung
keine	

Rückgabewerte	Beschreibung
OutErrorCode	Fehlercode

7.2 Allgemeine Kommandos

Allgemeine Kommandos können unabhängig vom Mikado-Programm ausgeführt werden. Die Rückgabewerte liegen auf der in der Konfigurationsdatei definierten Adresse an, sobald `OutBusyBit == 0`.

StartCalib

Mit diesem Kommando wird die Hand-Augen-Kalibrierung gestartet.

Eingabewerte	Beschreibung
InCameraID	ID der zu verwendenden Kamera
InFrontLightOn	Definiert, ob das Frontlicht während der Kalibrierung eingeschaltet ist

Rückgabewerte	Beschreibung
OutErrorCode	Fehlercode

CaptureCalib

Mit diesem Kommando werden Posen für die Hand-Augen-Kalibrierung gesammelt. Der Kalibrierungsprozess muss vorher durch das Kommando `StartCalib` gestartet worden sein. Für eine Kalibrierung müssen mindestens 6 gültige Posen gesammelt werden. Idealerweise sollten jedoch etwa 10 bis 20 Posen gesammelt werden, um ein möglichst gutes Ergebnis zu erzielen. Dieses Kommando muss dementsprechend häufig ausgeführt werden.

Eingabewerte	Beschreibung
InPose	Aktuelle Tool0-Pose (Roboterflansch)

Rückgabewerte	Beschreibung
OutErrorCode	Fehlercode

Beispielablauf:

- Roboter fährt eine Pose an
- SPS schreibt die aktuelle Flanschpose (tool0) auf `InPose`
- Das Kommando `CaptureCalib` wird ausgelöst
- Mikado liest die aktuelle Roboterpose von `InPose` und führt eine Bildaufnahme durch

FinishCalib

Mit diesem Kommando wird die Hand-Augen-Kalibrierung berechnet. Dies ist der letzte Schritt im Prozess der Hand-Augen-Kalibrierung. Dies erfolgt nach einer ausreichenden Anzahl von gesammelten Patternposen mit Hilfe von `CaptureCalib`.

Eingabewerte	Beschreibung
keine	

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode
`OutCalibMaxDist`	Maximaler Posen Fehler Distanz
`OutCalibMeanDist`	Mittlerer Posen Fehler Distanz
`OutCalibMeanX`	Mittlerer Posen Fehler x
`OutCalibMeanY`	Mittlerer Posen Fehler y
`OutCalibMeanZ`	Mittlerer Posen Fehler z
`OutCalibMeanAngle`	Mittlerer Posen Fehler Winkel

CancelCalib

Mit diesem Kommando wird die Hand-Augen-Kalibrierung abgebrochen.

Eingabewerte	Beschreibung
keine	

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode

readPoseAndPublishTool0

Bemerkung

Dieses Kommando findet in der Regel nur bei Select Zellen Anwendung.

Mit diesem Kommando wird eine von der SPS übermittelte Pose eingelesen und in Mikado als aktuelle Pose des Roboterflanschs (tool0) gesetzt. Für den Mikado-Programmablauf muss die Position des Roboters bei einer Bildaufnahme bekannt sein, wobei Mikado Select selbst keine Informationen über den Roboterzustand hat. Bei der Hand-Auge-Kalibrierung ist die Benutzung dieses Kommandos nicht notwendig, da das Kommando `CaptureCalib` im Hintergrund bereits dieselben Funktionalitäten ausführt.

Eingabewerte	Beschreibung
`InPose`	Aktuelle Tool0-Pose (Roboterflansch)

Rückgabewerte	Beschreibung
`OutErrorCode`	Fehlercode

Capture

Bemerkung

Dieses Kommando findet in der Regel nur bei Select Zellen Anwendung.

Mit diesem Kommando wird eine Aufnahme für die ausgewählte Kamera ausgelöst.

Eingabewerte	Beschreibung
InCameraID	ID der zu verwendenden Kamera

Rückgabewerte	Beschreibung
OutErrorCode	Fehlercode

StartPickSearch

Bemerkung

Dieses Kommando findet in der Regel nur bei Select Zellen Anwendung.

