BUCHER
automation

Application-oriented manual



60888037_01

Mobile IO API

This document has been compiled by Bucher Automation AG with due diligence based on the state of the art as known to them. Any revisions and technical advancements of our products are not automatically made available in a revised document.
Bucher Automation AG shall not be liable for any errors either in form or content, or for any missing updates, as well as for any damage or detriment resulting from such failure.

Bucher Automation AG
Thomas-Alva-Edison-Ring 10
71672 Marbach/Neckar, Germany
T +49 7141 2550-0
info@bucherautomation.com

Technical support
T +49 7141 2550-444
support@bucherautomation.com

Sales
T +49 7141 2550-663
sales@bucherautomation.com

www.bucherautomation.com

Translation of the original German document

Document revision: 1.00.1
Date of issue: 4/7/2025

Table of contents

1 Scope of application 6
2 Requirements 7
3 Integrating the library into JetSym 8
4 Overview 10
5 Components 11
5.1 C_CanOpenMaster 11
5.1.1 Types in use 11
5.1.2 Constructor 11
5.1.3 NMT messages 11
5.1.4 Functions of the CAN master 12
5.2 Base node 13
5.2.1 Constructor 13
5.2.2 NMT messages of the nodes 13
5.2.4 System parameters 14
5.2.5 Saving the settings to non-volatile memory 14
5.2.6 System information 15
5.2.7 Error history 16
5.2.8 Operating system update 16
5.2.9 Deleting the mapping on the controller 17
5.3 JSCM-720-E04 18
5.3.1 I/Os 18
5.3.2 Saving application parameters 18
5.3.3 Diagnostic voltages 19
5.4 JXM-IO-E30 20
5.4.1 I/Os 20
5.5 JXM-IO-E31 20
5.5.1 I/Os 20
5.6 JXM-IO-E32 21
5.6.1 I/Os 21
6 Interfaces 22
6.1 Basics 22
6.1.1 SetPorttype 22
6.1.2 Input values 22
6.1.3 Output values 23
6.1.4 Parameter 24
6.1.5 Port status 25
6.1.6 Resetting the interface 25

6.2 C_AI_Current 26
6.2.1 Constructor 26
6.2.2 SetPorttype 26
6.2.3 Input and output values 26
6.2.4 Parameter 26
6.3 C_AI_Temperature 27
6.3.1 Constructor 27
6.3.2 SetPorttype 27
6.3.3 Input and output values 27
6.3.4 Parameter 27
6.4 C_AI_VoltageHighRange 28
6.4.1 Constructor 28
6.4.2 SetPorttype 28
6.4.3 Input and output values 28
6.4.4 Parameter 28
6.5 C_AI_VoltageRatio 29
6.5.1 Constructor 29
6.5.2 SetPorttype 29
6.5.3 Input and output values 29
6.5.4 Parameter 29
6.6 C_AO_PVG 30
6.6.1 Constructor 30
6.6.2 SetPorttype 30
6.6.3 Input and output values 30
6.6.4 Parameter 30
6.7 C_AO_Voltage 31
6.7.1 Constructor 31
6.7.2 SetPorttype 31
6.7.3 Input and output values 31
6.7.4 Parameter 31
6.8 C_DI 32
6.8.1 Constructor 32
6.8.2 SetPorttype 32
6.8.3 Input and output values 32
6.8.4 Parameter 32
6.9 C_FI 33
6.9.1 Constructor 33
6.9.2 SetPorttype 33
6.9.3 Input and output values 33
6.9.4 Parameter 33

6.10 C_DO 34
6.10.1 Constructor 34
6.10.2 SetPorttype 34
6.10.3 Input and output values 34
6.10.4 Parameter 34
6.11 C_DO_LS (digital low side) 35
6.11.1 Constructor 35
6.11.2 SetPorttype 35
6.11.3 Input and output values 35
6.11.4 Parameter 35
6.12 C_PWMO_FB (full bridge) 36
6.12.1 Constructor 36
6.12.2 SetPorttype 36
6.12.3 Input and output values 36
6.12.4 Parameter 36
6.13 C_PWMO_HB (half bridge) 37
6.13.1 Constructor 37
6.13.2 SetPorttype 37
6.13.3 Input and output values 37
6.13.4 Parameter 37
6.14 C_PWMO_HS (high side) 38
6.14.1 Constructor 38
6.14.2 SetPorttype 38
6.14.3 Input and output values 38
6.14.4 Parameter 38
6.15 C_CPWMO_HS (high side current control) 39
6.15.1 Constructor 39
6.15.2 SetPorttype 39
6.15.3 Input and output values 39
6.15.4 Parameter 39
7 Application 40
7.1 Constructor 40
7.2 SetPorttype 41
7.3 Mapping 42
7.4 Setoperational 42
7.5 Set/Get 43
8 Examples 44
8.1 Universal hydraulic function 44
8.2 Saving the parameters 47

1 Scope of application

The Mobile IO API is a library designed to be used in mobile automation applications. However, this does not rule out its use in industrial automation.
This application-oriented manual describes how to use the library version 1.00.0.00.
The purpose of the library is to address and configure Bucher Automation CANopen peripheral nodes (e.g. JSCM-720-E04, JXM-IO-E30, JXM-IO-E31 and JXM-IO-E32).
The library integrates into the JetSym programming environment and runs on an STX-compliant controller.

2 Requirements

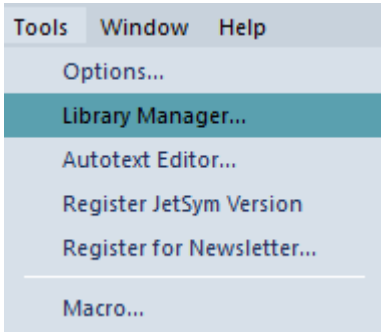
Using the library mandates the following requirements to be met:

- Basic understanding of object-oriented programming
- Previous experience in JetSym STX programming
- Presence of a JetSym programming environment (version 5.6 or higher)
- Presence of an STX-compliant controller (OS version 4.07 or higher or platform version 0x0109)

3 Integrating the library into JetSym

To integrate the library into JetSym, proceed as follows:

1. Go to Tools > Library Manager....



⇨ The Library Manager window opens.

