optonic
vision · automation

# Fieldbus Interface

# mikado

Version 4.0

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, 2015-2026

# 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 . . . 5

**4 Synchronising fieldbus data and Mikado variables** **5**

**5 Rotation conventions** **6**

**6 Error codes** **7**

**7 Commands** **8**
7.1 Program specific commands . . . 9
7.2 General Commands . . . 11

**8 Example sequence for select cells** **14**

**9 Updates of the fieldbus interface** **15**
9.1 Version 3.3.1 . . . 15
9.2 Version 3.3.0 . . . 15
9.3 Version 3.1.2 . . . 16
9.4 Version 3.1.0 . . . 16

**10 Fieldbus interface in general** **17**
10.1 Byte Order . . . 17
10.2 Data Types . . . 17
10.3 Address assignment . . . 18

**11 Specific configuration for PROFINET** **19**
11.1 GSDML . . . 19
11.2 Standard Fieldbus Interface . . . 20

**12 Spezifische Konfiguration für EtherCAT** **20**
12.1 Interface description . . . 20
12.2 Standard Fieldbus Interface . . . 20

**13 Spezifische Konfiguration für EtherNet/IP** **20**
13.1 Interface description . . . 20
13.2 Standard Fieldbus Interface . . . 21

**14 Specific configuration for Beckhoff ADS interface** **21**
14.1 Data types . . . 21
14.2 Configuration . . . 21

**15 Changelog** **23**


# 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`<br>Wait for `OutBusyBit = 0`<br>Set parameters | Waiting on command |
| $t_1$ | Set `InCmd`<br>Wait for `OutBusyBit = 1` | |
| $t_2$ | | `OutErrorCode = 0`<br>`OutBusyBit = 1` (fieldbus adapter processes command)<br>***Only when starting the program or a sequence:***<br>`OutRunningBit = 1` |
| $t_3$ | Reset `InCmd` to `0`<br>Wait for `OutBusyBit = 0` | |
| $t_4$ | | Write `OutErrorCode`<br>`OutBusyBit = 0` (fieldbus adapter has processed command) |
| $t_5$ | Reading `OutErrorCode`<br>***Nur when starting the program or a sequence:***<br>Wait for `OutRunningBit = 0` | |
| $t_6$ | ***Nur when starting the program or a sequence:***<br>Reading `OutErrorCode` | ***Only when the program or a sequence was started before:***<br>`OutRunningBit = 0` (MIK program ends) |
| from here | read variables if necessary | |

