optonic
vision · automation

Feldbus Schnittstelle

mikado

Version 3.0

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, 2015-2024

Inhaltsverzeichnis

1 Kurzbeschreibung der Schnittstelle **3**

2 Genereller Ablauf der Kommunikation **3**

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

4 Synchronisieren von SPS-Variablen und Mikado-Variablen **5**
4.1 Live-Variablen 6

5 Rotation-Konvention **6**

6 Fehler-Codes **7**

7 Kommandos **8**
7.1 `RunMikadoProg` 9
7.2 `StartSequence` 10
7.3 `PauseMikadoProg` 10
7.4 `StopMikadoProg` 10
7.5 `ResetMikadoProg` 10
7.6 `SyncAll` 10
7.7 `ResetInternalIndices` 11
7.8 `StartCalib` 11
7.9 `CaptureCalib` 11
7.10 `FinishCalib` 12
7.11 `readPoseAndPublishTool0` 12

8 Beispielablauf für Select-Zellen **12**

9 Spezifische Konfiguration für Profinet **14**
9.1 GSDML 14
9.2 Byte-Reihenfolge 14
9.3 Datentypen 14
9.4 Standard-Wortbelegung 15

10 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle **17**
10.1 Datentypen 17
10.2 Konfiguration 17

11 Changelog **19**

1 Kurzbeschreibung der Schnittstelle

Die Schnittstelle bietet die folgenden drei grundlegenden Funktionen:

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

Für die ersten beiden genannten Funktionen muss ein entsprechendes Programm in Mikado geladen sein.

2 Genereller Ablauf der Kommunikation

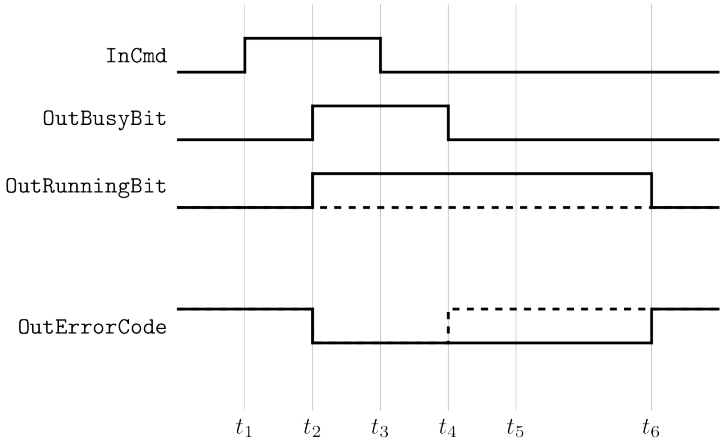
Die Kommunikation wird realisiert mittels Kommandos (siehe *Kommandos*) die von SPS an Mikado gesendet werden. Die folgende Ablaufbeschreibung gilt für alle Kommandos. Bedingung für die Kommunikation ist `OutReadyBit = 1` von Mikado, welches gesetzt wird, wenn Mikado und Kamera bereit sind (siehe *Status der Kommunikation*). Das SPS-Kommando `InCmd` darf nur gesetzt werden, wenn `OutBusyBit = 0`.

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`
		Ausführung des Befehls
		`OutRunningBit = 1` (MIK-Programm starten)

t_3	`InCmd` zurücksetzen auf `0` Warte auf `OutBusyBit = 0`	
t_4		`OutErrorCode` schreiben `OutBusyBit = 0`
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 diese 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 Mikado-Profinet-Adapter 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
`OutReadyBit`	System-Status von Mikado.	True = Mikado betriebsbereit False = Mikado nicht bereit
`OutBusyBit`	Status des Profinet-Adapters.	True = Kommando wird entgegengenommen und verarbeitet. False = Adapter ist bereit neues Kommando entgegenzunehmen
`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.

