BUCHER
automation

Themenhandbuch

60888036_01

Mobil-IO-API

Dieses Dokument wurde von der Bucher Automation AG mit der gebotenen Sorgfalt und basierend auf dem ihr bekannten Stand der Technik erstellt. Änderungen und technische Weiterentwicklungen an unseren Produkten werden nicht automatisch in einem überarbeiteten Dokument zur Verfügung gestellt.
Die Bucher Automation AG übernimmt keine Haftung und Verantwortung für inhaltliche oder formale Fehler, fehlende Aktualisierungen sowie daraus eventuell entstehende Schäden oder Nachteile.

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

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

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

www.bucherautomation.com

Originaldokument

Dokumentenversion: 1.00.1
Ausgabedatum: 07.04.2025

Inhaltsverzeichnis

1 Anwendungsbereich

Bei der Mobil-IO-API handelt es sich um eine Bibliothek, die bevorzugt im Bereich der mobilen Automatisierung angewendet werden kann. Eine Anwendung in industriellen Anlagen ist nicht ausgeschlossen.
Dieses Themenhandbuch beschreibt die Anwendung der Bibliothek in der Version 1.00.0.00.
Der Anwendungsbereich der Bibliothek ist das Ansprechen und Konfigurieren der CANopen-Peripherieknoten von Bucher Automation (z. B. JSCM-720-E04, JXM-IO-E30, JXM-IO-E31 und JXM-IO-E32).
Die Bibliothek wird in der Programmierumgebung JetSym eingebunden und auf einer STX-fähigen Steuerung angewendet.

2 Voraussetzungen

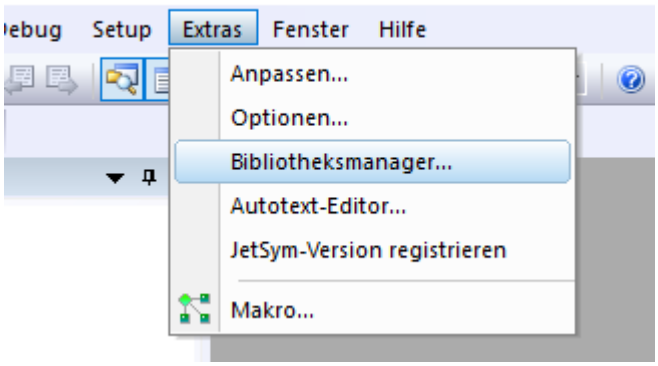
Um die Bibliothek verwenden zu können, sind die folgenden Voraussetzungen notwendig:

- Grundkenntnisse in objektorientierter Programmierung
- Programmierkenntnisse in JetSym-STX
- Programmierumgebung JetSym (Version 5.6 oder höher)
- STX-fähige Steuerung (OS-Version ab 4.07 oder Plattformversion 0x0109)

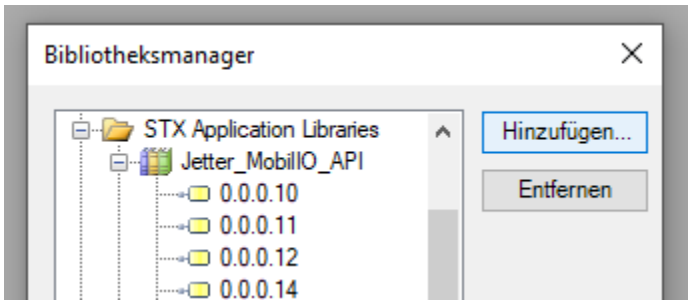
3 Bibliothek in JetSym einbinden

Um die Bibliothek in JetSym einzubinden, führen Sie in JetSym die folgenden Schritte aus:

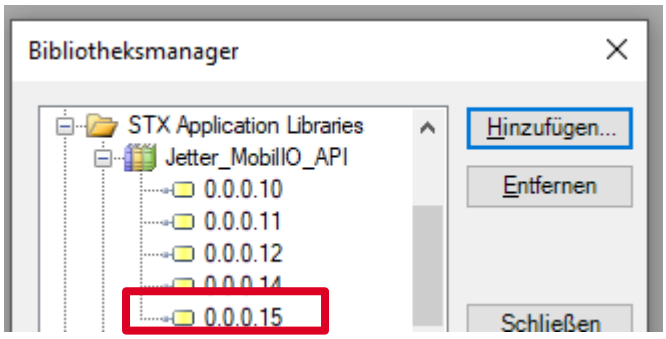
1. Wählen Sie Extras > Bibliotheksmanager… aus.