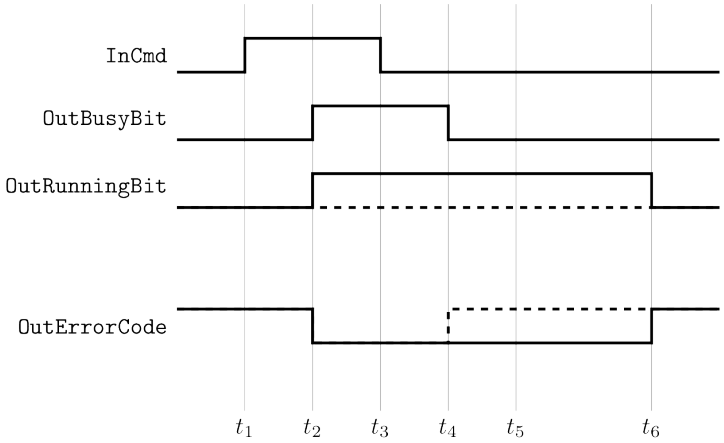


Fig. 1: General signal trace, alternative cases dashed

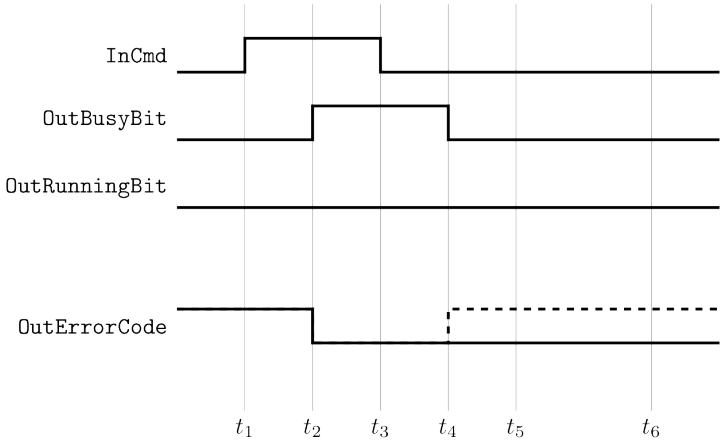


Fig. 2: Signal trace for commands like StartCalib, CaptureCalib, FinishCalib

- 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.<br>False = Adapter is ready to accept new command |
| `OutReadyBit` | System status of Mikado. | True = Mikado ready for operation<br>False = Mikado not ready |
| `OutRunningBit` | Program status of Mikado | True = Mikado program is running (corresponds to green or yellow in Mikado)<br>False = program is not running (corresponds in Mikado: orange, red or grey) |

**Note**

New commands are only accepted if `OutBusyBit` = False

# 4 Synchronising fieldbus data and Mikado variables

Fieldbus data can be mapped to Mikado variables and vice versa. This makes it possible to write data to the fieldbus or read data from it directly from within a Mikado program.

The mapping between fieldbus data and Mikado variables can be configured in the Mikado Assistant under 'Interfaces'. Dedicated configuration pages are available for the various supported communication protocols (ADS, EtherCAT, EtherNet/IP, Profinet).

In the *Address Assignment* table, the name of a Mikado variable can be specified in the "Variable name in Mikado" field for each element of the fieldbus interface. This automatically synchronises that element of the interface with the corresponding variable. The elements of the standard address assignment are named with the prefixes `FI` (short for 'fieldbus input') or `FO` (short for 'fieldbus output').