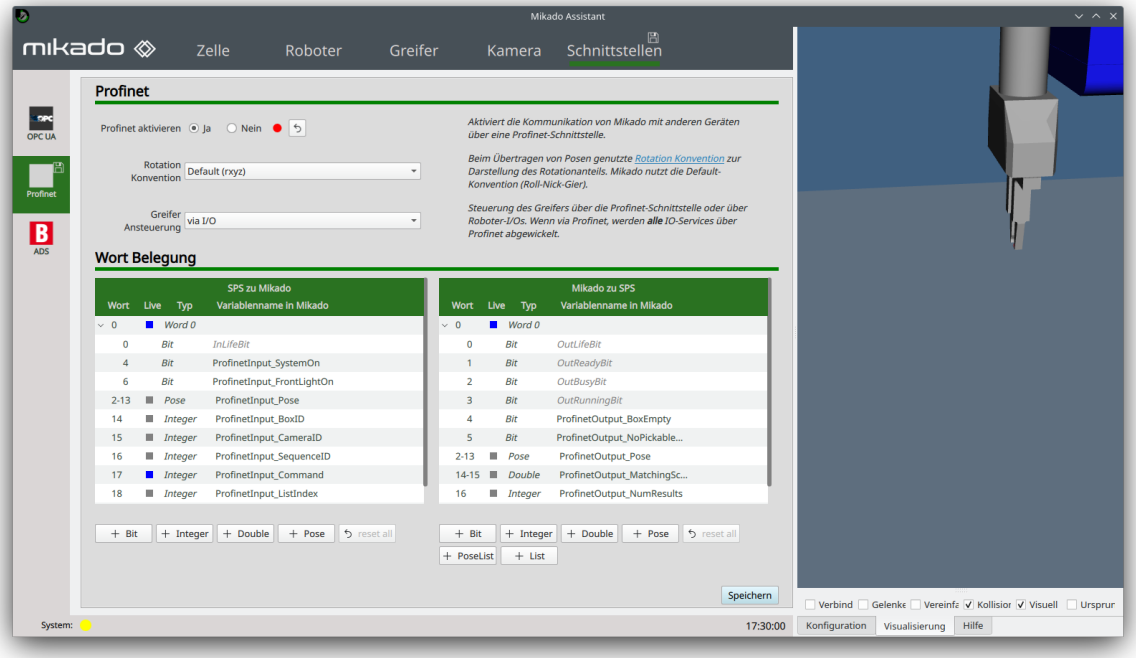


Abb. 1: Assistant zum Anpassen der Wortbelegung und synchronisierter Mikado-Variablen via Profinet

4.1 Live-Variablen

Die mit *live* gekennzeichneten Aus- und Eingangswörter werden in jedem Zyklus synchronisiert. Dies entspricht standardmäßig einer Rate von 50 Hz. Alle anderen Variablen werden nur bei Bedarf synchronisiert oder durch den Aufruf des Kommandos `SyncAll`.

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

7 Kommandos

Bemerkung: Wichtig: Vor dem Setzen des Kommandos zuerst zugehörige Parameter setzen.

Die notwendigen Informationen für ein Kommando werden zu Beginn des Kommandos eingelesen. Variablen und Ergebnisse hängen von der Implementierung im Mikado-Programm ab. Die meisten Output-Wörter repräsentieren Mikado-Variablen und können im Mikado-Programm

individuell bearbeitet werden. Beispielsweise können so eigene Fehlercodes über ProgramErrorCode nach außen gegeben werden.

Tab. 5: Verfügbare Kommandos

InCmd	Kommandoname	Beschreibung
0	`Idle`	Wartezustand
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
6	*SyncAll*	Handshake ausführen um alle Input-Variablen einzulesen und alle Output-Variablen zu schreiben
7	*ResetInternalIndices*	Setzt den Index aller ausgegebenen PoseListen und StringListen auf -1.
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 RunMikadoProg

Mit diesem Kommando wird das aktuelle Mikado-Programm gestartet oder fortgesetzt. Die notwendigen Informationen für das Kommando werden zu Beginn des Kommandos eingelesen:

- `InProductID`
- `InBoxID`
- `InListIndex`

Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = True` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, kann der `ErrorCode` ausgewertet werden und die Ergebnisse mit dem Kommando

- `SyncAll`

abgerufen werden.

7.2 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 ID gesendeten ID sein, beispielsweise die Ganzzahl ‚42'. Die notwendigen Informationen für das Kommando werden zu Beginn des Kommandos eingelesen:

- `InSequenceID`
- `InProductID`
- `InBoxID`
- `InListIndex`

Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = 1` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, kann der `ErrorCode` ausgewertet werden und die Ergebnisse mit dem Kommando

- `SyncAll`

abgerufen werden.

7.3 PauseMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms pausiert.

7.4 StopMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt.

7.5 ResetMikadoProg

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

7.6 SyncAll

Alle Input-Variablen von der SPS lesen, sowie alle Output-Variablen von Mikado auf die Profinet-Schnittstelle schreiben.

Bemerkung: Anmerkung zum Verhalten bei synchronisierten Listen:

Für Mikado-Variablen vom Typ `PoseList` oder `StringList` (siehe auch Abschnitt *Datentypen*) wird bei diesem Kommando das nächste Element in der Mikado-Variable geholt. Ist der aktuelle Index bereits beim letzten Messergebnis, so beginnt der Index wieder beim ersten Ergebnis (0). Listenwert an der Stelle Index, Länge der Liste und aktueller Index liegen danach an

den Ausgabedaten des Feldbusses an. Durch wiederholtes Aufrufen dieses Kommandos können somit iterativ alle Werte der Liste übertragen werden.

7.7 ResetInternalIndices

Setzt den Index aller ausgegebenen PoseListen und StringListen auf -1. Beim nächsten Aufruf von `SyncAll` liegt somit das erste Element (idx=0) an den Ausgabedaten des Feldbusses an.

7.8 StartCalib

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

Mit diesem Kommando wird die Hand-Augen-Kalibrierung gestartet. Die notwendigen Informationen für das Kommando werden zu Beginn eingelesen:

- `InCameraID`
- `InFrontLightOn`

7.9 CaptureCalib

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

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. Die notwendigen Informationen für ein Kommando werden zu Beginn des Kommandos eingelesen:

- `InPose`

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

7.10 FinishCalib

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

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`.