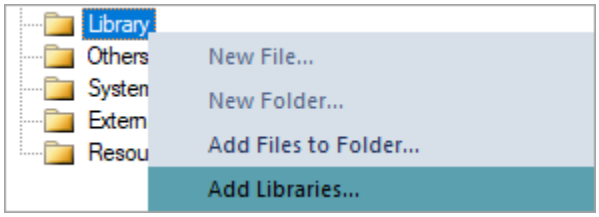


2. Click Add... to select the latest library version. If the latest version of the library is already present, skip this step.



⇨ The latest version of the library has been loaded and is ready to be included in your project.

3. In the file tree of your project, right-click Library and from the context menu select Add Libraries....



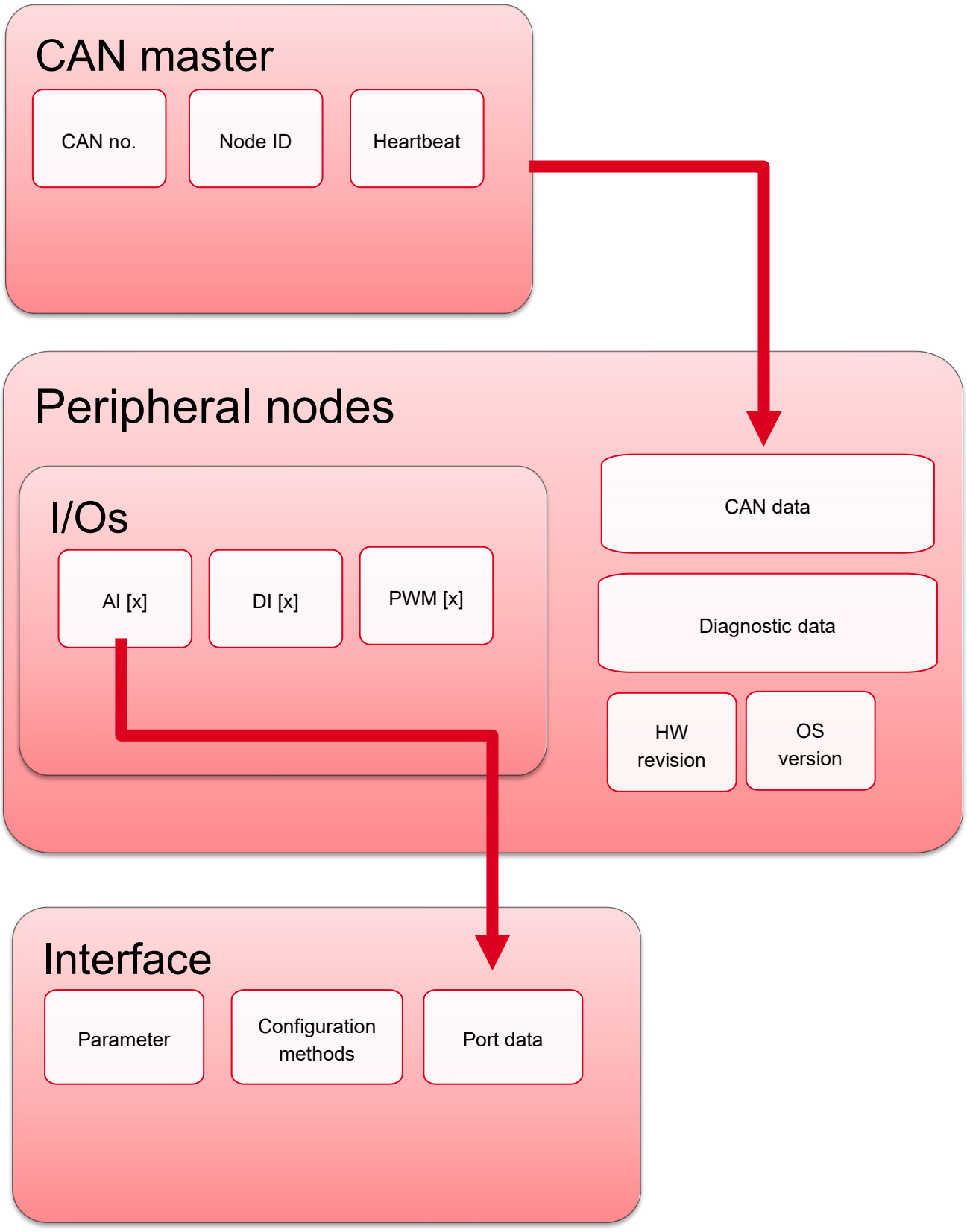
⇨ The Library Manager window opens.

4. Select the latest version from the list and click OK to confirm.



⇨ All classes and objects contained in the library are now available to be used in your project. No additional `#include` instruction is necessary.

4 Overview


CAN master
CAN no.
Node ID
Heartbeat
Peripheral nodes
I/Os
AI [x]
DI [x]
PWM [x]
CAN data
Diagnostic data
HW revision
OS version
Interface
Parameter
Configuration methods
Port data

5 Components

The following chapter describes the individual components of the Mobile IO API and how they are used.

5.1 C_CanOpenMaster

The `C_CanOpenMaster` class provides the basis for communication on the corresponding CANopen bus. It serves as central storage for key parametrization data of the CAN bus, including baud rate or CAN number. Each CAN interface of the controller requires a new instance of this class.

5.1.1 Types in use

To facilitate using this class, 2 enumerations have been defined.
The `T_CANOpenCmd` enumeration contains the states being used by the `SetCmdNMTall(cmd: T_CanOpenCmd)` function.

```
T_CanOpenCmd : enum(      START=1, STOP, PREOPERATIONAL=128);
```

The `T_CanOpenBaud` enumeration contains the baud rates that can be handed over in the constructor of the class.

```
T_CanOpenBaud: enum( Baud125K=125, Baud250K=250,Baud500K=500,Baud1M=1000);
```

5.1.2 Constructor

To call the class, use the following constructor:

```
public function C_CanOpenMaster(Baud: T_CanOpenBaud:=T_CanOpenBaud.Baud250K,
CANNo: int:=0, NodeID: int:=127, VersionString : string := '1.0');
```

This hands over all key parameters directly to the class.