⇨ Das Fenster Bibliotheksmanager öffnet sich.

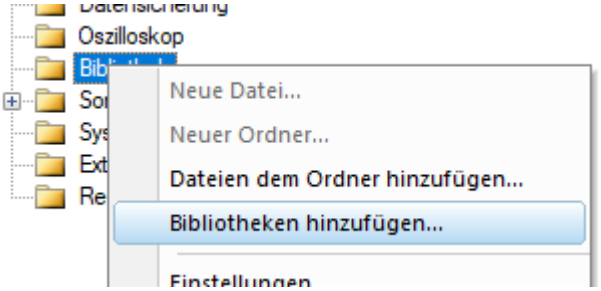


2. Wählen Sie über Hinzufügen… die aktuelle Version der Bibliothek aus. Wenn die aktuelle Version der Bibliothek bereits vorhanden ist, kann dieser Schritt übersprungen werden.



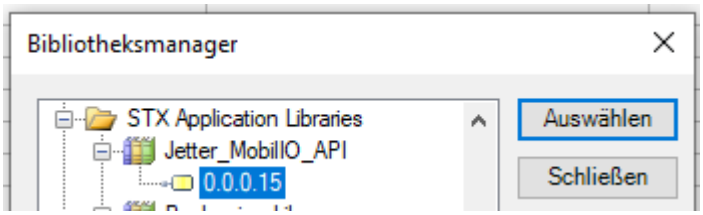
⇨ Die aktuelle Version der Bibliothek wurde hinzugefügt und kann in Ihr Projekt eingebunden werden.

3. Wählen Sie im Dateibaum Ihres Projekts via Rechtsklick auf den Ordner Bibliothek > Bibliotheken hinzufügen… aus.



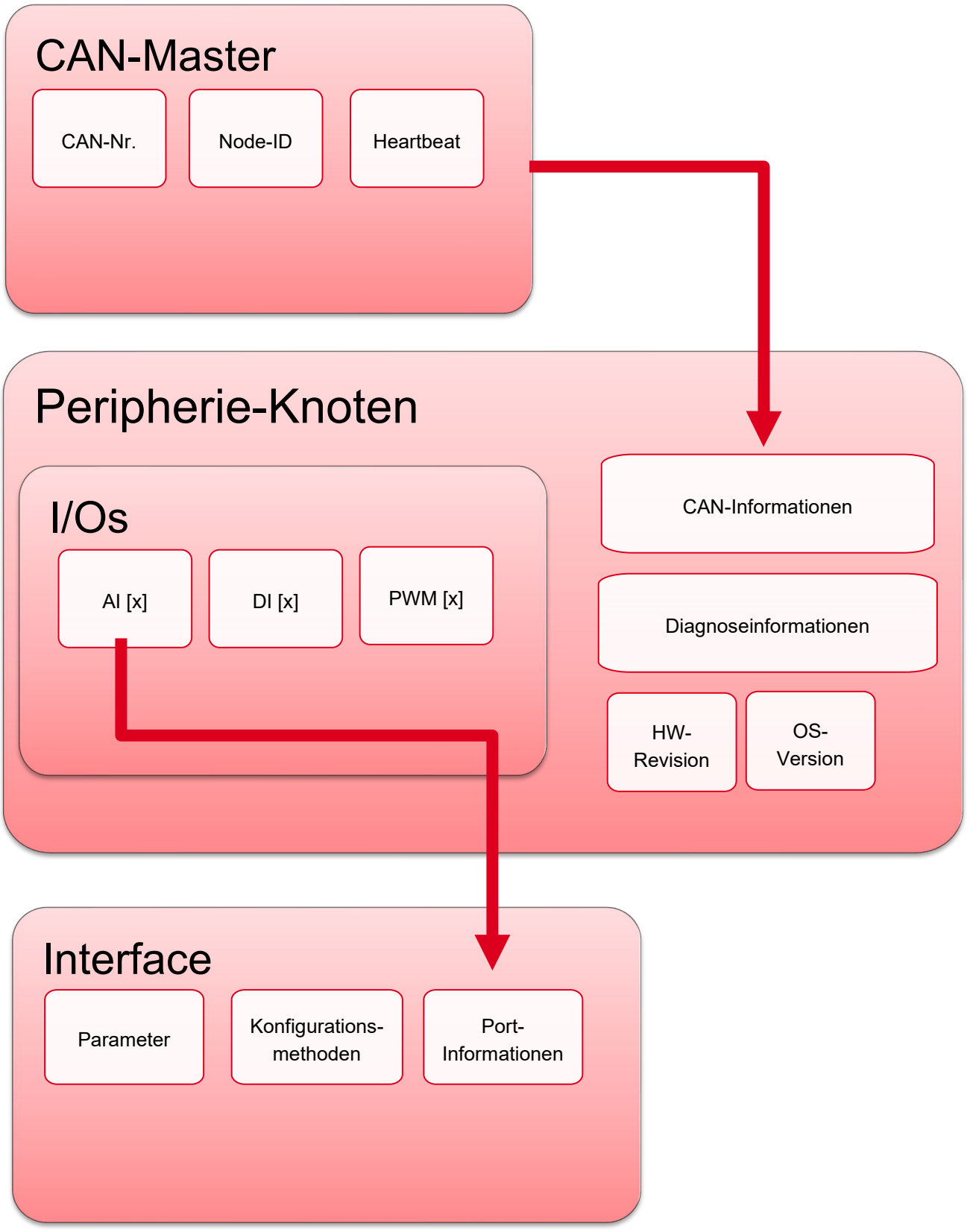
⇨ Das Fenster Bibliotheksmanager öffnet sich.

