optonic
vision · automation

Fieldbus Interface

mikado

Version 3.3

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, 2015-2025

Contents

1 Brief description of the interface **3**

2 General communication process **3**

3 Communication status **4**
3.1 LifeBit . . . 4
3.2 Mikado status . . . 5

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

5 Rotation conventions **6**

6 Error codes **7**

7 Commands **9**
7.1 Program specific commands . . . 9
7.2 General Commands . . . 10

8 Example sequence for select cells **12**

9 Updates of the fieldbus interface **13**
9.1 Version 3.1.0 . . . 13
9.2 Version 3.1.2 . . . 14

10 Standard Fieldbus Interface **14**
10.1 Byte Order . . . 14
10.2 Fieldbus Data Types . . . 14
10.3 Standard Word Assignment . . . 14

11 Specific configuration for Profinet **16**
11.1 GSDML . . . 16
11.2 Standard Fieldbus Interface . . . 16

12 Spezifische Konfiguration für EtherCAT **16**
12.1 Interface description . . . 16
12.2 Standard Fieldbus Interface . . . 17

13 Spezifische Konfiguration für EtherNet/IP **17**
13.1 Interface description . . . 17
13.2 Standard Fieldbus Interface . . . 17

14 Specific configuration for Beckhoff ADS interface **17**
14.1 Data types . . . 18
14.2 Configuration . . . 18

15 Changelog **20**

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`, which depends on the Mikado **system status**.

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)
		Only when starting the program or a sequence:
		`OutRunningBit = 1`

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` ***Nur when starting the program or a sequence:*** Wait for `OutRunningBit = 0`	
t_6	***Nur when starting the program or a sequence:*** Reading `OutErrorCode`	***Only when the program or a sequence was started before:*** `OutRunningBit = 0` (MIK program ends)
from here	read variables if necessary	

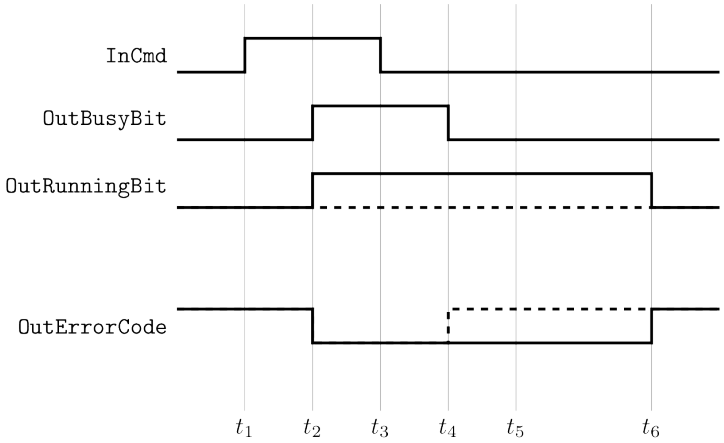

Fig. 1: General signal trace, alternative cases dashed

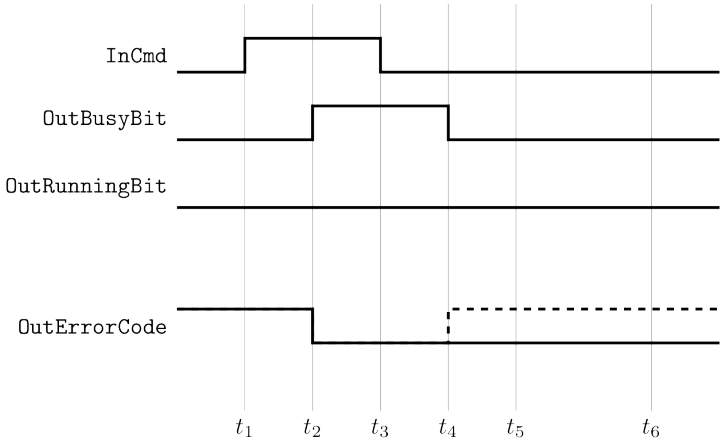

Fig. 2: Signal trace for commands like StartCalib, CaptureCalib, FinishCalib

- 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.
- Only commands `RunMikadoProg` and `StartSequence` cause `OutRunningBit = 1`. This signals that Mikado is processing the current program. As soon as `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, EtherCAT, EtherNet/IP, Profinet). The standard fieldbus configuration of newly created cells contains all the necessary entries for executing the commands.