Example of creating 2 CAN buses:

```
HMI_Can       : C_CanOpenMaster(T_CanOpenBaud.Baud250K,0,127);
IO_Can        : C_CanOpenMaster(T_CanOpenBaud.Baud500K,1,127);
```

5.1.3 NMT messages

If all nodes of the CAN bus are in either the Operational, Preoperational or Stopped state, the following function can be employed:

```
public function SetCmdNMTall(cmd: T_CanOpenCmd);
```

Example:

```
HMI_CAN.SetCmdNMTall(T_CanOpenCmd.Start);
```

This calls all HMI CAN components and sets the HMI CAN to Operational state.

5.1.4 Functions of the CAN master

This function changes the CAN state of only the controller running the code.
Calling this function initializes the interface prompting a heartbeat visible on the CAN bus:

```
public function init();
```

This function changes the state of the CAN interface itself thereby enabling or disabling the PDOs.

```
public function setOperational();
public function setPreoperational();
```

5.2 Base node

The purpose of the Mobile IO API is to address and easily connect the SPS with CANopen nodes.
To this end, the Mobile IO API uses and integrates the basic functions common to all CANopen nodes while allowing for a variety of combinations of node-specific features. The inputs and outputs of the nodes freely swap in order to connect to machines with varying configurations. This section details general features and special properties of the nodes.
The `C_MobilIO_Base_Node` class is inherited within the nodes described hereafter, but may also serve as a basis for CANopen-compatible components created by the user, such as a display class.

5.2.1 Constructor

Start by creating the node in the controller program:

```
MainNode      : C_JSCM_720(&IO_Can,0x20);
```

This generates a pointer to the CANopen master created previously. The pointer hands over all data including CAN number and semaphores.
It also provides the node ID required for all further communication.

5.2.2 NMT messages of the nodes

To change the CAN state of an individual node, use the following functions:

```
MainNode SetOperational();
MainNode SetPreOperational();
MainNode Restart();
```

To read the node state, use the following function:

```
MainNode HeartbeatState() :int;
```

Return value	Meaning
0	Bootup
4	Stopped
5	Operational
127	Preoperational
255	Offline (default value)

Each node is able to monitor the heartbeat of up to 4 CAN devices. The controller allows you to set the properties of the heartbeats to be monitored using the respective master node ID and timeout. If the device fails to detect a heartbeat within the set timeout (e.g. due to a disruption of communication), the state switches to Stopped and the outputs are de-energized.

```
public function addNewHeartbeatMonitoring(pNodeId :byte := -1,
                                          pHeartbeatTime :word := 2200) :int;
```

Return value	Meaning
0	OK
-1	SDO communication error while setting up the heartbeat monitoring
-2	SDO communication error while setting the heartbeat quantity
-3	Error: excessive number of heartbeats

5.2.4 System parameters

Use the system parameters to read and write the individual node parameters.
To request the CRC value, use the following function:

```
MainNode.SystemParameter_CRC.get();
```

The CRC checks whether the settings need to be transferred to the device again.
To set the node's baud rate, use the following function:

```
MainNode.SystemParameter_Baudrate.set(value);
```

Value	Baud rate
0	125 kBaud
1	250 kBaud
2	500 kBaud
3	1000 kBaud

To set the node ID, use the following function:

```
MainNodeSystemParameter_NodeID.set(value);
```

To read the config pin values, use the following function:

```
MainNodeSystemParameter_NodeID_Offset.get();
```

The node ID defines where the node can be reached after system restart; it is the sum of the node ID parameters and the node ID offset.
To set the interval between 2 heartbeats of the node, use the following function:

```
public Heartbeat_time      : C_Heartbeat_time_setting;
```

5.2.5 Saving the settings to non-volatile memory

To save the node settings (e.g. mappings or port types) permanently, use the following function:

```
public function Settings_IO_store();
```

The time required to save the settings differs for each node and may take as long as 20 s. With the JSCM-720-E04 device, the return value of the following function tells you whether or not the saving process was completed successfully:

```
Settings_IO_Store_cl.get();
```

Return value	Meaning
1	OK
-1	SDO communication error
0	Error while writing the settings

To re-apply the default settings, use the following function:

```
public function Settings_IO_reset();
Settings_IO_Reset_cl.get();
```

Return value	Meaning
1	OK
0	Error while writing the settings
-1	SDO communication error

System restart required to apply the changes.

Example:

```
if StoredCRC != MainNode.SystemParameter_CRC.get() then
       MainNode.Settings_IO_store();
       when MainNode.Settings_IO_Store_cl.get() = 1 then
              //saving successfully completed
              delay(100);
              StoredCRC := MainNode.SystemParameter_CRC.get()
       else_time T#20sec then
              //saving failed
       end_when;
end_if;
```

Please refer to chapter **Fehler! Verweisquelle konnte nicht gefunden werden.** for a detailed example.

5.2.6 System information

The following classes contain the `.get()` function. Use the `.get()` function to read system information:

```
public SW_Version   : C_SW_Version;
public HW_Version   : C_HW_Version;
public DeviceName   : C_DeviceName
```

To read overall system errors, use the `.get()` function of the following class:

```
public Status_Information  : C_Status_information;
```

Return value	Meaning
0	OK
-1	SDO communication error
Bit	**Description**
0	General error
1	Total overcurrent
3	Temperature
4	Communication error
7	I/O error

5.2.7 Error history

The software comes with an error memory displaying the latest 256 errors logged. The following functions are available:

```
MainNode.ErrorHistory_numbers.get();
MainNode.ErrorHistory_lastEntry.get();
MainNode.ErrorHistory_Entrys.get(number);
```

5.2.8 Operating system update

The OS file must be present in the `.os` or `.hex` format. Execute the following functions to perform the OS update of the node:

```
public function OSUpdate(OSFilePath :string,blocking :bool := true) : int;
```

`OSFilePath` hands the path for the new OS over to the function. Use `blocking` to define whether the controller must wait for the update to complete (the update process will be aborted after 250 s). If set to false, it must be ensured that the update was completed successfully prior to applying the SDO communication to the bus node.

Return value	Meaning
0	OK
-1	Timeout (target does not respond within set time period)
-2	Invalid parameter, e.g.: Download status is not a global variable
-10	Incorrect file size
-11	Invalid file name
-12	There has been a previous download to the bus node
-13	Node ID of the given device
-14	Invalid product ID (OS does not match target node ID)
-15	Failed to write to target flash
-16	Communication error occurred during transfer
-17	Failed to reach target node (offline)
-20	User aborted the process
-30	Target node does not support update
-40	Update process error

```
public function OSUpdate_Downloadstatus_Get():int;
```

This allows for the OS update state to be read back (0 ... 1,000 ‰)
After the download was completed successfully, the node must be restarted.

