optonic
vision · automation

Fieldbus Interface

mikado

Version 3.1

A product of 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

Contents

1 Brief description of the interface **2**

2 General communication process **3**

3 Communication status **4**
3.1 LifeBit 4
3.2 Mikado status 4

4 Synchronizing PLC variables and Mikado variables **5**

5 Rotation convention **6**

6 Error codes **7**

7 Commands **8**
7.1 Program specific commands 8
7.2 General Commands 10

8 Example sequence for select cells **11**

9 Updates **13**
9.1 Version 3.1.0 13

10 Specific configuration for Profinet **14**
10.1 GSDML 14
10.2 Byte sequence 15
10.3 Data types 15
10.4 Standard word assignment 16

11 Specific configuration for Beckhoff ADS interface **17**
11.1 Data types 17
11.2 Configuration 17

12 Changelog **19**

---

1 Brief description of the interface

The core functions of Mikado can be called up via the fieldbus interface. A distinction is made between general functions and functions for controlling a loaded Mikado program.

General functions

- Carry out the hand-eye calibration.

Functions for controlling a Mikado program

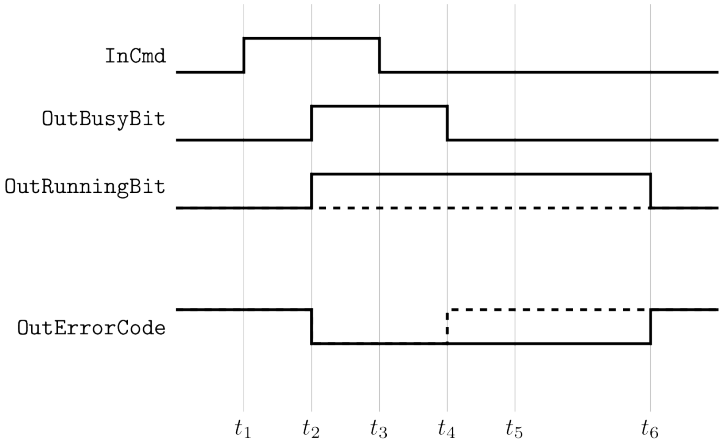
- Start and stop the Mikado program or an individual sequence within the Mikado program.
- The synchronization of Mikado variables with the PLC.

2 General communication process

Communication is realized by means of commands (see *Commands*) that are sent from the PLC to Mikado. The status of the communication is described using status bits (see *Communication status*). The following process description applies to all commands. The condition for communication is `OutReadyBit = 1` from Mikado, which is set when Mikado and camera are ready.

Table 1: General communication process

Time	SPS	Mikado
$< t_1$	Wait for `OutReadyBit = 1` Wait for `OutBusyBit = 0` Set parameters	Waiting on command
t_1	Set `InCmd` Wait for `OutBusyBit = 1`	
t_2		`OutErrorCode = 0` `OutBusyBit = 1` (fieldbus adapter processes command) `OutRunningBit = 1` (start MIK program)
t_3	Reset `InCmd` to `0` Wait for `OutBusyBit = 0`	
t_4		Write `OutErrorCode` `OutBusyBit = 0` (fieldbus adapter has processed command)
t_5	Reading `OutErrorCode` Wait for `OutRunningBit = 0`	
t_6	Reading `OutErrorCode` Read variables if necessary	`OutRunningBit = 0` (MIK program ends)



- If a command is set (`InCmd` $\neq$ `0`), Mikado responds with `OutBusyBit = 1` and first resets the `OutErrorCode`. The PLC responds to `OutBusyBit = 1` with `InCmd = 0` and then waits for `OutBusyBit = 0`.
- The PLC evaluates the `OutErrorCode`. If this has the value 0, the command has been executed successfully.
- Depending on the command, the `OutRunningBit = 1` and signals that Mikado has been started and is processing the current program. If the `OutRunningBit = 0`, the `OutErrorCode` and the corresponding output variables can be read out.

3 Communication status

3.1 LifeBit

A LifeBit is used to check the status of communication between Mikado and the PLC. The PLC reads the value of `OutLifeBit` and writes it unchanged to `InLifeBit`. The fieldbus adapter reads the value of `InLifeBit` and writes the negated value to `OutLifeBit`. If the value of `OutLifeBit` does not change regularly, communication is considered interrupted and a corresponding error is displayed in the Mikado status.

3.2 Mikado status

The status of Mikado is transmitted to the PLC using three different bits. The following table summarizes the meaning of the individual bits.