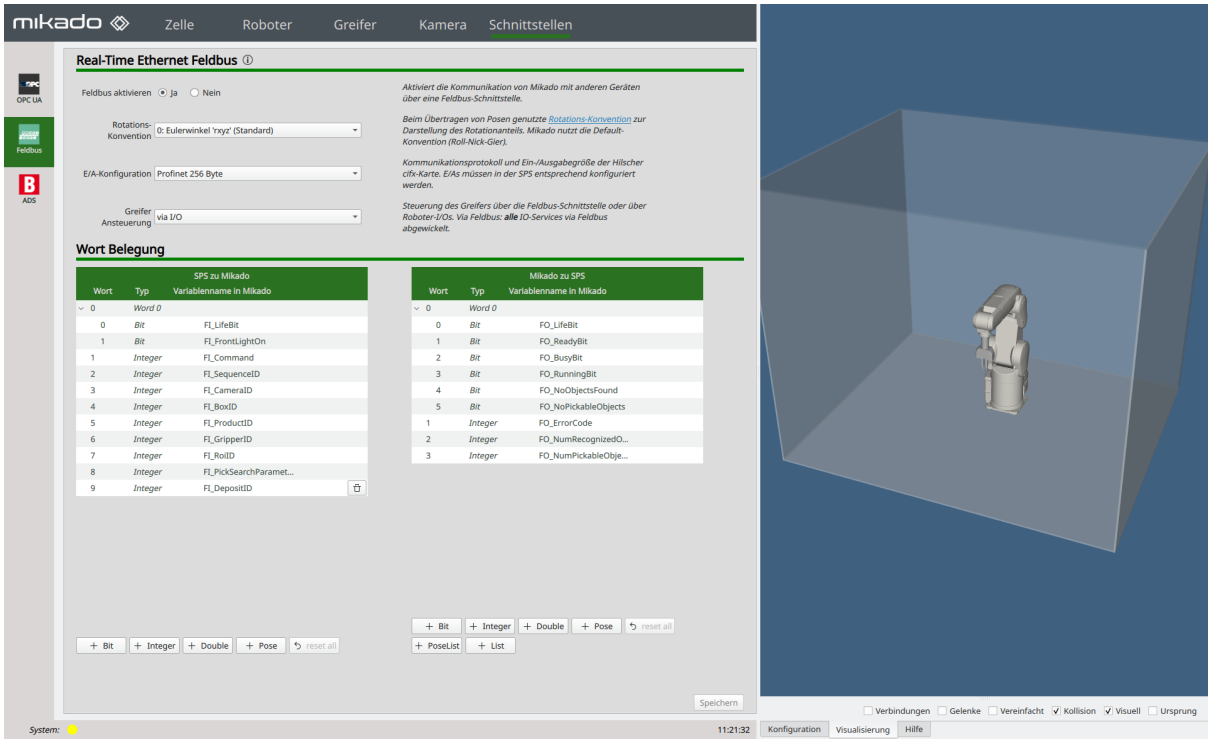


Fig. 3: Assistant for customising address mapping and synchronised variables

# 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)<br>Stäubli | *X* (Roll) | *Y'* (Nick/Pitch) | *Z"* (Gier/Yaw) |
| 1 | *sxyz* | Fanuc<br>Yaskawa Motoman<br>Universal Robots<br>Fruitcore<br>Mitsubishi<br>Techman | *X* (A) | *Y* (B) | *Z* (C) |
| 2 | *rzyx* | Kuka<br>ABB<br>Nachi | *Z* (A) | *Y'* (B) | *X"* (C) |
| 3 | *rzyz* | Doosan<br>Kawasaki<br>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 *Synchronising fieldbus data 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 |
| 20 | *Capture* | Triggers image acquisition |
| 21 | *StartPickSearch* | Start searching for objects and picks |
| 22 | *GetBestPick* | Reads the best available pick |

## 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 *Synchronising fieldbus data 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.

| Input values | Description |
|---|---|
| *none* | |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `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.

| Input values | Description |
|---|---|
| `InSequenceID` | ID of the sequence to be started |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `PauseMikadoProg`

This command pauses the execution of the current program.

| Input values | Description |
|---|---|
| *none* | |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `StopMikadoProg`

This command stops the execution of the current program.

| Input values | Description |
|---|---|
| *none* | |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `ResetMikadoProg`

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

| Input values | Description |
|---|---|
| *none* | |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

## 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.

| Input values | Description |
|---|---|
| `InCameraID` | ID of the camera to be used |
| `InFrontLightOn` | Specifies whether the front light is switched on during calibration |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `CaptureCalib`

This command is used to collect poses for hand-eye calibration. The calibration process must first have been started using the `StartCalib` command. At least 6 valid poses must be collected for calibration. Ideally, however, around 10 to 20 poses should be collected to achieve the best possible result. This command must therefore be run frequently.

| Input values | Description |
|---|---|
| `InPose` | Current Tool0 position (robot flange) |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

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`.

| Input values | Description |
|---|---|
| *none* | |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |
| `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.

| Input values | Description |
|---|---|
| *none* | |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `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.

| Input values | Description |
|---|---|
| `InPose` | Current Tool0 position (robot flange) |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `Capture`

**Note**

This command is generally only used for select cells.

This command triggers a capture for the selected camera.

| Input values | Description |
|---|---|
| `InCameraID` | ID of the camera to be used |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `StartPickSearch`

**Note**

This command is generally only used for select cells.

This command initiates a pick-search, i.e. a search for objects followed by picks. The most recent image captured by the selected camera is used for this purpose. It may therefore be necessary to trigger a capture beforehand using the *Capture* command.

Calling this command starts the search. However, the search itself runs asynchronously in the background. The command does not block until results are available!

| Input values | Description |
|---|---|
| `InCameraID` | ID of the camera to be used |
| `InBoxID` | ID of the camera to be used |
| `InProductID` | ID of the product to be used |
| `InGripperID` | ID of the gripper to be used |
| `InRoiID` | ID of the ROI (region of interest) to be used |
| `InPickSearchParametersID` | ID of the search parameters to be used |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |

### `GetBestPick`

**Note**

This command is generally only used for select cells.

This command retrieves the best available pick at the time it is called. To identify the correct search run, the parameters `InBoxID`, `InProductID` and `InGripperID` must be specified when calling the command. The term "best pick" refers to the criteria set for sorting picks.