5.2.9 Deleting the mapping on the controller

Prior to re-mapping the parameters of the node, clear the existing mapped values of this node.
To do so, the following function is available:

```
DeleteMappingONControl() :int;
```

To clear the complete configuration stored on the controller, execute the `ResetInterface();` function for each interface individually (see chapter 6.1.6).

5.3 JSCM-720-E04

This section details the elements distinctive of the JSCM-720-E04 device.

5.3.1 I/Os

The JSCM-720-E04 features the following inputs and outputs:

```
DI_P          : array[1..16]
DI            : array[1..12]
MFI_P         : array[1..8]
MFI_P_Rat     : array[1..4]
PWM_H3        : array[1..38]
PWMi_H3       : array[1..16]
PWM_HL8       : array[1..8]
AO            : array[1..2]
AO_PVG        : array[1..4]
```

Pointers allow for the inputs and outputs to be used as parameters with varying interfaces.
For a detailed description of the inputs and outputs, please refer to the product-specific documentation.

5.3.2 Saving application parameters

The software provides non-volatile memory for 1,020 U32-type application parameters.
To read the application parameters, call:

```
MainNode.Application_Settings_Storage1.get(valueNumber:int);
```

To write the application parameters, call:

```
MainNode.Application_Settings_Storage1.set(valueNumber:int,value:int);
```

where the `valueNumber` parameter can be any value between: 1 … 255:

```
public Application_Settings_Storage1    : C_Application_Settings_Store;
public Application_Settings_Storage2    : C_Application_Settings_Store;
public Application_Settings_Storage3    : C_Application_Settings_Store;
public Application_Settings_Storage4    : C_Application_Settings_Store;
```

Additionally, it is possible to read and write the parameters using a collective function,
where the `valueNumber` parameter can be any value between: 1 … 1020.

```
MainNode.Application_Settings.set(valueNumber:int,value:int);
```

Return value	Meaning
0	OK
-1	SDO communication error
-2	Incorrect interface configuration
-3	Parameters outside the permissible range

```
MainNode.Application_Settings.get(valueNumber:int);
```

Return value	Meaning
≥ 0	Parameter contents
-1	SDO communication error
-2	Incorrect interface configuration
-3	Parameters outside the permissible range

5.3.3 Diagnostic voltages

The software comes with the following diagnostic voltages:

```
DiagnosisVoltage.UB_ECU .get()
DiagnosisVoltage.VBAT_PWR_A.get()
DiagnosisVoltage.VBAT_PWR_B.get()
DiagnosisVoltage.VBAT_PWR_C.get()
DiagnosisVoltage.VEXT_SEN_1.get()
DiagnosisVoltage.VEXT_SEN_2.get()
DiagnosisVoltage.VEXT_SEN_3.get()
DiagnosisVoltage.VEXT_SEN_4.get()
DiagnosisVoltage.VEXT_SEN10_1.get()
DiagnosisVoltage.VEXT_SEN10_2.get()
```

5.4 JXM-IO-E30

This section details the elements distinctive of the JXM-IO-E30 device.

5.4.1 I/Os

The JXM-IO-E30 features the following inputs and outputs:

```
AI            : array[1..8]
DI_P          : array[1..4]
PWMi_H3       : array[1..4]
PWM_H7        : array[1..6]
DO_H3         : array[1..4]
```

Pointers allow for the inputs and outputs to be used as parameters with varying interfaces.
For a detailed description of the inputs and outputs, please refer to the product-specific documentation.

5.5 JXM-IO-E31

This section details the elements distinctive of the JXM-IO-E31 device.

5.5.1 I/Os

The JXM-IO-E31 features the following inputs and outputs:

```
AI            : array[1..6]
DI            : array[1..8]
PWMi_H3       : array[1..4]
PWMi_HL5      : array[1..4]
PWMi_HL12     : array[1..4]
AI_R          : array[1..1]
```

Pointers allow for the inputs and outputs to be used as parameters with varying interfaces.
For a detailed description of the inputs and outputs, please refer to the product-specific documentation.

5.6 JXM-IO-E32

This section details the elements distinctive of the JXM-IO-E32 device.

5.6.1 I/Os

The JXM-IO-E32 features the following inputs and outputs:

```
AI           : array[1..8]
DI           : array[1..6]
AI_prec      : array[1..2]
AO           : array[1..3]
```

Pointers allow for the inputs and outputs to be used as parameters with varying interfaces.
For a detailed description of the inputs and outputs, please refer to the product-specific documentation.

6 Interfaces

Within the project, instances of the individual interface classes are used to address the ports. The interface classes contain all parameters supported by this port along with the requisite functions.

6.1 Basics

There are different interface members and methods:

1. Constructor providing different parameters to hand over basic parameters (e.g. hardware ports).
2. `SetPorttype()` to write the port configuration to the node.
3. Input and output values of the interface to be mapped to the PDOs.
4. Parameters of the interface.
5. `ResetInterface()`, to clear all parameters of the interface.

6.1.1 SetPorttype

Use this function to configure the individual ports.
The constructor stores all parameters required for configuration.

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.1.2 Input values

The `I_` prefix marks an input value.
To read the input value contents, use the following function:

```
public function get():int;
```

Return value	Meaning
≥ 0	SDO contents
-1	SDO communication error
-2	Incorrect interface configuration

Additionally, the `map` function enables PDO mapping on both, the node and the controller, with the API automatically computing the mapping position.

```
function map(EventTime :int := 1000,
             // cycle time for receiving a telegram

                InhibitTime :int := 200,
             // minimum interval between two received telegrams

                alreadyStoredOnNode :bool := false
             // during saving: PDO is generated only on the controller);
```

Return value	Meaning
0	OK
-1	SDO communication error
-2	Incorrect interface configuration
-3	SDO – set PDO to invalid – error
-4	SDO – clear mapping counter – error
-5	Mapping: SDO – set mapping entry – error
-6	SDO – set mapping counter – error
-7	SDO – set Inhibit time – error
-8	SDO – set event time – error
-9	Set message ID – error
-10	PDO – on controller – error
-11	Excessive number of values mapped.