Table 2: Meaning of the individual status bits

Name	Explanation	Value
`OutBusyBit`	Status of the fieldbus adapter.	True = Command is processed. False = Adapter is ready to accept new command
`OutReadyBit`	System status of Mikado.	True = Mikado ready for operation False = Mikado not ready
`OutRunningBit`	Program status of Mikado	True = Mikado program is running (corresponds to green or yellow in Mikado) False = program is not running (corresponds in Mikado: orange, red or grey)

Note: New commands are only accepted if `OutBusyBit` = False

4 Synchronizing PLC variables and Mikado variables

PLC variables can be mapped to Mikado variables and vice versa. The mapping between PLC variables and Mikado variables can be adapted and extended in the Mikado Assistant under “Interfaces”. Corresponding setup pages are available for the various supported communication protocols (ADS, Profinet). The standard fieldbus configuration of newly created cells contains all the necessary entries for executing the commands.

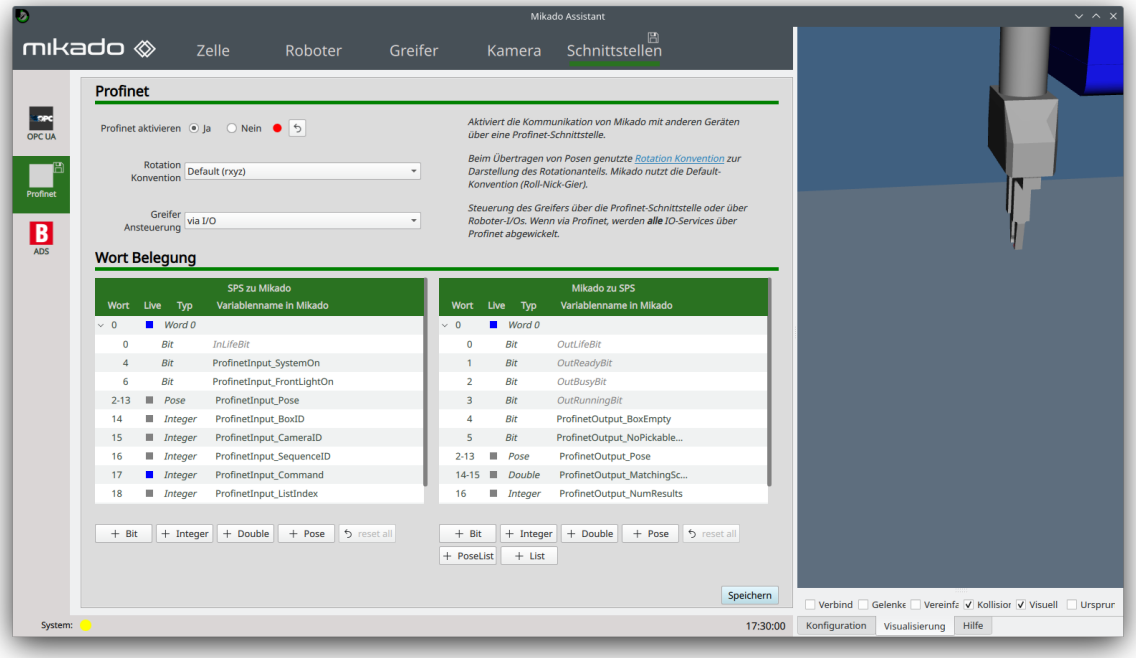


Fig. 1: Assistant for adapting the word assignment and synchronized Mikado variables via fieldbus

The fieldbus configuration is saved in the configuration file `config/profinet.yaml` or `config/ads.yaml`. The file is structured as follows:

```
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␣
```

(continues on next page)

(continued from previous page)

```
↪variables
    - ...
outputs:  # list of synchronized variables Mikado -> PLC
    - ...
```

All names of synchronized variables mentioned in this documentation refer to <VariableID>. If, for example, the variable InSequenceID is mentioned, it is assumed that the configuration file used contains an entry name: InSequenceID.

5 Rotation convention

The rotation convention defines how the orientation of poses is transferred to the interface or read from the interface. The orientation is always specified in the form of Euler angles, which can be interpreted in different ways depending on the convention (see below).

The rotation convention is specified as a combination of four letters: **r** (**r**elative, intrinsic, rotated coordinate system) or **s** (**s**tatic ,extrinsic, fixed coordinate system) followed by the order of the three rotation axes **x/y/z**. This theoretically results in 24 different combinations in which the orientation of poses can be represented in Euler angles.

The following table lists the rotation conventions for some robot manufacturers. The convention must be set to match the robot used in the Mikado Assistant under the "Interfaces" tab so that the alignment of poses can be interpreted correctly.

