optonic
vision · automation

Feldbus Schnittstelle

mikado

Version 3.1

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, 2015-2024

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 SPS-Variablen und Mikado-Variablen **5**

5 Rotation-Konvention **7**

6 Fehler-Codes **8**

7 Kommandos **9**
7.1 Programm spezifische Kommandos 9
7.2 Allgemeine Kommandos 11

8 Beispielablauf für Select-Zellen **12**

9 Aktualisierungen **13**
9.1 Version 3.1.0 13

10 Spezifische Konfiguration für Profinet **15**
10.1 GSDML 15
10.2 Byte-Reihenfolge 15
10.3 Datentypen 15
10.4 Standard-Wortbelegung 16

11 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle **18**
11.1 Datentypen 18
11.2 Konfiguration 18

12 Changelog **20**

---

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

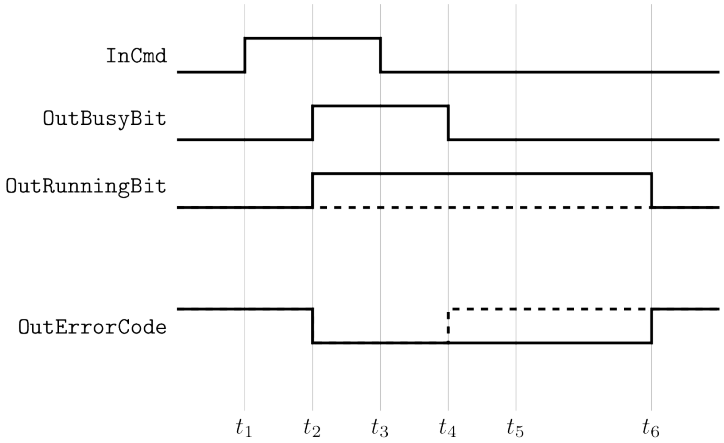
- Starten und Stoppen des Mikado-Programms bzw. einer einzelnen Sequenz innerhalb des Mikado-Programms.
- Die Synchronisation von Mikado-Variablen mit der SPS.

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 `OutReadyBit = 1` von Mikado, welches gesetzt wird, wenn Mikado und Kamera bereit sind.

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) `OutRunningBit = 1` (MIK-Programm starten)
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` Warte auf `OutRunningBit = 0`	
t_6	Lesen von `OutErrorCode` Ggf. Lesen von Variablen	`OutRunningBit = 0` (MIK-Programm endet)



- 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.
- Je nach Kommando wird das `OutRunningBit = 1` und signalisiert, dass Mikado gestartet wurde und das aktuelle Programm abarbeitet. Wird das `OutRunningBit = 0` können der `OutErrorCode` und die entsprechende 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 SPS-Variablen und Mikado-Variablen

SPS-Variablen können auf Mikado-Variablen abgebildet werden und umgekehrt. Das Mapping zwischen SPS-Variablen und Mikado-Variablen kann im Mikado-Assistant unter „Schnittstellen“ angepasst und erweitert werden. Für die verschiedenen, unterstützten Kommunikationsprotokolle (ADS, Profinet) stehen entsprechende Einrichtungsseiten zu Verfügung. Die standardmäßige Feldbus-Konfiguration neu erstellter Zellen enthält alle notwendigen Einträge für die Ausführung der Kommandos.