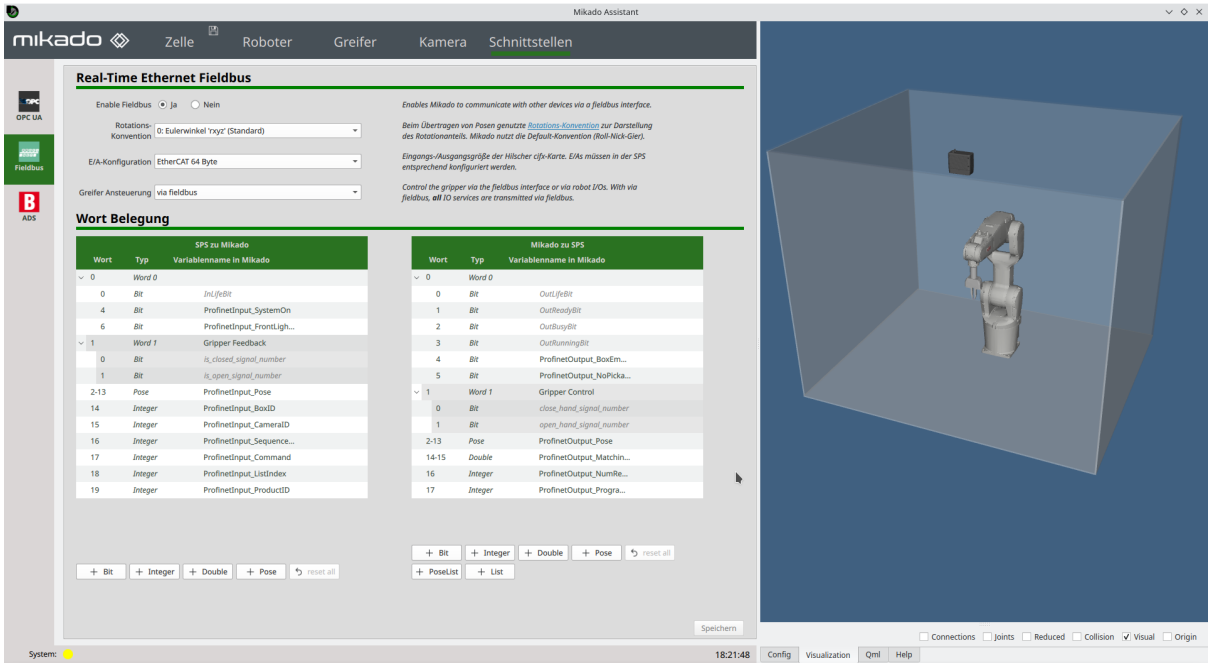

Fig. 3: Assistant for adapting the word assignment and synchronized Mikado variables via fieldbus