4. Wählen Sie die aktuelle Version der Bibliothek aus und bestätigen Sie mit Auswählen.



⇨ Sie können nun alle Klassen und Objekte der Bibliothek in Ihrem Projekt verwenden. Eine weitere Anweisung `#include` ist nicht notwendig.

4 Übersicht


CAN-Master
CAN-Nr.
Node-ID
Heartbeat
Peripherie-Knoten
I/Os
AI [x]
DI [x]
PWM [x]
CAN-Informationen
Diagnoseinformationen
HW-Revision
OS-Version
Interface
Parameter
Konfigurations-methoden
Port-Informationen

5 Komponenten

Im folgenden Kapitel sind die einzelnen Komponenten der Mobil-IO-API und deren Verwendung beschrieben.

5.1 C_CanOpenMaster

Die Klasse `C_CanOpenMaster` stellt die Grundlage zur Kommunikation auf dem entsprechenden CANopen-Bus zur Verfügung. Hier können zentral essenzielle Daten zur Parametrierung des CAN-Bus hinterlegt werden, wie z. B. Baudrate oder CAN-Nummer.
Für jede CAN-Schnittstelle der Steuerung muss eine neue Instanz dieser Klasse verwendet werden.

5.1.1 Verwendete Typen

Um die Verwendung dieser Klasse zu vereinfachen, wurden 2 Enumerationen definiert.
Das Enum `T_CANOpenCmd` enthält die verschiedenen Zustände, die in der Funktion `SetCmdNMTall(cmd: T_CanOpenCmd)` verwendet werden.

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

Das Enum `T_CanOpenBaud` enthält die verschiedenen Baudraten, die im Konstruktor der Klasse übergeben werden können.

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

5.1.2 Konstruktor

Die Klasse wird mit folgendem Konstruktor aufgerufen:

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

Dabei können alle wichtigen Parameter direkt an die Klasse übergeben werden.

Beispiel für das Anlegen von 2 CAN-Bussen:

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

5.1.3 NMT-Nachrichten

Wenn alle Teilnehmer des CAN-Busses in einen der Zustände Operational, Preoperational oder Stopped gesetzt werden sollen, dann kann die folgende Funktion verwendet werden:

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

Beispiel:

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

Dieser Aufruf startet alle Komponenten am HMI-CAN und versetzt den HMI-CAN in den Zustand Operational.