Mikado uses the convention *rxyz* (roll-nick-yaw / RPY) as standard. If this convention is selected, the Mikado poses are passed on to the interface unchanged. If a different setting is selected, the rotation part of a pose is converted by the interface into the corresponding format.

Table 3: Supported conventions

ID	Convention	Robot manufacturer	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 Error codes

The output word `OutErrorCode` encodes error messages. The following table summarizes all possible error codes and explains their meaning.

Table 4: Possible error messages

Code	Error name	Description
0	`NoError`	Command successful
>0	Error in the Mikado program	Error in the Mikado program. For details see Mikado program
-100	`UnexpectedError`	Unexpected error occurred
-1	`ServiceCallFailed`	Call failed
-2	`WrongType`	Incorrect data type
-3	`WrongIndex`	Requested index not available
-4	`NoReplyValue`	No return value received
-5	`TooManyReplyValues`	Number of return values higher than expected
-6	`ErrorMikadoProgram`	Called Mikado program has caused an error or left the sequence in an invalid state.
-7	`IndexOutOfRange`	Specified (result) index outside the value range
-8	`UnknownCommand`	unknown command received
-9	`LifebitClientMissing`	Lifebit of the client was not recognized. Outputs are not updated in this state.
-11	`TriggeringFailed`	Execution of a Mikado sequence failed
-12	`ReadingVariableFailed`	Mikado variable could not be read (does the variable exist in the currently loaded Mikado program?)
-13	`WritingVariableFailed`	Mikado variable could not be written (does the variable exist in the currently loaded Mikado program?)
-15	`CancelCalibrationFailed`	Aborting an already started calibration failed
-16	`CaptureForCalibrationFailed`	Recording for ongoing calibration not successful
-17	`FinishCalibrationFailed`	Calculate and save calibration was not successful
-18	`StartCalibrationFailed`	Calibration could not be started
-19	`SequenceDoesNotExist`	Requested sequence does not exist
-20	`MissingConfigurationItem`	Fieldbus configuration does not contain a required entry

7 Commands

All commands have at least `OutErrorCode` as a return value. Some commands have additional input and return values. The additional input and return values are listed in the detailed description of the commands, if they exist.

Note: The input values must be present on the fieldbus **before** the command is set so that they can be taken into account. The return values are present on the fieldbus as soon as `OutBusyBit == 0` and the configuration file contains the corresponding entries (see *Synchronizing PLC variables and Mikado variables*).

The following table lists all available commands:

Table 5: Available commands

InCmd	Command name	Description
0	`Idle`	Waiting
Program specific commands		
1	*RunMikadoProg*	Start/continue Mikado program
2	*StartSequence*	Start a sequence. SequenceID must first be set for this
3	*PauseMikadoProg*	Stop process after execution of the current task
4	*StopMikadoProg*	Stop the Mikado program
5	*ResetMikadoProg*	Stop Mikado program and reset all variables and states in the Mikado
General commands		
8	*StartCalib*	Start hand-eye calibration
9	*CaptureCalib*	Triggers image acquisition and measurement of the calibration pattern
10	*FinishCalib*	Calculate and save hand-eye calibration
11	*readPoseAndPublishTool0*	Sets the pose of the robot flange in Mikado

7.1 Program specific commands

Program-specific commands refer to the current Mikado program and can only be executed if a program is loaded. These commands have no additional return values. The desired results depend on the implementation in the Mikado program and are communicated individually via the synchronized variables (see *Synchronizing PLC variables and Mikado variables*).

Conversely, the behaviour of the Mikado program can be influenced via the fieldbus by using synchronized variables as inputs for tasks in the Mikado program. It should be noted that a synchronized variable used as task input in the Mikado program sequence is read out at the time of task execution. Therefore, synchronized variables on the PLC side should only be changed if no Mikado program is running.

Note: To avoid errors in the program sequence, synchronized variables on the PLC side should only be changed if no Mikado program is running (`OutRunningBit = 0`).

RunMikadoProg

This command is used to start or continue the current Mikado program. After successful execution of the command, the `RunningBit = True` changes and signals the running Mikado program. As soon as this is reset by Mikado, the synchronized variables can be evaluated.

StartSequence

This command executes the Mikado sequence with the ID stored in the word `SequenceID`. From program version 3, the ID of a sequence can be set in the text field to the right of the sequence name. In earlier program versions, the sequence name must be the same as the ID sent, for example the integer "42".

After successful execution of the command, the `RunningBit = 1` changes and signals the running Mikado program. As soon as this is reset by Mikado, the synchronized variables can be evaluated.