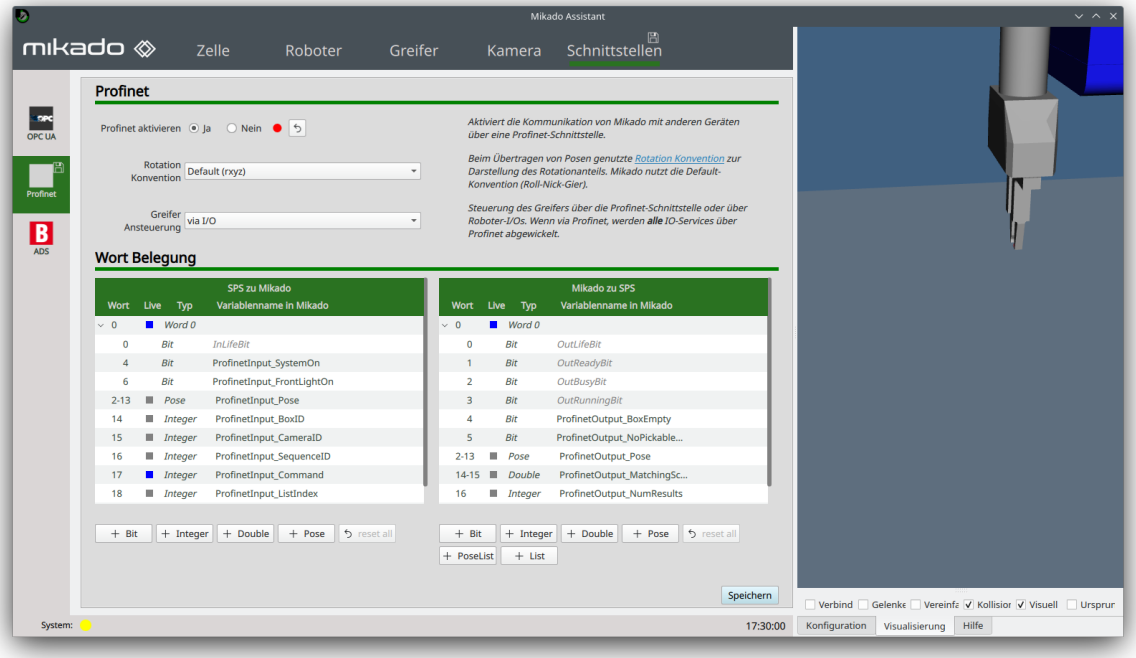


Abb. 1: Assistant zum Anpassen der Wortbelegung und synchronisierter Variablen via Feldbus

Die Feldbus-Konfiguration wird in der Konfigurationsdatei `config/profinet.yaml` bzw. `config/ads.yaml` gespeichert. Die Datei ist folgendermaßen aufgebaut:

```yaml
inputs:  # list of synchronized variables PLC -> Mikado
    - name: <VariableID>  # internal unique ID
      type: <VariableType>  # type of variable, e.g. Bool, Double, Pose, ...
      # Mikado address:
      mikadoVarName: <VariableNameMikadoSide> # variable name in Mikado as stated on Edit>Variables page
      # Fieldbus address:
      word: <AddressProfinetSide>  # only applicable for profinet variables
      bit: <AddressProfinetSide>   # only applicable for profinet variables of type 'Bool'
      remoteName: <VariableNameAdsSide>  # only applicable for ads variables
    - ...
outputs:  # list of synchronized variables Mikado -> PLC
    - ...
```

Alle in dieser Dokumentation erwähnten Namen von synchronisierten Variablen beziehen sich auf `<VariableID>`. Ist beispielsweise vom Einlesen der Variable `InSequenceID` die Rede, wird davon ausgegangen, dass die verwendete Konfigurationsdatei einen Eintrag `name: InSequenceID` enthält.

5 Rotation-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.

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“*

6 Fehler-Codes

Das Ausgangs-Wort `OutErrorCode` kodiert Fehlermeldungen. Folgende Tabelle fasst alle möglichen Fehler-Codes zusammen und erläutert deren Bedeutung.

Tab. 4: Mögliche Fehlermeldungen

Code	Fehlername	Beschreibung
0	`NoError`	Befehl erfolgreich
>0	Fehler im Mikado-Programm	Fehler im Mikado Ablauf. Details siehe Mikado-Programm
-100	`UnexpectedError`	Unerwarteter Fehler aufgetreten
-1	`ServiceCallFailed`	Aufruf fehlgeschlagen
-2	`WrongType`	Falscher Datentyp
-3	`WrongIndex`	Angeforderter Index nicht verfügbar
-4	`NoReplyValue`	Keinen Rückgabe-Wert erhalten
-5	`TooManyReplyValues`	Anzahl der Rückgabewerte höher als erwartet
-6	`ErrorMikadoProgram`	Aufgerufenes Mikado-Programm hat einen Fehler verursacht oder den Ablauf in einem invaliden Zustand verlassen.
-7	`IndexOutOfRange`	Angegebener (Result-)Index außerhalb des Wertebereichs
-8	`UnknownCommand`	unbekanntes Kommando erhalten
-9	`LifebitClientMissing`	Lifebit des Clients wurde nicht erkannt. Outputs werden in diesem Zustand nicht aktualisiert.
-11	`TriggeringFailed`	Ausführung einer Mikado-Sequenz fehlgeschlagen
-12	`ReadingVariableFailed`	Mikado-Variable konnte nicht gelesen werden (existiert die Variable im aktuell geladenen Mikado-Programm?)
-13	`WritingVariableFailed`	Mikado-Variable konnte nicht geschrieben werden (existiert die Variable im aktuell geladenen Mikado-Programm?)
-15	`CancelCalibrationFailed`	Abbrechen einer bereits gestarteten Kalibrierung fehlgeschlagen
-16	`CaptureForCalibrationFailed`	Aufnahme für laufende Kalibrierung nicht erfolgreich
-17	`FinishCalibrationFailed`	Kalibrierung berechnen und speichern war nicht erfolgreich
-18	`StartCalibrationFailed`	Kalibrierung konnte nicht gestartet werden
-19	`SequenceDoesNotExist`	Gewünschte Sequenz existiert nicht
-20	`MissingConfigurationItem`	Feldbus-Konfiguration enthält einen erforderlichen Eintrag nicht

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 SPS-Variablen und Mikado-Variablen*).