6.1.3 Output values

The `O_` prefix marks an output value.
To read the output value contents, use the following function:

```
public function set():int;
```

Return value	Meaning
0	OK
-1	SDO communication error
-2	Incorrect interface configuration

To read the output value contents back, use the following function:

```
public function get():int;
```

Return value	Meaning
≥ 0	SDO contents
-1	SDO communication error
-2	Incorrect interface configuration

Additionally, the `map` function enables PDO mapping on both, the node and the controller, with the API automatically computing the mapping position.

```
function map(EventTime :int := 1000,
             // cycle time for receiving a telegram

                InhibitTime :int := 200,
             // minimum interval between two received telegrams

                alreadyStoredOnNode :bool := false
             // during saving: PDO is generated only on the controller);
```

Return value	Meaning
0	OK
-1	SDO communication error
-2	Incorrect interface configuration
-3	SDO – set PDO to invalid – error
-4	SDO – clear mapping counter – error
-5	Mapping: SDO – set mapping entry – error
-6	SDO – set mapping counter – error
-7	SDO – set Inhibit time – error
-8	SDO – set event time – error
-9	Set message ID – error
-10	PDO – on controller – error
-11	Excessive number of values mapped.

6.1.4 Parameter

The `IO_` prefix marks a parameter.
To set the parameter contents, use the following function:

```
public function set():int;
```

Return value	Meaning
0	OK
-1	SDO communication error
-2	Incorrect interface configuration

To read the parameter contents back, use the following function:

```
public function get():int;
```

Return value	Meaning
≥ 0	SDO contents
-1	SDO communication error
-2	Incorrect interface configuration

6.1.5 Port status

While the port status parameter owns a set of functions identical to a normal parameter, it returns the value type `T_Status` rather than `int`.

```
T_Status: Bits(
       SHORTCIRCUIT  = 0,
       OPENCIRCUIT   = 1,
       OVERVOLTAGE   = 2,
       OVERCURRENT   = 3,
       UPDATETIMEOUT = 4,
       DEFECT        = 5,
       SAFESTATE     = 6,
       ESTOP         = 7,
       SUPPLYFAULT   = 8,
       SIGNAL_MISMATCH = 9,
       ERROR         = 10,
       CC_NOTLOCKED  = 11,
       TEMPERATUREFAULT = 12
       );
```

6.1.6 Resetting the interface

The following function can be used to clear all parameters of the interface:

```
ResetInterface()
```

It allows for rerunning the parametrization without requiring a restart of the controller. It has no effect on the bus node. To clear the parameters from the node, either restart the node, or, where the parametrization is stored, execute the `Settings_IO_reset()` function.

6.2 C_AI_Current

6.2.1 Constructor

```
public function C_AI_Current(InterfacePort :T_P_InterfacePort:=NULL
                                            // hardware interface
                             );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.2.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.2.3 Input and output values

```
public I_Current    : C_I_Current_ua;
```

6.2.4 Parameter

```
public PortStatus               : C_PortStatus;
public IO_AI_Sample_Time_ms     : C_IO_AI_Sample_Time_ms;
public IO_AI_Average_Time_ms    : C_IO_AI_Average_Time_ms;
public IO_Supply                : C_IO_Supply;
public IO_Min_Deviation         : C_IO_MIN_Deviation;
```

6.3 C_AI_Temperature

6.3.1 Constructor

```
public function C_AI_Temperature (InterfacePort :T_P_InterfacePort:=NULL
                                                    // hardware interface
                                  );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.3.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.3.3 Input and output values

```
public I_Temperature       : C_I_Temperature;
```

6.3.4 Parameter

```
public       PortStatus                : C_PortStatus;
public       IO_MAX_Temperature        : C_IO_MAX_TEMPERATURE;
public       IO_MIN_Temperature        : C_IO_MIN_TEMPERATURE;
public       IO_Min_Deviation          : C_IO_MIN_Deviation;
```

6.4 C_AI_VoltageHighRange

6.4.1 Constructor

```
public function C_AI_VoltageHighRange (InterfacePort :T_P_InterfacePort:=NULL
                                                      // hardware interface
                                       );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.4.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.4.3 Input and output values

```
public I_Voltage     : C_I_Voltage_mv;
```

6.4.4 Parameter

```
public PortStatus                : C_PortStatus;
public IO_AI_Sample_Time_ms      : C_IO_AI_Sample_Time_ms;
public IO_AI_Average_Time_ms     : C_IO_AI_Average_Time_ms;
public IO_Supply                 : C_IO_Supply;
public IO_Min_Deviation          : C_IO_MIN_Deviation;
```

6.5 C_AI_VoltageRatio

6.5.1 Constructor

```
public function C_AI_VoltageRatio (InterfacePort :T_P_InterfacePort:=NULL
                                                  // hardware interface
                                   );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.5.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.5.3 Input and output values

```
public I_VoltagePromille          : C_I_Voltage_promille;
```

6.5.4 Parameter

```
public PortStatus                 : C_PortStatus;
public IO_AI_Sample_Time_ms       : C_IO_AI_Sample_Time_ms;
public IO_AI_Average_Time_ms      : C_IO_AI_Average_Time_ms;
public IO_Supply                  : C_IO_Supply;
public IO_Min_Deviation           : C_IO_MIN_Deviation;
```

6.6 C_AO_PVG

6.6.1 Constructor

```
public function C_AO_PVG (InterfacePort :T_P_InterfacePort:=NULL // hardware interface
                          );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.6.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.6.3 Input and output values

```
public I_Voltage           : C_I_Voltage_mv;
public O_Voltage_Promille  : C_O_Voltage_promille;
```

6.6.4 Parameter

```
public IO_Supply           : C_IO_Supply;
public PortStatus          : C_PortStatus;
public IO_Min_Deviation    :C_IO_MIN_Deviation
```

6.7 C_AO_Voltage

6.7.1 Constructor

```
public function C_AO_Voltage (InterfacePort :T_P_InterfacePort:=NULL
                                                  // hardware interface
                                 );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.7.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.7.3 Input and output values

```
public O_VOLTAGE    : C_O_Voltage_mv;
```