5.1.4 Funktionen CAN-Master

Diese Funktionen ändern ausschließlich den CAN-Zustand der Steuerung, auf welcher der Code ausgeführt wird.
Wenn diese Funktion aufgerufen wird, dann wird die Schnittstelle initialisiert und auf dem CAN-Bus ist ein Heartbeat zu sehen:

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

Mit diesen Funktionen wird der Status der CAN-Schnittstelle selbst geändert und dadurch die PDOs aktiviert oder deaktiviert.

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

5.2 Basisknoten

Das Ziel der Mobil-IO-API ist das Ansprechen und einfache Verbinden der SPS mit CANopen-Knoten. Die verschiedenen Knoten können alle gleichartig und in verschiedenen Konstellationen zusammen verwendet und eingebunden werden. Die Ein- und Ausgänge der Knoten können flexibel untereinander ausgetauscht werden, um Maschinen mit verschiedenen Konfigurationen flexibel einsetzen zu können. Dieser Abschnitt setzt sich mit den generellen Funktionen der Knoten und dann mit den speziellen Eigenschaften auseinander.
Die Klasse `C_MobilIO_Base_Node` wird in den später beschriebenen Knoten vererbt, kann allerdings auch als Grundlage für verschiedene eigene CANOpen-fähige Komponenten (z.B. für eine Display-Klasse) verwendet werden.

5.2.1 Konstruktor

Zunächst wird der Knoten im Steuerungsprogramm angelegt:

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

Dabei wird ein Pointer auf den bereits angelegten CANopen-Master übergeben. Hier werden alle Daten, wie CAN-Nummer und Semaphoren übergeben.
Weiter wird die Node-ID angegeben, welche für die gesamte Kommunikation benötigt wird.

5.2.2 NMT-Nachrichten der Knoten

Die folgenden Funktionen werden verwendet, um den CAN-Status des einzelnen Knoten zu ändern:

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

Mit der folgenden Funktion kann der Status des Knotens ausgewertet werden:

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

Rückgabewert	Bedeutung
0	Bootup
4	Stopped
5	Operational
127	Preoperational
255	Offline (Default-Wert)

Je Knoten können die Heartbeats von bis zu 4 CAN-Teilnehmern überwacht werden. Die Eigenschaften der zu überwachenden Heartbeats lässt sich mit der entsprechenden Master-Node-ID und entsprechendem Timeout über die Steuerung einstellen. Wenn das Gerät keinen Heartbeat innerhalb der angegebenen Timeout-Zeit erkennt (z. B. im Falle eines Kommunikationsabbruchs), erfolgt der Wechsel in den Zustand Stopped und die Ausgänge werden spannungsfrei geschaltet.

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation beim Anlegen des Heartbeat-Monitoring
-2	Fehler in SDO-Kommunikation beim Setzen der Heartbeat-Anzahl
-3	Fehler: zu viele Heartbeats

5.2.4 Systemparameter

Über die Systemparameter können die einzelnen Parameter des Knoten gelesen oder geschrieben werden.
Die folgende Funktion fragt ab, wie groß der Wert für die CRC ist.

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

Mit der CRC kann geprüft werden, ob die Einstellungen ins Gerät übertragen werden müssen.
Mit der folgenden Funktion kann die Baudrate des Knotens gesetzt werden:

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

Wert	Baudrate
0	125 kBaud
1	250 kBaud
2	500 kBaud
3	1000 kBaud

Die Einstellung der Node-ID erfolgt mit der Funktion:

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

Die Werte der Config-Pins werden mit der folgenden Funktion ausgelesen:

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

Die Summe der Parameter für die Node-ID und den Node-ID-Offset ergibt die endgültige Node-ID, auf welcher der Knoten nach einem Neustart erreichbar ist.
Die folgende Funktion stellt die Zeit zwischen 2 Heartbeats des Knotens ein:

```
public Heartbeat_time      : C_Heartbeat_time_setting;
```

5.2.5 Einstellungen remanent speichern

Um gewählte Einstellungen (z. B. Mapping oder Port-Typen) remanent auf dem Knoten zu speichern, wird die folgende Funktion verwendet:

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

Dieser Vorgang benötigt auf unterschiedlichen Knoten unterschiedlich lange, was eine Dauer von bis zu 20 s bedeuten kann. Um zu sehen, ob der Speichervorgang beendet ist, wird auf dem JSCM-720-E04 der Rückgabewert der folgenden Funktion verwendet:

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

Rückgabewert	Bedeutung
1	OK
-1	Fehler in SDO-Kommunikation
0	Fehler beim Schreiben der Settings

Um die Einstellungen wieder auf die Standartwerte zu setzen, wird folgende Funktion verwendet:

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

Rückgabewert	Bedeutung
1	OK
0	Fehler beim Schreiben der Settings
-1	Fehler in SDO-Kommunikation

Die Änderungen sind erst nach einem Neustart wirksam.

Beispiel:

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

Ein ausführliches Beispiel ist zudem in Kapitel 8.2 zu finden.

5.2.6 Systeminformationen

Die folgenden Klassen enthalten die Funktion `.get()`. Mit der Funktion `.get()` können Systeminformationen abgerufen werden:

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

Fehler des Gesamtsystems können ebenfalls über die Funktion `.get()` der folgenden Klasse ausgelesen werden:

```
public Status_Information  : C_Status_information;
```

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
Bit	**Beschreibung**
0	Allgemeiner Fehler
1	Gesamter Überstrom
3	Temperatur
4	Kommunikationsfehler
7	I/O-Fehler

5.2.7 Fehlerhistorie

Die Software stellt einen Fehlerspeicher zur Verfügung. Der Fehlerspeicher zeigt die letzten 256 Fehler an. Die folgenden Funktionen können verwendet werden:

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

5.2.8 Betriebssystemupdate

Die OS-Datei muss im Format `.os` oder `.hex` vorliegen. Das Betriebssystemupdate eines Knotens wird mit folgenden Funktionen durchgeführt:

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

`OSFilePath` übergibt den Pfad für das neue OS an die Funktion. Über `blocking` wird definiert, ob die Steuerung warten soll, bis das Update erfolgreich abgeschlossen ist (nach 250 s wird das Update abgebrochen). Sollte hier false verwendet werden, dann muss gewährleistet sein, dass das Update erfolgreich abgeschlossen wurde, bevor der Knoten mit neuer SDO-Kommunikation beaufschlagt wird.

Rückgabewert	Bedeutung
0	OK
-1	Timeout (Ziel antwortet nicht in der vorgegebenen Zeit)
-2	Ungültiger Parameter, z.B.: Downloadstatus ist keine globale Variable
-10	Dateigröße nicht korrekt
-11	Dateiname ungültig
-12	Auf den Knoten wird bereits ein Download durchgeführt
-13	Eigene Node-ID
-14	Ungültige Produkt-ID (OS passt nicht zur Ziel-Node-ID)
-15	Schreiben in Flash des Ziels nicht erfolgreich
-16	Kommunikationsfehler während der Übertragung
-17	Zielknoten nicht erreichbar (offline)
-20	Abbruch durch Anwender

-30	Update wird von Ziel-Node-ID nicht unterstützt
-40	Fehler im Updateprozess

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

Hiermit kann der Status des Betriebssystemupdates zurückgelesen werden (0 … 1.000 ‰)
Nach erfolgreichem Download muss der Knoten neu gestartet werden.

5.2.9 Mapping auf Steuerung löschen

Wenn das Mapping von Parametern eines Knotens nochmals durchgeführt werden soll, dann müssen zunächst die bisher gespeicherten Werte für das Mapping bezogen auf diesen Knoten gelöscht werden. Hierfür steht folgende Funktion zur Verfügung:

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

Um die gesamte Konfiguration auf der Steuerung zu löschen, sollte nun die Funktion `ResetInterface();` für jedes Interface einzeln (siehe Kapitel 6.1.6) verwendet werden.

5.3 JSCM-720-E04

Im Folgenden werden die Elemente beschrieben, die speziell für den JSCM-720-E04 verfügbar sind.

5.3.1 I/Os

Auf dem JSCM-720-E04 stehen die folgenden Ein- und Ausgänge zur Verfügung:

```
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]
```

Die Ein- und Ausgänge können als Parameter in den verschiedenen Interfaces via Pointer verwendet werden.
Die genauen Eigenschaften der Ein- und Ausgänge lesen Sie bitte in der gerätespezifischen Dokumentation nach.