Mit diesem Kommando wird eine Griffsuche gestartet, d.h. die Suche nach Objekten und anschließend Griffen. Hierfür wird jeweils die letzte zuvor aufgenommene Aufnahme der ausgewählten Kamera verwendet. Es muss also gegebenenfalls zuvor eine Aufnahme mit dem Kommando *Capture* ausgelöst werden.

Durch den Aufruf dieses Kommandos wird der Suchlauf gestartet. Die Suche selbst läuft allerdings asynchron im Hintergrund. Das Kommando blockiert nicht, bis Ergebnisse vorliegen!

Eingabewerte	Beschreibung
InCameraID	ID der zu verwendenden Kamera
InBoxID	ID des zu verwendenden Ladungsträgers
InProductID	ID des zu verwendenden Produkts
InGripperID	ID des zu verwendenden Greifers
InRoiID	ID der zu verwendenden Suchregion (Region of Interest)
InPickSearchParametersID	ID der zu verwendenden Suchparameter

Rückgabewerte	Beschreibung
OutErrorCode	Fehlercode

GetBestPick

Bemerkung

Dieses Kommando findet in der Regel nur bei Select Zellen Anwendung.

Mit diesem Kommando wird der zum Zeitpunkt des Aufrufs beste verfügbare Griff ausgelesen. Zur Identifikation des richtigen Suchlaufs müssen die Parameter InBoxID , InProductID und InGripperID beim Aufruf des Kommandos angegeben werden. Die Bezeichnung „bester Griff“ bezieht sich hierbei auf die eingestellten Kriterien für die Sortierung von Griffen.

Die Griffsuche profitiert davon, wenn zwischen dem Start (*StartPickSearch*) und dem Auslesen von Ergebnissen (*GetBestPick*) ein möglichst großer Zeitraum liegt. Nur so bleibt genügend Zeit, um alle möglichen Griffvariationen zu berechnen und sie untereinander zu priorisieren, sodass der bestmögliche Griff ermittelt werden kann.

Wenn die Suche zum Zeitpunkt des Aufrufs dieses Kommandos noch nicht beendet ist, wird solange blockiert, bis entweder mindestens ein gültiger Griff gefunden wurde oder alle möglichen Griffvariationen berechnet wurden, jedoch kein einziger gültiger Griff vorhanden ist.

Um ein schnelles Ergebnis zu liefern, wird die Suche in diesem Fall allerdings gegebenenfalls vorzeitig beendet und der erstmögliche Griff zurückgeliefert. Dabei handelt es sich dann um irgendeinen gültigen Griff, aber nicht unbedingt um den besten theoretisch möglichen Griff.

Für optimale Ergebnisse sollte *StartPickSearch* daher so früh wie möglich im Zyklus aufgerufen werden, *GetBestPick* hingegen so spät wie möglich (kurz vor dem Griff). Dadurch wird der Griffsuche der größtmögliche Berechnungszeitraum ermöglicht, um einen idealen Griff zu finden.

Eingabewerte	**Beschreibung**
`InBoxID`	ID des zu verwendenden Ladungsträgers
`InProductID`	ID des zu verwendenden Produkts
`InGripperID`	ID des zu verwendenden Greifers

Rückgabewerte	**Beschreibung**
`OutErrorCode`	Fehlercode
`OutNoObjectsFoundBit`	Mit diesem Bit wird signalisiert, ob Produktobjekte gefunden wurden. `true` wenn keine Objekte vorhanden sind, ansonsten `false`.
`OutNoPickableObjectsBit`	Mit diesem Bit wird signalisiert, ob greifbare Objekte gefunden wurden. `true` wenn keine greifbaren Objekte vorhanden sind, ansonsten `false`.
`OutNumRecognizedObjects`	Anzahl der erkannten Produktobjekte
`OutNumPickableObjects`	Anzahl der zum Zeitpunkt des Aufrufs greifbaren Produktobjekte
`OutPickGripperId`	ID des Greifers, mit dem der ausgewählte Griff auszuführen ist
`OutPickCategory`	Kategorie des Griffs
`OutPickSubcategory`	Subkategorie des Griffs
`OutPickStartPose`	Startpose des Griffs. Die Pose kann je nach Einstellung in TCP- oder in Tool0-Koordinaten (Roboterflansch) ausgegeben werden.
`OutPickApproachPose`	Anfahrtspose des Griffs. Die Pose kann je nach Einstellung in TCP- oder in Tool0-Koordinaten (Roboterflansch) ausgegeben werden.
`OutPickGraspPose`	Greifpose. Die Pose kann je nach Einstellung in TCP- oder in Tool0-Koordinaten (Roboterflansch) ausgegeben werden.
`OutPickRetractPose`	Entnahmepose des Griffs. Die Pose kann je nach Einstellung in TCP- oder in Tool0-Koordinaten (Roboterflansch) ausgegeben werden.
`OutPickFinalPose`	Endpose des Griffs. Die Pose kann je nach Einstellung in TCP- oder in Tool0-Koordinaten (Roboterflansch) ausgegeben werden.