Table 6: Additional input and return values

Input value		Return value
`InSequenceID`	ID of the sequence to be started	*none*

PauseMikadoProg

This command pauses the execution of the current program.

StopMikadoProg

This command stops the execution of the current program.

ResetMikadoProg

This command stops the execution of the current program and resets the status of the program and all variables.

7.2 General Commands

General commands can be executed independently of the Mikado program. The return values are at the address defined in the configuration file as soon as `OutBusyBit == 0`.

`StartCalib`

This command starts the hand-eye calibration.

Table 7: Additional input and return values

Input value		Return value
`InCameraID`	ID of the camera to be used	*none*
`InFrontLightOn`	Switch on the light during calibration?	

`CaptureCalib`

This command is used to collect poses for hand-eye calibration. The calibration process must have been started beforehand using the `StartCalib` command. Between 10 and 20 poses should be collected. This command must therefore be executed frequently.

Table 8: Additional input and return values

Input value		Return value
`InPose`	Current pose of tool0	*none*

Example procedure:

- Robot strikes a pose
- PLC writes the current flange pose (tool0) to `InPose`
- The command `CaptureCalib` is triggered
- Mikado reads the current robot pose from `InPose` and takes a picture

`FinishCalib`

This command is used to calculate the hand-eye calibration. This is the last step in the hand-eye calibration process. This takes place after a sufficient number of collected pattern poses with the help of `CaptureCalib`.

Table 9: Additional input and return values

Input value	Return value	
none	`OutCalibMaxDist`	Maximum pose error distance
	`OutCalibMeanDist`	Mean pose error distance
	`OutCalibMeanX`	Mean pose error x
	`OutCalibMeanY`	Mean pose error y
	`OutCalibMeanZ`	Mean pose error z
	`OutCalibMeanAngle`	Mean pose error angle

`readPoseAndPublishTool0`

Note: This command is generally only used for select cells.

This command reads in a pose transmitted by the PLC and sets it in Mikado as the current pose of the robot flange (tool0). The position of the robot during image acquisition must be known for the Mikado program sequence, whereby Mikado Select itself has no information about the robot status. It is not necessary to use this command for hand-eye calibration, as the `CaptureCalib` command already performs the same functions in the background.

Table 10: Additional input and return values

Input value		Return value
`InPose`	Current pose of tool0	*none*

8 Example sequence for select cells

The example program `select_findObjects.program` is used below to show how a Mikado program can be controlled via the fieldbus. The sample program can be found in the program library from Mikado version 2.1.

Prerequisites:

- A product list must be loaded that contains a product with the name `P_0`.
- A scene must be loaded that contains an ROI with the name `ROI_0`.

Table 11: Example procedure for a select cell

#	SPS	Mikado
Only for mobile camera: Set robot pose (for static camera continue directly with #3)		
1	Set `InPose` to the current pose of the robot flange (tool0) Set `InCmd = 11`	
2		Adopts the values in `InPose` as the current robot pose
Find objects		
3	Set `InSequenceID = 1` Set `InBoxID= 0` Set `InProductID = 0` Set `InCmd = 2` (`RunSequence`)	
4		Start sequence with ID = 1 (*Find objects*) Set `OutNumResults` End of the sequence: `OutRunningBit = 0`
Communicate results		
5	Lese `OutNumResults` Set `InListIndex = 0` Set `InSequenceID = 2` Set `InCmd = 2` (`RunSequence`)	
6		Start sequence with ID = 2 (*get Result*) Set `OutPose` w.r.t. `InListIndex` End of the sequence: `OutRunningBit = 0`
7	Lese `OutPose` Set `InListIndex = 1` Set `InSequenceID = 2` Set `InCmd = 2` (`RunSequence`)	
8		*repeat step 6*
9	*repeat step 5 with* `InListIndex++`	
10	Lese `OutPose` Set `InListIndex` = `OutNumResults` Set `InSequenceID = 2` Set `InCmd = 2` (`RunSequence`)	
11		*repeat step 6*
12	Lese `OutPose`	

9 Updates

9.1 Version 3.1.0

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

Note: If PLC-side value changes are made at program runtime, unexpected behavior may occur with this version.

The new fieldbus adapter version brings the significant innovation that synchronized variables are constantly updated. In previous versions, variables had to be updated with the `syncAll` command. This change has the following consequences:

- The user-defined live property of variables no longer applies because all variables are now "live".
- The `syncAll` command is superfluous and is therefore no longer listed in this documentation. The command can still be sent. The `OutErrorCode` returned in this case is `NoError`.
- If PLC-side value changes are made at program runtime, unexpected behavior may occur.