5.3.2 Applikationsparameter speichern

Die Software kann 1.020 U32-Applikationsparameter remanent speichern.
Die Applikationsparameter können über den folgenden Aufruf gelesen werden:

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

Die Applikationsparameter können über den folgenden Aufruf geschrieben werden:

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

Dabei kann der Parameter `valueNumber` die folgenden Werte annehmen: 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;
```

Ebenfalls ist es möglich, die Parameter über eine gesammelte Funktion zu lesen und zu schreiben.
Dabei kann der Parameter `valueNumber` die folgenden Werte annehmen: 1 … 1.020.

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt
-3	Parameter nicht im zulässigen Bereich

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

Rückgabewert	Bedeutung
≥ 0	Inhalt des Parameters
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt
-3	Parameter nicht im zulässigen Bereich

5.3.3 Diagnose-Spannungen

Die Software stellt folgende Diagnose-Spannungen zur Verfügung:

```
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

Im Folgenden werden die Elemente beschrieben, die speziell für den JXM-IO-E30 verfügbar sind.

5.4.1 I/Os

Auf dem JXM-IO-E30 stehen die folgenden Ein- und Ausgänge zur Verfügung:

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

Die Ein- und Ausgänge können als Parameter in den verschiedenen Interfaces via Pointer verwendet werden.
Die genauen Eigenschaften der Ein- und Ausgänge lesen Sie bitte in der gerätespezifischen Dokumentation nach.

5.5 JXM-IO-E31

Im Folgenden werden die Elemente beschrieben, die speziell für den JXM-IO-E31 verfügbar sind.

5.5.1 I/Os

Auf dem JXM-IO-E31 stehen folgende Ein- und Ausgänge zur Verfügung:

```
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]
```

Die Ein- und Ausgänge können als Parameter in den verschiedenen Interfaces via Pointer verwendet werden.
Die genauen Eigenschaften der Ein- und Ausgänge lesen Sie bitte in der gerätespezifischen Dokumentation nach.

5.6 JXM-IO-E32

Im Folgenden werden die Elemente beschrieben, die speziell für den JXM-IO-E32 verfügbar sind.

5.6.1 I/Os

Auf dem JXM-IO-E32 stehen folgende Ein- und Ausgänge zur Verfügung:

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

Die Ein- und Ausgänge können als Parameter in den verschiedenen Interfaces via Pointer verwendet werden.
Die genauen Eigenschaften der Ein- und Ausgänge lesen Sie bitte in der gerätespezifischen Dokumentation nach.

6 Interfaces

Im Projekt werden die Schnittstellen über Instanzen der einzelnen Interface-Klassen angesprochen. Diese Interface-Klassen beinhalten alle von dieser Schnittstelle unterstützten Parameter und benötigten Funktionen.

6.1 Grundlagen

Es gibt verschiedene Member und Methoden der einzelnen Interfaces:

1. Konstruktor mit verschiedenen Parametern, um grundsätzliche Parameter (z. B. die Hardware-Schnittstelle) zu übergeben.
2. `SetPorttype()`, um die Konfiguration des Ports an den Knoten zu schreiben.
3. Ein- und Ausgangswerte des Interface, die auf PDOs gemapped werden können.
4. Parameter des Interface.
5. `ResetInterface()`, um alle Parameter des Interfaces zu löschen.

6.1.1 SetPorttype

Diese Funktion wird verwendet, um die einzelnen Ports zu konfigurieren.
Alle Parameter für die Konfiguration sind bereits über den Konstruktor gespeichert.