The pick search benefits from having as long a time interval as possible between the start (*StartPickSearch*) and the retrieval of results (*GetBestPick*). Only then is there sufficient time to calculate all possible pick variations and prioritise them against one another, so that the best possible pick can be determined.

If the search has not yet finished when this command is called, the command will be blocked until either at least one valid pick has been found or all possible pick variations have been calculated, but no valid picks are found.

However, in order to produce a result quickly, the search may be terminated prematurely in this case, and the first possible pick will be returned. This will be any valid pick, but not necessarily the best theoretically possible pick.

For best results, *StartPickSearch* should therefore be called as early as possible in the cycle, whilst *GetBestPick* should be called as late as possible (just before the pick). This allows the pick search the maximum possible calculation time to find an ideal pick.

| Input values | Description |
|---|---|
| `InBoxID` | ID of the camera to be used |
| `InProductID` | ID of the product to be used |
| `InGripperID` | ID of the gripper to be used |

| Return values | Description |
|---|---|
| `OutErrorCode` | Error code |
| `OutNoObjectsFoundBit` | This bit indicates whether product objects were found. `true` if no objects are present, otherwise `false`. |
| `OutNoPickableObjectsBit` | This bit indicates whether pickable objects were found. `true` if no objects are present, otherwise `false`. |
| `OutNumRecognizedObjects` | Number of recognized product objects |
| `OutNumPickableObjects` | Number of pickable objects available at the time of the call |
| `OutPickGripperId` | ID of the gripper with which the selected pick is to be performed |
| `OutPickCategory` | Category of the pick |
| `OutPickSubcategory` | Subcategory of the pick |
| `OutPickStartPose` | Starting pose of the pick. Depending on the settings, the pose can be given in TCP or Tool0 coordinates (robot flange). |
| `OutPickApproachPose` | Approach pose of the pick. Depending on the settings, the pose can be given in TCP or Tool0 coordinates (robot flange). |
| `OutPickGraspPose` | Pick pose of the pick. Depending on the settings, the pose can be given in TCP or Tool0 coordinates (robot flange). |
| `OutPickRetractPose` | Retract pose of the pick. Depending on the settings, the pose can be given in TCP or Tool0 coordinates (robot flange). |
| `OutPickFinalPose` | Final pose of the pick. Depending on the settings, the pose can be given in TCP or Tool0 coordinates (robot flange). |

# 8 Example sequence for select cells

The following example illustrates how a typical process unfolds in Mikado Select.

Table 7: Example procedure for a select cell

| # | SPS | Mikado |
|---|---|---|
| *Capture* | | |
| 1 | Set `InCameraID`<br>Setze `InCmd = 20` (`Capture`) | |
| 2 | | Triggers image acquisition<br>End of the sequence: `OutBusyBit = 0` |
| *Searching for objects and picks* | | |
| 3 | Set `InCameraID`, `InBoxID`, `InProductID`, `InGripperID`, `InRoiID`, `InPickSearchParametersID`<br>Set `InCmd = 21` (`StartPickSearch`) | |
| 4 | | Start searching for objects and picks<br>End of the sequence: `OutBusyBit = 0` |
| *Read results* | | |
| 5 | Set `InBoxID`, `InProductID`, `InGripperID`<br>Set `InCmd = 22` (`GetBestPick`) | |
| 6 | | Reads the best available pick<br>Set return values; see *GetBestPick*<br>End of the sequence: `OutBusyBit = 0` |

# 9 Updates of the fieldbus interface

## 9.1 Version 3.3.1

Commands 20 (*Capture*), 21 (*StartPickSearch*) and 22 (*GetBestPick*) have been added. These commands allow pick searches to be initiated directly via the fieldbus interface and their results to be read. This means, for example, that Mikado Select cells can be operated without necessarily requiring a Mikado program sequence.

As part of this, the default address assingment for both Mikado-Control and Mikado-Select has also been revised.

## 9.2 Version 3.3.0

A PROFINET configuration that has been officially certified by Siemens is now being shipped. This requires the use of a new `Optonic` GSDML file. A slightly modified TIA module is available for communication with Siemens PLCs, which incorporates this new GSDML file.