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

```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 / EtherCAT /␣
↪EtherNet/IP variables
      bit: <AddressProfinetSide>   # only applicable for Profinet / EtherCAT /␣
↪EtherNet/IP variables of type 'Bool'
      listSize: <size of list>     # only applicable for Profinet / EtherCAT /␣
↪EtherNet/IP variables of type 'PoseList or StringList'
      remoteName: <VariableNameAdsSide>  # only applicable for ads 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 conventions

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.

The rotation convention *uvw*, which is used natively by Universal Robots, has a special position. Instead of converting this to Euler angles on the robot side, it can be used directly by selecting the corresponding convention. The parameters **u**, **v** and **w** represent a vector around which the pose is rotated. The type of rotation description corresponds to the *Axis-Angle Representation*.

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"*
4	*uvw*	Universal Robots	*U*	*V*	*W*

6 Error codes

The output word `OutErrorCode` encodes error messages. The following table summarises all possible error codes and explains their meaning and recommended corrective actions.

Table 4: Possible error messages

Code	Error name	Description	Recommended action
0	`NoError`	Initial state. No error present	-
>0	Error in the Mikado program	Project-specific error in the Mikado program	For details, see the Mikado program or project-specific documentation
-1	`ServiceCallFailed`	Command failed due to an internal error.	Contact Mikado Support
-4	`NoReplyValue`	No return value received (internal error)	Contact Mikado Support
-5	`TooManyReplyValues`	Number of return values higher than expected (internal error)	Contact Mikado Support
-6	`FatalErrorMikadoProgram`	The Mikado program has a fatal error. The program cannot continue from this point.	Reset the Mikado program and restart the program / sequence
-8	`UnknownCommand`	Unknown command received	Check `InCmd`. For valid commands, see *Commands*.
-9	`LifebitClientMissing`	Lifebit of the client was not recognized.	Check the fieldbus connection between Mikado and the communication partner and the configuration of the 'LifeBit'
-11	`TriggeringFailed`	Failed to call a Mikado sequence. This can occur, for example, if Mikado is not ready for operation or if a program is already running.	Ensure that Mikado is ready for operation (`ReadyBit = true`) and that no program is already running (`RunningBit = false`)
-12	`ReadingVariableFailed`	Internal error when reading data from the fieldbus interface	Contact Mikado Support
-13	`WritingVariableFailed`	Internal error when reading data from the fieldbus interface	Contact Mikado Support

Table 5: Possible error messages, continued

Code	Error name	Description	Recommended action
-15	`CancelCalibrationFailed`	Aborting an already started calibration failed	Ensure that calibration has been started previously (*StartCalib*) and that no program is running (`RunningBit = false`)
-16	`CaptureForCalibrationFailed`	Command *CaptureCalib* failed. Capture for ongoing calibration unsuccessful	Ensure that calibration has been started previously (*StartCalib*) and that no program is running (`RunningBit = false`)
-17	`FinishCalibrationFailed`	Command *FinishCalib* failed. Calculating and saving calibration was not successful	Ensure that calibration has been started previously (*StartCalib*) and that no program is running (`RunningBit = false`)
-18	`StartCalibrationFailed`	Command *StartCalib* failed. Calibration could not be started	Ensure that no calibration is already in progress and no program is running (`RunningBit = false`)
-19	`SequenceDoesNotExist`	Command *StartSequence* failed. Mikado sequence with the specified ID does not exist	Check `SequenceID` and call command *StartSequence* with a valid ID
-20	`MissingConfigurationItem`	Invalid fieldbus configuration. A required entry is missing.	Contact Mikado Support
-60	`RetryErrorMikadoProgram`	The Mikado program has an error, but the execution can be retried. This can occur if the connection to the robot or camera is briefly interrupted while Mikado is attempting to execute an action.	Execute the command *RunMikadoProg*. In many cases, the cause of the error will have been automatically corrected by the time the retry is attempted, and the process can be continued successfully. If not, another error will be triggered.
-61	`RetryActionErrorMikadoProgram`	Robot movement failed. A movement command from Mikado could not be successfully executed by the robot. This can occur if the connection to the robot is lost or if the robot is stopped unexpectedly, for example due to an error or a safety stop (emergency stop).	If applicable, first rectify the cause of the unexpected stop and then execute the command *RunMikadoProg*. The program will continue. If the movement still cannot be executed successfully, an error will be triggered again.
-100	`UnexpectedError`	Unexpected error occurred	Contact Mikado Support

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 6: Available commands

`InCmd`	Command name	Description
0	`Idle`	Waiting
Program specific commands		
1	*RunMikadoProg*	Start/continue Mikado program from the current task
2	*StartSequence*	Start a specific sequence of the Mikado program. SequenceID must be set beforehand
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
14	*CancelCalib*	Cancel hand-eye calibration

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 7: 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 8: 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 9: 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 10: 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

CancelCalib

This command cancels the active hand-eye calibration.

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 11: 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 12: 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 of the fieldbus interface

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 (`RunningBit = 1`), unexpected behavior may occur with this version, as they are immediately visible to the program.