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.1.2 Eingangswerte

Eingangswerte sind durch ein `I_` am Beginn des Namens gekennzeichnet.
Mit der folgenden Funktion kann der Inhalt des Eingangswerts ausgelesen werden:

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

Rückgabewert	Bedeutung
≥ 0	Inhalt des SDO
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt

Zudem wird die Funktion `map` bereitgestellt, mit der das PDO-Mapping auf dem Knoten und auf der Steuerung durchgeführt wird. Die Mapping-Position wird dabei automatisch durch die API ermittelt.

```
function map(EventTime :int := 1000,
             // Zykluszeit, in der ein Telegramm empfangen werden soll

                InhibitTime :int := 200,
             // Mindestabstand zwischen zwei empfangenen Telegrammen

                alreadyStoredOnNode :bool := false
             // bei Speicherung: PDO wird nur auf Controller angelegt
               );
```

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt
-3	SDO – PDO auf ungültig setzen – Fehler
-4	SDO – Mapping-Zähler löschen – Fehler
-5	Mapping: SDO – Mapping-Eintrag setzen – Fehler
-6	SDO – Mapping-Zähler setzen – Fehler
-7	SDO – Inhibit-Time setzen – Fehler
-8	SDO – Event-Time setzen – Fehler
-9	Nachrichten-ID setzen - Fehler
-10	PDO – auf Steuerung - Fehler
-11	Zu viele Werte gemappt.

6.1.3 Ausgangswerte

Ausgangswerte sind durch ein `O_` am Beginn des Namens gekennzeichnet.
Mit der folgenden Funktion kann der Inhalt des Ausgangswerts ausgelesen werden:

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt

Mit der folgenden Funktion kann der Inhalt des Ausgangswerts zurückgelesen werden:

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

Rückgabewert	Bedeutung
≥ 0	Inhalt des SDO
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt

Zudem wird die Funktion `map` bereitgestellt, mit der das PDO-Mapping auf dem Knoten und auf der Steuerung durchgeführt wird. Die Mapping-Position wird dabei automatisch durch die API ermittelt.

```
function map(EventTime :int := 1000,
             // Zykluszeit, in der ein Telegramm empfangen werden soll

                InhibitTime :int := 200,
             // Mindestabstand zwischen zwei empfangenen Telegrammen

                alreadyStoredOnNode :bool := false
             // bei Speicherung: PDO wird nur auf Controller angelegt
               );
```

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt
-3	SDO – PDO auf ungültig setzen – Fehler
-4	SDO – Mapping-Zähler löschen – Fehler
-5	Mapping: SDO – Mapping-Eintrag setzen – Fehler
-6	SDO – Mapping-Zähler setzen – Fehler
-7	SDO – Inhibit-Time setzen – Fehler
-8	SDO – Event-Time setzen – Fehler
-9	Nachrichten-ID setzen - Fehler
-10	PDO – auf Steuerung - Fehler
-11	Zu viele Werte gemappt.

6.1.4 Parameter

Parameter sind durch ein `IO_` am Beginn des Namens gekennzeichnet.
Mit der folgenden Funktion kann der Inhalt des Parameters gesetzt werden:

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt

Mit der folgenden Funktion kann der Inhalt des Parameters zurückgelesen werden:

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

Rückgabewert	Bedeutung
≥ 0	Inhalt des SDO
-1	Fehler in SDO-Kommunikation
-2	Interface nicht korrekt angelegt

6.1.5 Port-Status

Der Port-Status ist ein Parameter, der über dieselben Funktionen wie ein gewöhnlicher Parameter verfügt. Allerdings ist der Rückgabewert nicht vom Typ `int`, sondern vom Typ `T_Status`.

```
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 Interface zurücksetzen

Um alle Parameter des Interface zu löschen, wird folgende Funktion bereitgestellt:

```
ResetInterface()
```

Verwendet wird diese, wenn die Parametrierung nochmals durchlaufen werden soll, ohne die Steuerung neu zu starten. Auf dem Knoten passiert in diesem Fall nichts. Um die Parameter auf dem Knoten zu löschen, muss dieser neu gestartet werden, oder bei gespeicherter Parametrierung `Settings_IO_reset()` ausgeführt werden.

6.2 C_AI_Current

6.2.1 Konstruktor