Existing cells are not updated automatically. The changes are not backwards compatible, and adjustments are required on both the Mikado and PLC sides. If necessary, you can switch between the older and the current configuration in the Mikado Assistant (see figure below; you may need to use “advanced mode” to display this option). For new cells, the newer `Optonic` configuration is automatically preselected from version 3.3.0 of the Mikado Assistant onwards. It is strongly recommended to retain this selection. The obsolete `Hilscher` configuration remains selectable solely for reasons of backward compatibility.

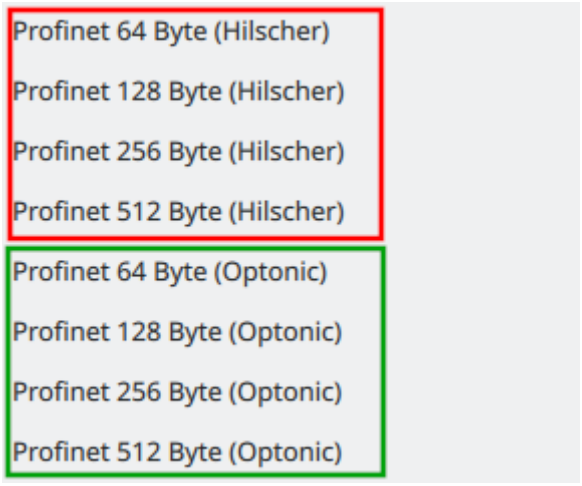


Fig. 4: I/O configuration for PROFINET: obsolete (red) and new (green) options

## 9.3 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.

## 9.4 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 8: Example of PLC-side changes at an unfavorable time

| Time | SPS | Mikado | Comment |
|---|---|---|---|
| $t_1$ | Set `InBoxID = 42`<br>Set `InCmd = 1` (start program) | | PLC sets the variable `InBoxID = 42` and sends start command |
| $t_2$ | | `OutBusyBit = 1`<br>`OutRunningBit = 1`<br>`OutBusyBit = 0` | Adapter executes command<br>Mikado starts program<br>Command execution completed |
| $t_3$ | `InBoxID = 0` (**false**)<br>`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 Fieldbus interface in general

Mikado's fieldbus interface communicates via a Hilscher cifX PCI card and operates in much the same way regardless of the protocol used. The following protocols are available:

- PROFINET
- EtherCAT
- EtherNet/IP

## 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 Data Types

Data is generally transmitted to and from the fieldbus interface in 2-byte packets (16 bits). The following data types are available:

| Data Type | Description |
|---|---|
| `Bit` | The 16 bits of a word are interpreted as 16 individual Boolean values |
| `Integer` | A word is interpreted as a signed integer |
| `Double` | Two words (4 bytes) are interpreted together as a floating-point number. For Siemens-PLCs, for example, this corresponds to the REAL data type |
| `Pose` | 12 consecutive words are interpreted as six double values of a pose in the following order: X, Y, Z, R1, R2, R3 |
| `PoseList` | (Mikado to PLC only): `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): s``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 Address assignment

The address assignment defines the assignment of data to the corresponding memory addresses. Both Mikado Control and Mikado Select have a default address assingment that includes all the data structures essential for operation (see below). These configurations can also be extended and modified as required, for example to transmit additional project-specific data.

Depending on the selected settings and the protocol used, up to 256 words (512 bytes) are available for each input and output. All ranges not explicitly listed in the default address allocation are unassigned by default and can be freely configured.

### Mikado Control

MIKADO INPUTS

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

MIKADO OUTPUTS

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

### Mikado Select

MIKADO INPUTS

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

MIKADO OUTPUTS

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

# 11 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-V2.43-OPTONIC-MIKADO-20250210.xml`

For reasons of backwards compatibility, an outdated GSDML file `GSDML-V2.35-HILSCHER-CIFX RE PNS-20190108.xml` is also available, see *Version 3.3.0*.

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 configured *address 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 *address 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 configured *address 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`:

```
# 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)

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

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

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

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

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

# 15 Changelog

| Version | 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 |