8 Beispielablauf für Select-Zellen

Anhand des folgenden Beispiels soll verdeutlicht werden, wie eine typischer Ablauf in Mikado Select abfolgt.

Tab. 7: Beispiel-Ablauf für eine Select-Zelle

#	SPS	Mikado
Aufnahme		
1	Setze `InCameraID` Setze `InCmd = 20` (`Capture`)	
2		Eine Aufnahme wird ausgelöst Ende der Sequenz: `OutBusyBit = 0`
Objekt- und Griffsuche		
3	Setze `InCameraID`, `InBoxID`, `InProductID`, `InGripperID`, `InRoiID`, `InPickSearchParametersID` Setze `InCmd = 21` (`StartPickSearch`)	
4		Objekt- und Griffsuche wird gestartet Ende der Sequenz: `OutBusyBit = 0`
Ergebnisse auslesen		
5	Setze `InBoxID`, `InProductID`, `InGripperID` Setze `InCmd = 22` (`GetBestPick`)	
6		Besten verfügbaren Griff auslesen Setze Rückgabewerte, siehe *GetBestPick* Ende der Sequenz: `OutBusyBit = 0`

9 Aktualisierungen der Feldbus-Schnittstelle

9.1 Version 3.3.1

Es wurden die Kommandos 20 (*Capture*), 21 (*StartPickSearch*) und 22 (*GetBestPick*) hinzugefügt. Mithilfe dieser Kommandos können Griffsuchen direkt über die Feldbus-Schnittstelle gestartet und deren Ergebnisse ausgelesen werden. Somit können beispielsweise Mikado-Select-Zellen bedient werden, ohne dass zwingend ein Mikado-Programmablauf notwendig ist.

Im Zuge dessen wurde auch die Standard-Adressbelegung sowohl für Mikado-Control als auch für Mikado-Select überarbeitet.

9.2 Version 3.3.0

Es wird nun eine PROFINET Konfiguration ausgeliefert, die offiziell von Siemens zertifiziert wurde. Diese erfordert die Verwendung einer neuen `Optonic` GSDML-Datei. Für die Kommunikation mit Siemens-SPS steht ein minimal angepasster TIA-Baustein zur Verfügung, der diese neue GSDML-Datei einbindet.

Bei existierenden Zellen erfolgt keine automatische Aktualisierung. Die Änderungen sind nicht rückwärtskompatibel und eine Anpassung ist sowohl Mikado- als auch SPS-seitig erforderlich. Bei Bedarf kann im Mikado Assistant zwischen der älteren und der nun aktuellen Konfiguration gewechselt werden (siehe Abbildung unten; ggf. ist die Verwendung des „erweiterten Modus" notwendig, um diese Option einzublenden). Für neue Zellen ist ab Version 3.3.0 des Mikado Assistant automatisch die neuere `Optonic` Konfiguration vorausgewählt. Es wird dringend empfohlen, diese Auswahl beizubehalten. Die veraltete `Hilscher` Konfiguration ist nur aus Gründen der Rückwärtskompatibilität weiterhin auswählbar.