6.7.4 Parameter

```
public IO_Supply          : C_IO_Supply;
public PortStatus         : C_PortStatus;
public IO_Min_Deviation   : C_IO_MIN_Deviation
```

6.8 C_DI

6.8.1 Constructor

```
public function C_DI(InterfacePort :T_P_InterfacePort:=NULL,       // hardware interface
                     NPN :bool := false                // NPN - PNP (default PNP)
                     );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use
-3	NPN/PNP differs from other configurations in the same block

6.8.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.8.3 Input and output values

```
public I_Logic_Level       : C_I_Logic_Level;
```

6.8.4 Parameter

```
public IO_Supply           : C_IO_Supply;
public PortStatus          : C_PortStatus;
public IO_Min_Deviation    :C_IO_MIN_Deviation
```

6.9 C_FI

6.9.1 Constructor

```
public function C_FI(InterfacePort :T_P_InterfacePort:=NULL,       // hardware interface
                     NPN :bool := false                // NPN - PNP (default PNP)
                     );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use
-3	NPN/PNP differs from other configurations in the same block

6.9.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.9.3 Input and output values

```
public I_Logic_Level       : C_I_Logic_Level;
public I_Frequency         : C_I_Frequency_hz;
public I_Dutycycle         : C_I_Duty_cycle_promill;
public I_Counter           : C_I_Counter;
public I_LowImpuls_us      : C_I_Low_Impuls_us;
public I_HighImpuls_us     : C_I_High_Impuls_us;
public I_Period_us         : C_I_Period_us;
```

6.9.4 Parameter

```
public IO_Timeout_ms       : C_IO_TI_Timeout_ms;
public IO_GateTime_ms      : C_IO_FI_Gate_Time_ms;
public IO_Supply           : C_IO_Supply;
public PortStatus          : C_PortStatus;
public IO_Min_Deviation    :C_IO_MIN_Deviation
```

6.10 C_DO

6.10.1 Constructor

```
public function C_DO(InterfacePort :T_P_InterfacePort:=NULL,       // hardware interface
                     MaxCurrentmA :int := -1                       // maximum current
                    );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.10.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.10.3 Input and output values

```
public O_Logic_Level             : C_O_Logic_Level;
public I_Current                 : C_I_Current_ma;
```

6.10.4 Parameter

```
public IO_Current_Limit          : C_IO_Current_Limit_ma;
public IO_OverCurrent_timeout    : C_IO_Over_Current_Timeout_ms;
public IO_OpenCircuit_detection  : C_IO_Open_Circuit_Detection;
public IO_Supply                 : C_IO_Supply;
public IO_LoadType               : C_IO_Load_Type;
public PortStatus                : C_PortStatus;
public IO_Min_Deviation          : C_IO_MIN_Deviation
```

6.11 C_DO_LS (digital low side)

6.11.1 Constructor

```
public function C_DO_LS (InterfacePort :T_P_InterfacePort:=NULL,   // hardware interface
                         MaxCurrentmA :int := -1
                         // maximum current (if -1, the default value is used)
                         parallel :bool := false          //parallel connection 1||2,
                    3||4, 5||6
                         );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.11.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.11.3 Input and output values

```
public O_Logic_Level             : C_O_Logic_Level;
public I_Current                 : C_I_Current_ma;
```

6.11.4 Parameter

```
public IO_Current_Limit          : C_IO_Current_Limit_ma;
public IO_OverCurrent_timeout    : C_IO_Over_Current_Timeout_ms;
public IO_OpenCircuit_detection  : C_IO_Open_Circuit_Detection;
public IO_Supply                 : C_IO_Supply;
public IO_LoadType               : C_IO_Load_Type;
public PortStatus                : C_PortStatus;
public IO_Min_Deviation          : C_IO_MIN_Deviation
```

6.12 C_PWMO_FB (full bridge)

6.12.1 Constructor

```
public function C_PWMO_FB (InterfacePort :T_P_InterfacePort:=NULL, // hardware interface
                           MaxCurrentmA :int := -1
                           // maximum current (if -1, the default value is used)
                           parallel :bool := false          //parallel connection 1||2,
                     3||4, 5||6
                           );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use
-3	Parallel connection no enabled on port

6.12.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.12.3 Input and output values

```
public O_Dutycycle         : C_O_Dutycycle_promille;
public I_Current    :              C_I_Current_ma;
```

6.12.4 Parameter

```
public IO_Frequency        : C_IO_PWM_Freq_hz ;
public IO_Current_Limit           : C_IO_Current_Limit_ma;
public IO_OverCurrent_timeout     : C_IO_Over_Current_Timeout_ms;
public IO_OpenCircuit_detection   : C_IO_Open_Circuit_Detection;
public IO_Dither_Frequ            : C_IO_Dither_freq;
public IO_Dither_Ampl             : C_IO_Dither_Ampl_percent;
public IO_Supply           : C_IO_Supply;
public IO_LoadType         : C_IO_Load_Type;
public PortStatus          : C_PortStatus;
public IO_Min_Deviation    :C_IO_MIN_Deviation
```

6.13 C_PWMO_HB (half bridge)

6.13.1 Constructor

```
public function C_PWMO_HB (InterfacePort :T_P_InterfacePort:=NULL, // hardware interface
                           MaxCurrentmA :int := -1
                           // maximum current (if -1, the default value is used)
                           parallel :bool := false          // parallel connection 1||2,
                     3||4, 5||6
                           );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use
-3	Parallel connection no enabled on port

6.13.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.13.3 Input and output values

```
public O_Dutycycle  : C_O_Dutycycle_promille;
public I_Current    : C_I_Current_ma;
```

6.13.4 Parameter

```
public IO_Frequency              : C_IO_PWM_Freq_hz ;
public IO_Current_Limit          : C_IO_Current_Limit_ma;
public IO_OverCurrent_timeout    : C_IO_Over_Current_Timeout_ms;
public IO_OpenCircuit_detection  : C_IO_Open_Circuit_Detection;
public IO_Dither_Frequ           : C_IO_Dither_freq;
public IO_Dither_Ampl            : C_IO_Dither_Ampl_percent;
public IO_Supply                 : C_IO_Supply;
public IO_LoadType               : C_IO_Load_Type;
public PortStatus                : C_PortStatus;
public IO_Min_Deviation          :C_IO_MIN_Deviation
```