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 13: Example of PLC-side changes at an unfavorable time

Time	SPS	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.**

9.2 Version 3.1.2

Since *mikado-profinet* version 3.1.2 the package also supports the protocols EtherCAT and EtherNet/IP in addition to Profinet.

The protocol can be configured in Mikado Assistant on the fieldbus interface page.

10 Standard Fieldbus Interface

The different fieldbus interfaces have many commonalities. Following you will find the possible data types and the *Standard Word Mapping*.

This interface is fully supported for the protocols Profinet, EtherCAT and EtherNet/IP. Other protocols deviate from the standard as described in the respective section, like e.g Beckhoff ADS which maps variables to names instead of memory addresses.

An overview of the fieldbus interface documentation can be found *here*.

10.1 Byte Order

The byte order 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.

10.2 Fieldbus Data Types

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

- The word is broken down into 16 individual bits, whereby each bit is interpreted as a bool.
- The word is interpreted as an integer.
- `Double`: Two words (32 bits) are interpreted together as a floating point number.
- `Pose`: 12 consecutive words which are interpreted as six double values (X, Y, Z, R1, R2, R3).
- `PoseList` (Mikado to PLC only): `12 x <list length>` consecutive words per Pose, which are interpreted as a list of Poses. A list of length 5 will e.g. require 60 consecutive words.
- `StringList` (Mikado to PLC only): `2 x <list length>` consecutive words, which are interpreted as a list of double values. Strings are automatically converted to Double. A String that is not convertible to Double can not be transmitted using this interface.

10.3 Standard Word Assignment

Most of the assigned fieldbus words are directly mapped to Mikado variables.

Excluded from this are the words marked in grey in the Figure *Overview of the standard mapping of input and output byte arrays*, whose values are set independently of the Mikado program by the fieldbus adapter.

SPS zu Mikado

Word	Byte	Bit		
0	0	0	InLifeBit	
		1		
		2		
		3		
		4	InSystemOn	
		5		
		6	InFrontLightOn	
		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
	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	
		1	OutReadyBit	
		2	OutBusyBit	
		3	OutRunningBit	
		4	OutBoxEmptyBit	
		5	OutNoPickableObjectsBit	
		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
	35			
18	36			
	37			
19	38			
	39			
20	40			
	41			
21	42			
	43			
22	44			
	45			

Fig. 4: Overview of the standard mapping of input and output byte arrays

11 Specific configuration for Profinet

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

An overview of the fieldbus interface documentation can be found *here*.

11.1 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 depending on the chosen configuration. Inputs and outputs are defined from the perspective of the Profinet master:

- 64 byte input and output
- 128 byte input and output
- 256 byte input and output
- 512 byte input and output

These Inputs and Outputs are written and read according to the word mapping, see *Standard Word Assignment*.

11.2 Standard Fieldbus Interface

Byte order, available data types and the standard word mapping follow the *Standard Fieldbus Interface*.

12 Spezifische Konfiguration für EtherCAT

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

An overview of the fieldbus interface documentation can be found *here*.

12.1 Interface description

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

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

The following EtherCAT device is to be added on the master: `Hilscher CIFX RE ECS`

Depending on the selected configuration, the Mikado controller provides the following input/output layouts. Inputs and outputs are defined from the perspective of the EtherCAT master:

- 64 byte input and output
- 128 byte input and output
- 256 byte input and output

These inputs and outputs are written or read according to the configured word assignment, see *Standard Word Assignment*.

12.2 Standard Fieldbus Interface

Byte order, available data types and the standard word mapping follow the *Standard Fieldbus Interface*.

13 Spezifische Konfiguration für EtherNet/IP

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

An overview of the fieldbus interface documentation can be found *here*.

13.1 Interface description

The interface description for a EtherNet/IP connection is defined in an EDS file. After installing the Debian package `mikado-profinet`, this file can be found under the following path:

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

The following EtherNet/IP device is to be added on the master:

`Hilscher CIFX 50-RE EIS`

The Mikado controller provides the following input/output layout depending on the chosen configuration. Inputs and outputs are defined from the perspective of the EtherNet/IP master:

- 64 byte input and output
- 128 byte input and output
- 256 byte input and output

These Inputs and Outputs are written and read according to the word mapping, see *Standard Word Assignment*.

Warning: The Mikado EtherNet/IP interface is configured by default to assign an IP address via `DHCP`. This setting cannot currently be changed by the user. If you require a static IP address for the interface for your application, please contact Mikado Support.

13.2 Standard Fieldbus Interface

Byte order, available data types and the standard word mapping follow the *Standard Fieldbus Interface*.

14 Specific configuration for Beckhoff ADS interface

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

An overview of the fieldbus interface documentation can be found *here*.

14.1 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, 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.

14.2 Configuration

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

```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)

# 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
```

(continues on next page)

(continued from previous page)

```
# 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	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