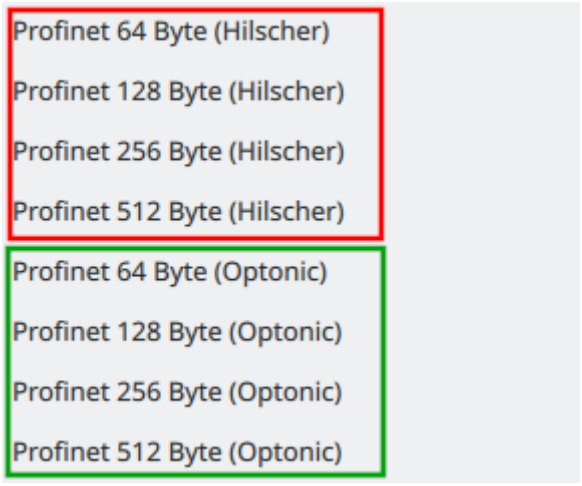


Abb. 4: E/A-Konfiguration für PROFINET: veraltete (rot) und neue (grün) Optionen

9.3 Version 3.1.2

Ab *mikado-profinet* Version 3.1.2 unterstützt das Paket neben Profinet auch die Feldbus-Protokolle EtherCAT und EtherNet/IP.

Das gewünschte Protokoll kann im Mikado Assistant auf der Seite `Feldbus-Schnittstelle` konfiguriert werden.

9.4 Version 3.1.0

(mikado-profinet v3.1.0, mikado_io_beckhoff v3.1.0)

Bemerkung

SPS-seitige Wertänderungen während der Programm-Laufzeit (`RunningBit = 1`) können ab dieser Version zu unerwartetem Verhalten führen, da sie sofort für das Programm sichtbar werden.

Die neue Feldbus-Adapter Version bringt die wesentliche Neuerung, dass synchronisierte Variablen ständig aktualisiert werden. Bei vorherigen Versionen mussten Variablen mit dem Kommando `syncAll` aktualisiert werden. Aus dieser Änderung resultieren folgende Konsequenzen:

- Die Nutzer-definierte Live-Eigenschaft von Variablen entfällt, weil nun alle Variablen „live“ sind.
- Das Kommando `syncAll` ist überflüssig und deshalb auch nicht mehr in dieser Dokumentation aufgeführt. Das Kommando kann weiterhin gesendet werden. Der zurückgegebene `OutErrorCode` ist in diesem Fall `NoError`.
- Erfolgen SPS-seitige Wertänderungen zur Programm-Laufzeit kann es zu unerwartetem Verhalten kommen.

Nachstehende Tabelle zeigt beispielhaft wie SPS-seitige Änderungen zu einem ungünstigen Zeitpunkt zu einem unerwartetem Verhalten führen können. Angenommen in Mikado ist ein Programm geladen, das unter Anderem den Task `Finde Szenenobjekt` aufruft. Als Task-Input wird die synchronisierte Variable `InBoxID` verwendet.

Tab. 8: Beispiel für SPS-seitige Änderungen zu einem ungünstigen Zeitpunkt

Zeit	SPS	Mikado	Kommentar
t_1	Setze `InBoxID = 42` Setze `InCmd = 1` (Starte Programm)		SPS setzt die Variable `InBoxID = 42` und sendet Start-Kommando

t_2		`OutBusyBit = 1`
		`OutRunningBit = 1`
		`OutBusyBit = 0`
			Adapter führt Kommando aus
			Mikado startet Programm
			Kommando-Ausführung beendet

t_3	`InBoxID = 0` (**falsch**) `InCmd = 0`		**Dies ist ein ungünstiger Zeitpunkt.** Im Programmablauf verwendete Variablen sollten nicht zur Laufzeit geändert werden.
t_4		Programm ruft auf `FindSceneObject (InBoxID)`	Mikado ist mittlerweile beim Aufruf des Tasks angekommen. Es wird die momentan anliegenden `InBoxID = 0` verwendet.
t_5		`OutRunningBit = 0`	Programm-Ende erreicht
t_6	`InBoxID = 0` (**richtig**)		**Der richtige Zeitpunkt um Variablen zurückzusetzen.**

10 Feldbus-Schnittstelle allgemein

Die Feldbus-Schnittstelle von Mikado kommuniziert über eine Hilscher cifX-PCI-Karte und verhält sich unabhängig vom verwendeten Protokoll weitgehend identisch. Es stehen die folgenden Protokolle zur Verfügung:

- PROFINET
- EtherCAT
- EtherNet/IP

10.1 Byte-Reihenfolge

Die Byte-Reihenfolge entspricht der Siemens Konvention Big-Endian. Bei der Interpretation der Wörter muss deshalb auf der SPS je nach Konfiguration des Master-Systems ein Byte-Swap erfolgen.