Folgende Tabelle listet alle verfügbaren Kommandos auf:

Tab. 5: Verfügbare Kommandos

`InCmd`	**Kommandoname**	**Beschreibung**
0	`Idle`	Wartezustand
Programm spezifische Kommandos		
1	*RunMikadoProg*	Mikado-Programm starten/fortsetzen
2	*StartSequence*	Eine Sequenz 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

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 SPS-Variablen 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.

`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.

Tab. 6: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		**Rückgabewert**
`InSequenceID`	ID der zu startenden Sequenz	*keine*

`PauseMikadoProg`

Mit diesem Kommando wird die Ausführung des aktuellen Programms pausiert.

StopMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt.

ResetMikadoProg

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

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.

Tab. 7: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		**Rückgabewert**
`InCameraID`	ID der zu verwendenden Kamera	*keine*
`InFrontLightOn`	Licht während der Kalibrierung anschalten?	

CaptureCalib

Mit diesem Kommando werden Posen für die Hand-Augen-Kalibrierung gesammelt. Der Kalibrierungsprozess muss vorher durch das Kommando `StartCalib` gestartet worden sein. Es sollten zwischen 10 und 20 Posen gesammelt werden. Dieses Kommando muss dementsprechend häufig ausgeführt werden.

Tab. 8: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		**Rückgabewert**
`InPose`	Derzeitige Tool0-Pose	*keine*

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`.

Tab. 9: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert	**Rückgabewert**	
keine	`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

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.

Tab. 10: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		**Rückgabewert**
`InPose`	Derzeitige Tool0-Pose	*keine*

8 Beispielablauf für Select-Zellen

Anhand des Beispiel-Programms `select_findObjects.program` wird im Folgenden gezeigt, wie sich ein Mikado-Programm über den Feldbus steuern lässt. Das Beispiel-Programm findet sich ab Mikado-Version 2.1 in der Programm-Bibliothek.

Voraussetzungen:

- Es muss eine Produktliste geladen sein, die ein Produkt mit dem Namen `P_0` enthält.
- Es muss eine Szene geladen sein, die eine ROI mit dem Namen `ROI_0` enthält.

Tab. 11: Beispiel-Ablauf für eine Select-Zelle

#	SPS	Mikado
Nur bei mobiler Kamera: Setze Roboterpose (bei statischer Kamera direkt weiter mit #3)		
1	Setze `InPose` auf die aktuelle Pose des Roboterflansch (tool0) Setze `InCmd = 11`	
2		Übernimmt die Werte in `InPose` als aktuelle Roboterpose
Finde Objekte		
3	Setze `InSequenceID = 1` Setze `InBoxID= 0` Setze `InProductID = 0` Setze `InCmd = 2` (`RunSequence`)	
4		Starte Sequenz mit ID = 1 (*Finde Objekte*) Setze `OutNumResults` Ende der Sequenz: `OutRunningBit = 0`
Kommuniziere Ergebnisse		
5	Lese `OutNumResults` Setze `InListIndex = 0` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
6		Starte Sequenz mit ID = 2 (*get Result*) Setze `OutPose` w.r.t. `InListIndex` Ende der Sequenz: `OutRunningBit = 0`
7	Lese `OutPose` Setze `InListIndex = 1` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
8		*wiederhole Schritt 6*
9	*wiederhole Schritt 5 mit* `InListIndex++`	
10	Lese `OutPose` Setze `InListIndex` = `OutNumResults` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
11		*wiederhole Schritt 6*
12	Lese `OutPose`	

9 Aktualisierungen

9.1 Version 3.1.0

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

Bemerkung: Erfolgen SPS-seitige Wertänderungen zur Programm-Laufzeit kann es mit dieser

Version zu unerwartetem Verhalten kommen.

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. 12: 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`	Adapter führt Kommando aus
		`OutRunningBit = 1`	Mikado startet Programm
		`OutBusyBit = 0`	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 Spezifische Konfiguration für Profinet

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

10.1 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-V2.35-HILSCHER-CIFX RE PNS-20190108.xml`

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

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

Der Mikado-Controller stellt folgendes Ein-/Ausgangslayout zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des Profinet-Masters definiert:

- 64 Byte Eingänge
- 64 Byte Ausgänge

10.2 Byte-Reihenfolge

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

10.3 Datentypen

Daten werden prinzipiell wortweise, also in 2-Byte-Pakete, übertragen. Wie oben erwähnt, ist bei der Interpretation der Daten die Byte-Reihenfolge Big-Endian zu beachten. Folgende Datentypen sind möglich:

- `Bits`: Das Wort wird in 16 einzelne Bits zerlegt, wobei jedes Bit als Bool interpretiert wird.
- `Integer`: Das Wort wird als Ganzzahl interpretiert.
- `Double`: Zwei Wörter (32 Bit) werden gemeinsam als eine Gleitkommazahl interpretiert.
- `Pose`: 12 Wörter die als sechs Double-Werte interpretiert werden (X,Y,Z,R1,R2,R3).
- `PoseList` (nur Mikado an SPS): 14 Wörter, die als sechs Double-Werte und zwei Integer-Werte interpretiert werden:

  - `X` Double: x-Wert der Pose an Stelle `Idx`
  - `Y` Double: y-Wert der Pose an Stelle `Idx`
  - `Z` Double: z-Wert der Pose an Stelle `Idx`

- `R1` Double: R1-Wert der Pose an Stelle `Idx`
- `R2` Double: R2-Wert der Pose an Stelle `Idx`
- `R3` Double: R3-Wert der Pose an Stelle `Idx`
- `Len` Integer: Länge der PoseList
- `Idx` Integer: Index der zurückgegebenen Pose innerhalb der PoseList

Beim mehrmaligen Lesen einer PoseList-Variablen wird der Index `Idx` inkrementiert und die ersten 6 Werte ändern sich entsprechend.

- `StringList` (nur Mikado an SPS): 4 Wörter, die als ein Double-Wert und zwei Integer-Werte interpretiert werden:
  - `Val` Double: Zu Double konvertierter Wert der StringList an Stelle `Idx`
  - `Len` Integer: Länge der StringList
  - `Idx` Integer: Index des zurückgegebenen Strings innerhalb der StringList

  Beim mehrmaligen Lesen einer StringList-Variablen wird der Index `Idx` inkrementiert und der erste Wert ändert sich entsprechend.

10.4 Standard-Wortbelegung

Größtenteils fungieren die verwendeten Wörter zur Übertragung der Werte von Mikado-Variablen. Davon ausgenommen sind die in Abbildung *Übersicht der Standard-Belegung von Input und Output Byte-Arrays* grau gekennzeichneten Wörter, deren Werte unabhängig vom Mikado-Programm gesetzt werden.

SPS zu Mikado

Word	Byte	Bit		
0	0	0	|	InLifeBit (live)
		1		
		2		
		3		
		4	|	InSystemOn (live)
		5		
		6	|	InFrontLightOn (live)
		7		
	1	8		
		9		
		10		
		11		
		12		
		13		
		14		
		15		
1	2		|	*reserviert für Bits*
	3			
2	4		| Double	InPoseX
	5			
3	6			
	7			
4	8		| Double	InPoseY
	9			
5	10			
	11			
6	12		| Double	InPoseZ
	13			
7	14			
	15			
8	16		| Double	InPoseR1
	17			
9	18			
	19			
10	20		| Double	InPoseR2
	21			
11	22			
	23			
12	24		| Double	InPoseR3
	25			
13	26			
	27			
14	28		| Integer	InBoxID
	29			
15	30		| Integer	InCameraID
	31			
16	32		| Integer	InSequenceID
	33			
17	34		| Integer	InCmd (live)
	35			
18	36		| Integer	InListIndex
	37			
19	38		| Integer	InProductID
	39			
20	40			
	41			
21	42			
	43			
22	44			
	45			

Mikado zu SPS

Word	Byte	Bit		
0	0	0	|	OutLifeBit (live)
		1	|	OutReadyBit (live)
		2	|	OutBusyBit (live)
		3	|	OutRunningBit (live)
		4	|	OutBoxEmptyBit (live)
		5	|	OutNoPickableObjectsBit (live)
		6		
		7		
	1	8		
		9		
		10		
		11		
		12		
		13		
		14		
		15		
1	2		|	*reserviert für Bits*
	3			
2	4		| Double	OutPoseX
	5			
3	6			
	7			
4	8		| Double	OutPoseY
	9			
5	10			
	11			
6	12		| Double	OutPoseZ
	13			
7	14			
	15			
8	16		| Double	OutPoseR1
	17			
9	18			
	19			
10	20		| Double	OutPoseR2
	21			
11	22			
	23			
12	24		| Double	OutPoseR3
	25			
13	26			
	27			
14	28		| Double	OutMatchingScore
	29			
15	30			
	31			
16	32		| Integer	OutNumResults
	33			
17	34		| Integer	OutErrorCode (live)
	35			
18	36			
	37			
19	38			
	39			
20	40			
	41			
21	42			
	43			
22	44			
	45			

Abb. 2: Übersicht der Standard-Belegung von Input und Output Byte-Arrays

11 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle

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

11.1 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 und 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.

11.2 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 #␣
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
 ↪Name of main robot's joint positions on PLC
main_robot_prefix: "" # prefix as stated in the xacro-macro of main␣
 ↪robot (works at 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
```

12 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