The following table shows an example of how PLC-side changes can lead to unexpected behavior at an inopportune time. Assume a program is loaded in Mikado that calls the task `Find Sceneobject`, among others. The synchronized variable `InBoxID` is used as task input.

Table 12: Example of PLC-side changes at an unfavorable time

TimeSPS		Mikado	Comment
t_1	Set `InBoxID = 42` Set `InCmd = 1` (start program)		PLC sets the variable `InBoxID = 42` and sends start command

t_2		`OutBusyBit = 1`
		`OutRunningBit = 1`
		`OutBusyBit = 0`
			Adapter executes command
			Mikado starts program
			Command execution completed

t_3	`InBoxID = 0` (**false**) `InCmd = 0`		**This is an unfavorable time.** Variables used in the program sequence should not be changed at runtime.
t_4		Program calls `FindSceneObject (InBoxID)`	Mikado has now arrived at the task call. The current `InBoxID = 0` is used.
t_5		`OutRunningBit = 0`	End of program reached
t_6	`InBoxID = 0` (**correct**)		**The right time to reset variables.**

10 Specific configuration for Profinet

This section describes the steps required to set up the hardware for communication via Profinet.

10.1 GSDML

The interface description for ProfiNet connection is defined via the GSDML file. After installing the Debian package `mikado-profinet`, this file can be found under the following path:

`/opt/cifx/gsdml/GSDML-V2.35-HILSCHER-CIFX RE PNS-20190108.xml`

The following Profinet IO device of the GSDML must be integrated into the master:

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

The Mikado controller provides the following input/output layout. Inputs and outputs are defined from the perspective of the Profinet master:

- 64 byte inputs
- 64 byte outputs

10.2 Byte sequence

The byte sequence corresponds to the Siemens big-endian convention. When interpreting the words, a byte swap must therefore take place depending on the configuration of the master system.

10.3 Data types

In principle, data is transmitted word by word, i.e. in 2-byte packets. As mentioned above, the big-endian byte sequence must be observed when interpreting the data. The following data types are possible:

- `Bits`: The word is broken down into 16 individual bits, whereby each bit is interpreted as a bool.
- `Integer`: The word is interpreted as an integer.
- `Double`: Two words (32 bits) are interpreted together as a floating point number.
- `Pose`: 12 words which are interpreted as six double values (X,Y,Z,R1,R2,R3).
- `PoseList` (Mikado to PLC only): 14 words, which are interpreted as six double values and two integer values:

  - `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

  When a PoseList variable is read several times, the index `Idx` is incremented and the first 6 values change accordingly.

- `StringList` (Mikado to PLC only): 4 words, which are interpreted as one double value and two integer values:
  - `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

  When a StringList variable is read several times, the index `Idx` is incremented and the first value changes accordingly.

10.4 Standard word assignment

For the most part, the words used function to transfer the values of Mikado variables. The words marked in grey in Figure *Overview of the standard assignment of input and output byte arrays*, whose values are set independently of the Mikado program, are excluded from this.

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			

Fig. 2: Overview of the standard assignment of input and output byte arrays

11 Specific configuration for Beckhoff ADS interface

This section describes the steps required to set up communication via Beckhoff ADS.

11.1 Data types

Data type Mikado	Data type PLC	Remark
`Bool`	`BOOL`	
`Integer`	`INT`	
`Double`	`REAL`	
`Pose`	`ARRAY [1..6] OF REAL`	The six values represent x,y,z,R1,R2 and R3 of the Mikado pose
`PoseList`	`ARRAY [1..6*n] OF REAL`	The length of the array must be a multiple of 6. Each 6 neighboring values represent a pose.
`String`	`STRING`	
`StringList`	`ARRAY [1..n] OF REAL`	**Mikado to PLC only**. The length of the array is not subject to any restrictions. The values within the Mikado-StringList are converted to doubles.

Note: If a PoseList or StringList in Mikado is longer than the available space on the PLC side, the Mikado-side list is truncated to the available length.

11.2 Configuration

In addition to the fieldbus adapter parameters, the following ADS-specific parameters can be set in the configuration file `ads.yaml`:

```
# 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 least for staeubli)
```

(continues on next page)

(continued from previous page)

```yaml
# 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	Date	Comment
3.0	30.06.2023	Profinet-Interface described within manual
3.0.1	29.09.2023	Re-sorting: General interface and special Profinet interface
3.1.0	11.07.2024	Revision of the documentation, adaptations to Mikado 3.1