10.2 Datentypen

Daten werden von bzw. zu der Feldbus-Schnittstelle prinzipiell wortweise in 2-Byte-Paketen (16 Bit) übertragen. Es stehen folgende Datentypen zur Verfügung:

Datentyp	Beschreibung
`Bit`	Die 16 Bits eines Worts werden als 16 einzelne Bool-Werte interpretiert
`Integer`	Ein Wort wird als vorzeichenbehaftete Ganzzahl interpretiert
`Double`	Zwei Wörter (4 Byte) werden gemeinsam als eine Gleitkommazahl interpretier. Dies entspricht z.B. bei Siemens dem Datentyp REAL
`Pose`	12 aufeinanderfolgende Wörter werden als sechs Double-Werte einer Pose in folgender Reihenfolge interpretiert: X, Y, Z, R1, R2, R3
`PoseList`	(nur Mikado an SPS): `12 x <Listenlänge>` aufeinanderfolgende Wörter, die als Liste von Posen interpretiert werden. Für eine Liste der Länge 5 werden z.B. 60 aufeinanderfolgende Wörter benötigt.
`StringList`	(nur Mikado an SPS): `2 x <Listenlänge>` aufeinanderfolgende Wörter werden als eine Liste von Double-Werten interpretiert. Strings werden dabei automatisch zu Double konvertiert. Falls ein String nicht zu einem Double konvertierbar ist, kann er nicht über diese Schnittstelle übetragen werden.

10.3 Addressbelegung

Die Adressbelegung definiert die Zuweisung von Daten zu den entsprechenden Speicheradressen. Sowohl für Mikado Control als auch für Mikado Select gibt es eine Standard-Adressbelegung, die alle für den Betrieb unbedingt erforderlichen Datenstrukturen umfasst (siehe unten). Diese Konfigurationen können bei Bedarf auch erweitert und modifiziert werden, um beispielsweise zusätzliche projektspezifische Daten zu übertragen.

Je nach gewählten Einstellungen und verwendetem Protokoll stehen für die Ein- und Ausgänge jeweils bis zu maximal 256 Wörter (512 Byte) zur Verfügung. Alle Bereiche, die nicht in der Standard-Adressbelegung explizit aufgeführt sind, sind standardmäßig unbelegt und können frei konfiguriert werden.

Mikado Control

MIKADO INPUTS

Address	Type	Name
W0.0–7	–	(Reserved)
W0.8	Bit	InLifeBit
W0.9	Bit	InFrontLightOn
W0.10–15	–	(Reserved)
W1	–	(Reserved: gripper bits)
W2	Int	InCmd
W3	Int	InSequenceID
W4	Int	InCameraID
W5	Int	InBoxID
W6	Int	InProductID
W7	Int	InGripperID
W8	Int	InRoiID
W9	Int	InPickSearchParametersID
W10	Int	InDepositID

MIKADO OUTPUTS

Address	Type	Name
W0.0–7	–	(Reserved)
W0.8	Bit	OutLifeBit
W0.9	Bit	OutReadyBit
W0.10	Bit	OutBusyBit
W0.11	Bit	OutRunningBit
W0.12	Bit	OutNoObjectsFoundBit
W0.13	Bit	OutNoPickableObjectsBit
W0.14–15	–	(Reserved)
W1	–	(Reserved: gripper bits)
W2	Int	OutErrorCode
W3	Int	OutNumRecognizedObjects
W4	Int	OutNumPickableObjects
W5	Int	OutPickCategory
W6	Int	OutPickSubcategory
W7	Int	OutPickGripperID
W8-19	Pose	OutPickOffset

Mikado Select

MIKADO INPUTS

Address	Type	Name
W0.0–7	–	(Reserved)
W0.8	Bit	InLifeBit
W0.9	Bit	InFrontLightOn
W0.10–15	–	(Reserved)
W1	–	(Reserved: gripper bits)
W2	Int	InCmd
W3	Int	InSequenceID
W4	Int	InCameraID
W5	Int	InBoxID
W6	Int	InProductID
W7	Int	InGripperID
W8	Int	InRoiID
W9	Int	InPickSearchParametersID
W10–21	Pose	InPose