6.14 C_PWMO_HS (high side)

6.14.1 Constructor

```
public function C_PWMO_HS (InterfacePort :T_P_InterfacePort:=NULL, // hardware interface
                           MaxCurrentmA :int := -1
                           // maximum current (if -1, the default value is used)
                           parallel :bool := false          //parallel connection 1||2,
                     3||4, 5||6
                           );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use
-3	Parallel connection no enabled on port

6.14.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.14.3 Input and output values

```
public O_Dutycycle         : C_O_Dutycycle_promille;
public I_Current           : C_I_Current_ma;
```

6.14.4 Parameter

```
public IO_Frequency              : C_IO_PWM_Freq_hz ;
public IO_Current_Limit          : C_IO_Current_Limit_ma;
public IO_OverCurrent_timeout    : C_IO_Over_Current_Timeout_ms;
public IO_OpenCircuit_detection  : C_IO_Open_Circuit_Detection;
public IO_Dither_Frequ           : C_IO_Dither_freq;
public IO_Dither_Ampl            : C_IO_Dither_Ampl_percent;
public IO_Supply                 : C_IO_Supply;
public IO_LoadType               : C_IO_Load_Type;
public PortStatus                : C_PortStatus;
public IO_Min_Deviation          : C_IO_MIN_Deviation
```

6.15 C_CPWMO_HS (high side current control)

6.15.1 Constructor

```
public function C_CPWMO_HS (InterfacePort :T_P_InterfacePort:=NULL,
                                  // hardware interface
                            MaxCurrentmA :int := -1
                            // maximum current (if -1, the default value is used)
                            );
```

Return value	Meaning
0	OK
-1	Port does not support interface
-2	Port is already in use

6.15.2 SetPorttype

```
public function SetPorttype();
```

Return value	Meaning
0	OK
-1	SDO communication error
-5	Incorrect interface configuration

6.15.3 Input and output values

```
public O_Current_mA        : C_O_Current_ma;
public I_Current           : C_I_Current_ma;
```

6.15.4 Parameter

```
public IO_Frequency              : C_IO_PWM_Freq_hz ;
public IO_Current_Limit          : C_IO_Current_Limit_ma;
public IO_OverCurrent_timeout    : C_IO_Over_Current_Timeout_ms;
public IO_OpenCircuit_detection  : C_IO_Open_Circuit_Detection;
public IO_Dither_Frequ           : C_IO_Dither_freq;
public IO_Dither_Ampl            : C_IO_Dither_Ampl_percent;
public IO_CCO_P_Term             : C_IO_CCO_P_Term;
public IO_CCO_I_Term             : C_IO_CCO_I_Term;
public IO_CCO_D_Term             : C_IO_CCO_D_Term;
public IO_CCO_Cycle_Interval     : C_IO_CCO_Cycle_Interval;
public IO_Supply                 : C_IO_Supply;
public IO_LoadType               : C_IO_Load_Type;
public PortStatus                : C_PortStatus;
public IO_Min_Deviation          : C_IO_MIN_Deviation
```

7 Application

This chapter describes how to use the Mobile IO API. Perform the following steps in the given order and with reference to the port being addressed.
The names of the individual components (e.g. `MainNode`) are given by way of example and need to be adjusted to the use case as required.

7.1 Constructor

There are 2 options for calling the relevant constructor:
Together with the variable definition:

```
var
       IO_Can              : C_CanOpenMaster(T_CanOpenBaud.Baud500K,1,127);
       MainNode            : C_JSCM_720(&IO_Can,0x20);
       Lights_Back_Node    : C_JXM_IO_E30(&IO_Can,0x30);

       BrakeLight_Left           : C_DO (&Light_Back_Node.DO_H3[1],2500);
       TurnSignal_left_back      : C_DO(&Light_Back_Node.DO_H3[2]);

       PumpSpeed_Sensor    : C_FI(&MainNode.DI_P[10]);
       PumpOutput          : C_HB(&MainNode.PWM_HL8[1]);

End_var;
```

Later in the code sequence, which delivers the same content.

```
var

              IO_Can              : C_CanOpenMaster;
              MainNode            : C_JSCM_720(&IO_Can,0x20);
              Lights_Back_Node    : C_JXM_IO_E30(&IO_Can,0x30);

              BrakeLight_Left           : C_DO;
              TurnSignal_left_back      : C_DO;

              PumpSpeed_Sensor    : C_FI;
              PumpOutput          : C_PWMO_HB;

       end_var
```

```
task mainTask autorun


      IO_Can.C_CanOpenMaster(T_CanOpenBaud.Baud500K,2,127);
      MainNode.C_JSCM_720(&IO_Can,0x10);
      Lights_Back_Node.C_JXM_IO_E30(&IO_Can,0x30);

      BrakeLight_Left.C_DO (&Lights_Back_Node.DO_H3[1],2500);
      TurnSignal_left_back.C_DO(&Lights_Back_Node.DO_H3[2]);

      PumpSpeed_Sensor.C_FI(&MainNode.DI_P[10]);
      PumpOutput.C_PWMO_HB(&MainNode.PWM_HL8[1]);
```

…

The benefit of the second approach is that it allows you to assign properties to inputs and outputs depending on the user settings or a configuration file.
It also allows for a mixed variation. Because pointers hand over properties to the individual ports, any changes introduced at a later point (e.g. the node ID or CAN number) will come into effect.

7.2 SetPorttype

Setting the port type makes the port configuration known to the node. This is the first occurrence of SDOs being sent to the respective node. Therefore, we recommend querying the node's online state:

```
if MainNode.HeartbeatState() != 0x7F then
      //Node not in Preoperational
end_if;

if Lights_Back_Node.HeartbeatState() != 0x7F then
       //Node not in Preoperational
end_if;
```

Call the inputs and outputs defined above as follows:

```
BrakeLight_Left.setporttype();
TurnSignal_left_back.setporttype();

PumpSpeed_Sensor.setporttype();
PumpOutput.setporttype();
```

No additional parameters are required. While the constructor has handed the parameters over to the respective class, it is not until now that additional properties (e.g. maximum current) are passed on.
Ignoring this step would result in the port not being properly configured making it impossible for the node to process inputs, outputs and parameters correctly.

7.3 Mapping

Once the ports have been assigned the corresponding parameters, the values can be mapped. This works for all values of the input and output classes prefixed `I_` or `O_`.
The Mobile IO API automatically distributes the values to the respective PDOs.