7.11 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. Die notwendigen Informationen für das Kommando werden zu Beginn des Kommandos eingelesen:

- `InPose`

8 Beispielablauf für Select-Zellen

Anhand des Beispiel-Programms `select_findObjects.program` wird im Folgenden gezeigt, wie sich Mikado über Profinet 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. 6: 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	Setze `InCmd = 6` (`syncAll`) 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	Setze `InCmd = 6` (`syncAll`) Lese `OutPoseX`, `OutPoseY`, ... Setze `InListIndex = 1` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
8		*wiederhole Schritt 6*
9	*wiederhole Schritt 5 mit* `InListIndex++`	
10	Setze `InCmd = 6` (`syncAll`) Lese `OutPoseX`, `OutPoseY`, ... Setze `InListIndex` = `OutNumResults` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
11		*wiederhole Schritt 6*
12	Setze `InCmd = 6` (`syncAll`) Lese `OutPoseX`, `OutPoseY`, ...	

9 Spezifische Konfiguration für Profinet

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

9.1 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

9.2 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.

9.3 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.

9.4 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.

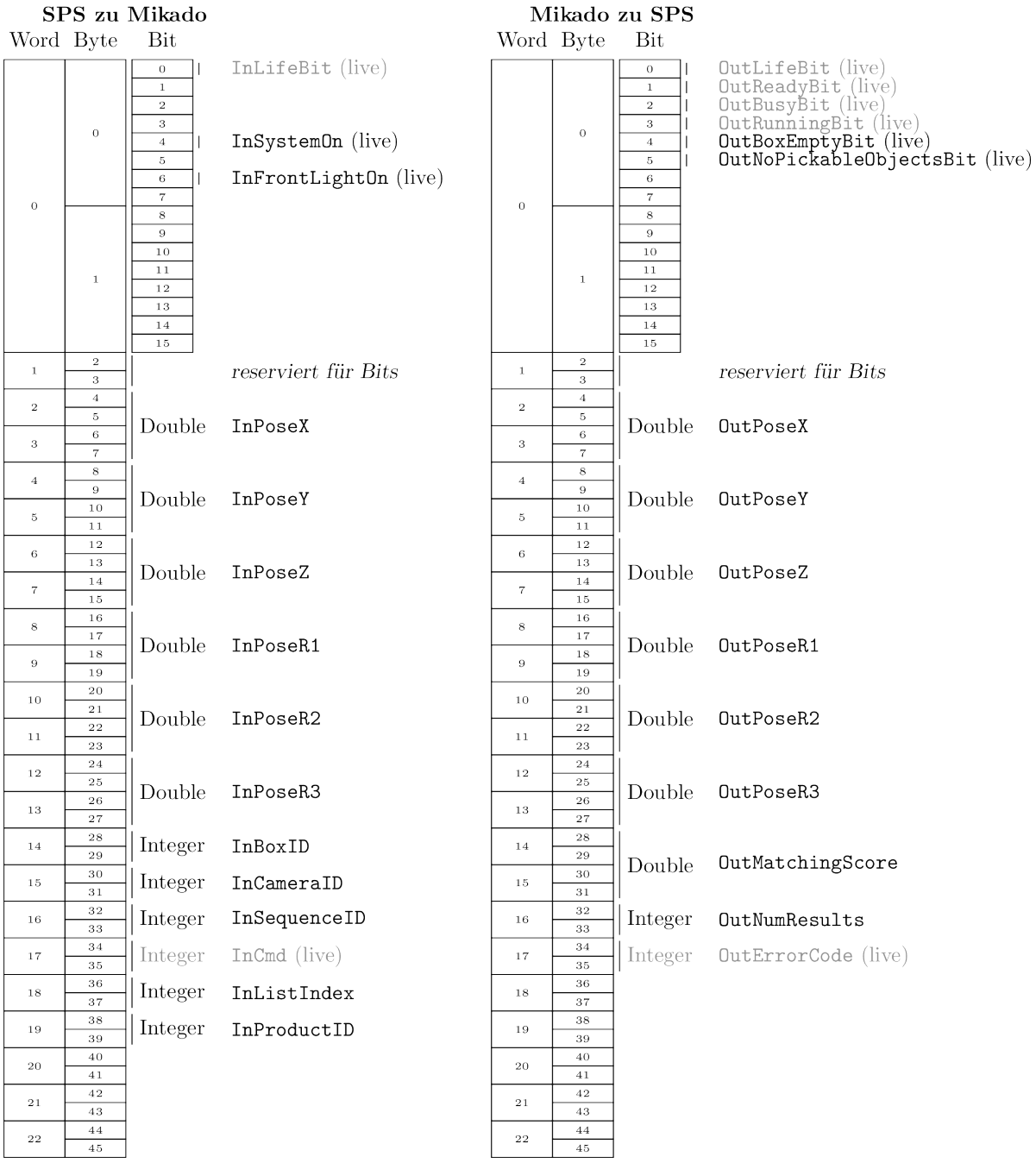


Abb. 2: Übersicht der Standard-Belegung von Input und Output Byte-Arrays

10 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle

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

10.1 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.

10.2 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
```

11 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