MIKADO OUTPUTS

Address	Type	Name
W0.0–7	–	(Reserved)
W0.8	Bit	OutLifeBit
W0.9	Bit	OutReadyBit
W0.10	Bit	OutBusyBit
W0.11	Bit	OutRunningBit
W0.12	Bit	OutNoObjectsFoundBit
W0.13	Bit	OutNoPickableObjectsBit
W0.14	Bit	OutPickContainedCollisionsBit
W0.15	–	(Reserved)
W1	–	(Reserved: gripper bits)
W2	Int	OutErrorCode
W3	Int	OutNumRecognizedObjects
W4	Int	OutNumPickableObjects
W5	Int	OutPickCategory
W6	Int	OutPickSubcategory
W7	Int	OutPickGripperID
W8-19	Pose	OutPickOffset
W20-31	Pose	OutPickStartPose
W32-43	Pose	OutPickApproachPose
W44-55	Pose	OutPickGraspPose
W56-67	Pose	OutPickRetractPose
W68-79	Pose	OutPickFinalPose

11 Spezifische Konfiguration für PROFINET

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via Profinet.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

11.1 GSDML

Die Schnittstellenbeschreibung bei PROFINET-Anbindung ist über die GSDML-Datei festgelegt. Nach der Installation des Debian-Pakets `mikado-profinet` findet sich diese Datei unter folgendem Pfad:

`/opt/cifx/gsdml/GSDML-V2.43-OPTONIC-MIKADO-20250210.xml`

Aus Gründen der Rückwärtskompatibilität ist außerdem eine mittlerweile veraltete GSDML-Datei `GSDML-V2.35-HILSCHER-CIFX RE PNS-20190108.xml` vorhanden, siehe *Version 3.3.0*.

Folgendes Profinet-IO Gerät der GSDML ist hierbei am Master einzubinden:

`CIFX RE/PNS V3.5 - V3.5.x`

Der Mikado-Controller stellt je nach ausgewählter Konfiguration folgende Ein-/Ausgangslayouts zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des Profinet-Masters definiert:

- 64 Byte Ein- und Ausgang
- 128 Byte Ein- und Ausgang
- 256 Byte Ein- und Ausgang
- 512 Byte Ein- und Ausgang

Diese Ein- und Ausgänge werden entsprechend der konfigurierten *Adressbelegung* geschrieben bzw. gelesen.

11.2 Standard-Feldbus-Schnittstelle

Die Byte-Reihenfolge, die verfügbaren Datentypen und die Standard-Wortbelegung entsprechen der *Standard-Feldbus-Schnittstelle*.

12 Spezifische Konfiguration für EtherCAT

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via EtherCAT.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

12.1 Schnittstellenbeschreibung

Die Schnittstellenbeschreibung bei EtherCAT-Anbindung ist über XML-Datei festgelegt. Nach der Installation des Debian-Pakets `mikado-profinet` findet sich diese Datei unter folgendem Pfad:

`/opt/cifx/ethercat/Hilscher CIFX RE ECS V4.6.X.{BYTE_GRÖßE}.xml`

Folgendes EtherCAT Gerät ist hierbei am Master einzubinden: `Hilscher CIFX RE ECS`

Der Mikado-Controller stellt je nach ausgewählter Konfiguration folgende Ein-/Ausgangslayouts zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des EtherCAT-Masters definiert:

- 64 Byte Ein- und Ausgang
- 128 Byte Ein- und Ausgang
- 256 Byte Ein- und Ausgang

Diese Ein- und Ausgänge werden entsprechend der konfigurierten *Adressbelegung* geschrieben bzw. gelesen.

12.2 Standard-Feldbus-Schnittstelle

Die Byte-Reihenfolge, die verfügbaren Datentypen und die Standard-Wortbelegung entsprechen der *Standard-Feldbus-Schnittstelle*.