```
BrakeLight_Left.O_Logic_Level.map();
TurnSignal_left_back.O_Logic_Level.map();

PumpSpeed_Sensor.I_Frequency.map();

PumpOutput.O_Dutycycle.map();
PumpOutput.I_Current.map();
```

The first node switching to Operational state enables the PDOs.
As soon as the mapping has been enabled, communication is sent via PDO instead of SDO. Communication is asynchronous. It does not wait for the node to respond. This enables the controller to process code faster. Do not map values that rarely change as this places unnecessary load on the bus.

7.4 Setoperational

To enable the CAN PDOs and activate the outputs, all components are set to Operational state. Component can be set to Operational one by one:

```
IO_Can.setOperational();
MainNode.SetOperational();
Lights_Back_Node.SetOperational();
```

Or all at once:

```
IO_Can.SetCmdNMTall(T_CanOpenCmd.START);
```

The configuration is complete. The nodes are ready to be used.

7.5 Set/Get

You can call the `Get` and `Set` functions of the individual parameters and values while the node is in both, Preoperational and Operational state.
This is what ensures communication with the node in the first place.
The parameters are set or read via PDO or SDO.

Example:

```
PumpSpeed_Sensor.IO_Timeout_ms.set(1000);
PumpSpeed_Sensor.IO_GateTime_ms.set(500);
PumpOutput.IO_Frequency.set(1000);
BrakeLight_Left.O_Logic_Level.set(1);

if flashlight_left_active then
	TurnSignal_left_back.O_Logic_Level.set(1);
	delay(500);
	TurnSignal_left_back.O_Logic_Level.set(0);
	delay(500);
end_if;
if PumpSpeed_Sensor.I_Frequency.get() < 1000 then
		PumpOutput.O_Dutycycle.set(PumpOutputValue_TMP+20);
	else
		PumpOutput.O_Dutycycle.set(PumpOutputValue_TMP-20);
end_if;
```

8 Examples

8.1 Universal hydraulic function

The following example shows how to the create a class including 2 inputs and 2 outputs. Its universal configuration allows the class to be used in various hydraulic functions.

```
type
	C_hydraulicFunction :class

		private Output_open		: C_DO;
		private Output_close		: C_DO;
		private input_limit_switch_close : C_DI;
		private input_limit_switch_open	: C_DI;

		public function C_hydraulicFunction(
				Interface_open: T_P_InterfacePort :=NULL,
				Interface_close: T_P_InterfacePort:=NULL,
				Interface_sensor_open: T_P_InterfacePort:=NULL,
				Interface_sensor_close: T_P_InterfacePort:=NULL);

		public function init();
		public function close(maxtime_ms :int);
		public function open(maxtime_ms :int);

	end_class
end_type;
// Konstruktor
function C_hydraulicFunction.C_hydraulicFunction

	this.Output_open.C_DO(Interface_open);
	this.Output_close.C_DO(Interface_close);
	this.input_limit_switch_close.C_DI(Interface_sensor_close);
	this.input_limit_switch_open.C_DI(Interface_sensor_open);

end_function

function C_hydraulicFunction.init()

	this.Output_open.setporttype();
	this.Output_open.O_Logic_Level.map();

	this.Output_close.setporttype();
	this.Output_close.O_Logic_Level.map();
```

```
	this.input_limit_switch_close.setporttype();
	this.input_limit_switch_close.I_Logic_Level.map();

	this.input_limit_switch_open.setporttype();
	this.input_limit_switch_open.I_Logic_Level.map();

end_function

function C_hydraulicFunction.open

	this.Output_open.O_Logic_Level.set(1);

	when this.input_limit_switch_open.I_Logic_Level.get() = 1 then
	else_time maxtime_ms then
	end_when

	this.Output_open.O_Logic_Level.set(0);

end_function

function C_hydraulicFunction.close

	this.Output_close.O_Logic_Level.set(1);

	when this.input_limit_switch_close.I_Logic_Level.get() = 1 then
	else_time maxtime_ms then
	end_when

	this.Output_close.O_Logic_Level.set(0);

end_function

var
	IO_Can              : C_CanOpenMaster;
	MainNode            : C_JSCM_720(&IO_Can,0x40);

	BroomUpDown   : C_hydraulicFunction();
	BroomInOut    : C_hydraulicFunction();

end_var

task hydraulicSteering autorun

	IO_Can.init();
```

```
	//readConfig

	BroomInOut.C_hydraulicFunction(&MainNode.PWM_H3[1],&MainNode.PWM_H3[2],
				&MainNode.DI[1], &MainNode.DI[2]);
	BroomUpDown.C_hydraulicFunction(&MainNode.PWM_H3[3],&MainNode.PWM_H3[4],
				&MainNode.DI[3], &MainNode.DI[4]);

	//init everything
	BroomInOut.init();
	BroomUpDown.init();

	IO_Can.setOperational();
	MainNode.SetOperational();

	loop

		BroomInOut.open(5000);
		BroomUpDown.open(5000);

		delay(100);

		BroomInOut.close(3000);
		BroomUpDown.close(2500);

		delay(100);

	end_loop
end_task;
```

8.2 Saving the parameters

```
var
	IO_Can		: C_CanOpenMaster;
	MainNode	: C_JSCM_720(&IO_Can,0x40);

	BeakonLeft	: C_DO(&MainNode.PWM_HL8[1]);

	StoredCRC	: int at %rl 1000000;

end_var

task StoreConfig autorun

	IO_Can.init();

	when MainNode.HeartbeatState() = 0x7F continue;

	if StoredCRC = MainNode.SystemParameter_CRC.get() then

		BeakonLeft.O_Logic_Level.map(,,true);
                                //parameters just set on PLC, not on Node
	else

		BeakonLeft.setporttype();
		BeakonLeft.O_Logic_Level.map();

		MainNode.Settings_IO_store();

		when MainNode.Settings_IO_store_cl.get() = 1 continue;

		delay(200);

		StoredCRC := MainNode.SystemParameter_CRC.get();

	end_if;

	// normal operation of the device

end_task;
```