13 Spezifische Konfiguration für EtherNet/IP

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via EtherNet/IP.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

13.1 Schnittstellenbeschreibung

Die Schnittstellenbeschreibung bei EtherNet/IP-Anbindung ist in einer EDS-Datei festgelegt. Nach der Installation des Debian-Pakets `mikado-profinet` findet sich diese Datei unter folgendem Pfad:

`/opt/cifx/ethernet_ip/HILSCHER CIFX-RE EIS V1.1.EDS`

Folgendes EtherNet/IP Gerät ist hierbei am Master einzubinden:

`Hilscher CIFX 50-RE EIS`

Der Mikado-Controller stellt je nach ausgewählter Konfiguration folgende Ein-/Ausgangslayouts zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des EtherNet/IP-Masters definiert:

- 64 Byte Ein- und Ausgang
- 128 Byte Ein- und Ausgang
- 256 Byte Ein- und Ausgang

Diese Ein- und Ausgänge werden entsprechend der konfigurierten *Adressbelegung* geschrieben bzw. gelesen.

Warnung

Die EtherNet/IP-Schnittstelle von Mikado ist standardmäßig für die Zuweisung einer IP-Adresse über DHCP konfiguriert. Diese Einstellung kann derzeit nicht vom Anwender geändert werden. Sollten Sie für Ihre Anwendung eine statische IP-Adresse für die Schnittstelle benötigen, wenden Sie sich bitte an den Mikado Support.

13.2 Standard-Feldbus-Schnittstelle

Die Byte-Reihenfolge, die verfügbaren Datentypen und die Standard-Wortbelegung entsprechen der *Standard-Feldbus-Schnittstelle*.

14 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Kommunikation via Beckhoff ADS.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

14.1 Datentypen

Datentyp Mikado	Datentyp SPS	Bemerkung
`Bool`	`BOOL`	
`Integer`	`INT`	
`Double`	`REAL`	
`Pose`	`ARRAY [1..6] OF REAL`	Die sechs Werte repräsentieren X, Y, Z, R1, R2, R3 der Mikado-Pose
`PoseList`	`ARRAY [1..6*n] OF REAL`	Die Länge des Arrays muss ein Vielfaches von 6 sein. Jeweils 6 benachbarte Werte repräsentieren eine Pose.
`String`	`STRING`	
`StringList`	`ARRAY [1..n] OF REAL`	**Nur Mikado an SPS**. Die Länge des Arrays unterliegt keinen Beschränkungen. Die Werte innerhalb der Mikado-StringList werden zu Doubles konvertiert.

Bemerkung

Ist eine PoseList oder StringList in Mikado länger als der SPS-seitige verfügbare Platz, wird die Mikado-seitige Liste auf die verfügbare Länge abgeschnitten.

14.2 Konfiguration

In der Konfigurationsdatei `ads.yaml` lassen sich zusätzlich zu den Feldbus-Adapter-Parametern folgende ADS-spezifischen Parameter einstellen:

```
# for publishing joint state:
publish_robot_state: false # if true, the robot joint state is published every cycle
current_joint_position_array_name: MAIN.OutCurrentJointPositions # Name of main robot
↪'s joint positions on PLC
main_robot_prefix: "" # prefix as stated in the xacro-macro of main robot (works at␣
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
↪least for staeubli)

# publishing additional robots (optional):
additional_robot1_prefix: "example_prefix" # prefix as stated in xacro-macro of␣
↪additional robot
additional_robot1_variable: MAIN.OutJointPosOfRobotWithExamplePrefix # Name of␣
↪additional robot's joint positions on PLC

additional_robot2_prefix: "example_prefix2"
additional_robot2_variable: MAIN.OutJointPosOfRobotWithExamplePrefix2
# .. and so on

# network settings:
local_net_id: ""
remote_ip: ""
remote_net_id: ""
remote_port: 851

# IOs
input_array_name: GVL.inputBytes  # IO-Name on PLC
output_array_name: GVL.outputBytes # IO-Name on PLC

# robot control:
enable_robot_control: false # if true, the robot is controlled via ADS-interface
publish_robot_status: false # if true, the robot status is published every cycle
trajectory_array_name: MAIN.InTrajectory
in_motion_name: MAIN.OutInMotion
writing_trajectory_allowed_name: MAIN.OutWritingTrajectoryAllowed
```

15 Changelog

Version	Datum	Bemerkung
3.0	30.06.2023	Profinet-Schnittstelle im MIK-Handbuch
3.0.1	29.09.2023	Neusortierung: Allgemeine Schnittstelle und spezielle Profinet-Schnittstelle
3.1.0	11.07.2024	Überarbeitung der Dokumentation, Anpassungen an Mikado 3.1