```
public function C_AI_Current(InterfacePort :T_P_InterfacePort:=NULL
                                                  // Hardware Interface
                                    );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.2.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.2.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_AI_Temperature (InterfacePort :T_P_InterfacePort:=NULL
                                                    // Hardware Interface
                                  );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.3.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.3.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_AI_VoltageHighRange (InterfacePort :T_P_InterfacePort:=NULL
                                                      // Hardware Interface
                                       );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.4.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.4.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_AI_VoltageRatio (InterfacePort :T_P_InterfacePort:=NULL
                                                  // Hardware Interface
                                  );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.5.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.5.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_AO_PVG (InterfacePort :T_P_InterfacePort:=NULL // Hardware Interface
                          );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.6.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.6.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_AO_Voltage (InterfacePort :T_P_InterfacePort:=NULL
                                                  // Hardware Interface
                                 );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.7.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.7.3 Ein- und Ausgangswerte

```
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 Konstruktor

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

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet
-3	NPN/PNP unterscheidet sich von anderen Konfigurationen im selben Block

6.8.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.8.3 Ein- und Ausgangswerte

```
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 Konstruktor

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

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet
-3	NPN/PNP unterscheidet sich von anderen Konfigurationen im selben Block

6.9.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.9.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_DO(InterfacePort :T_P_InterfacePort:=NULL,       // Hardware Interface
                     MaxCurrentmA :int := -1                       // Maximaler Strom
                    );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.10.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.10.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_DO_LS (InterfacePort :T_P_InterfacePort:=NULL,   // Hardware Interface
                         MaxCurrentmA :int := -1
                         // Maximaler Strom (bei -1 wird der default Wert verwendet)
                         parallel :bool := false          //parallel 1||2, 3||4, 5||6
                         );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.11.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.11.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_PWMO_FB (InterfacePort :T_P_InterfacePort:=NULL, // Hardware Interface
                           MaxCurrentmA :int := -1
                           // Maximaler Strom (bei -1 wird der default Wert verwendet)
                           parallel :bool := false          //parallel 1||2, 3||4, 5||6
                           );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet
-3	Port ist nicht für die Parallelschaltung freigegeben

6.12.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.12.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_PWMO_HB (InterfacePort :T_P_InterfacePort:=NULL, // Hardware Interface
                           MaxCurrentmA :int := -1
                           // Maximaler Strom (bei -1 wird der default Wert verwendet)
                           parallel :bool := false          // parallel 1||2, 3||4, 5||6
                           );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet
-3	Port ist nicht für die Parallelschaltung freigegeben

6.13.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.13.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_PWMO_HS (InterfacePort :T_P_InterfacePort:=NULL, // Hardware Interface
                           MaxCurrentmA :int := -1
                           // Maximaler Strom (bei -1 wird der default Wert verwendet)
                           parallel :bool := false          //parallel 1||2, 3||4, 5||6
                           );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet
-3	Port ist nicht für die Parallelschaltung freigegeben

6.14.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.14.3 Ein- und Ausgangswerte

```
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 Konstruktor

```
public function C_CPWMO_HS (InterfacePort :T_P_InterfacePort:=NULL,
                                  // Hardware-Interface
                            MaxCurrentmA :int := -1
                            // Maximaler Strom (bei -1 wird der default Wert verwendet)
                            );
```

Rückgabewert	Bedeutung
0	OK
-1	Interface wird für diesen Port nicht unterstützt
-2	Port wird bereits verwendet

6.15.2 SetPorttype

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

Rückgabewert	Bedeutung
0	OK
-1	Fehler in SDO-Kommunikation
-5	Interface nicht korrekt konfiguriert

6.15.3 Ein- und Ausgangswerte

```
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 Anwendung

In diesem Kapitel wird die Anwendung der Mobil-IO-API erklärt. Die folgenden Schritte müssen in der angegebenen Reihenfolge und bezogen auf den einzelnen Port bearbeitet werden.
Die Namen der einzelnen Komponenten (z. B. `MainNode`) sind Beispiele und können bezogen auf den einzelnen Anwendungsfall individuell angepasst werden.

7.1 Constructor

Der jeweilige Konstruktor kann direkt bei der Variablendefinition aufgerufen werden:

```
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;
```

Den gleichen Inhalt bekommt man, wenn man den Konstruktor erst später im Code aufruft.

```
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]);
```

…

Die zweite Variante hat den Vorteil, dass hier abhängig von Einstellungen oder einer Konfigurationsdatei die Eigenschaften der Ein- und Ausgänge zugewiesen werden können.
Hier ist auch eine Mischvariation möglich. Da die Eigenschaften durch Pointer in die verschiedenen Ports gegeben werden, kommen auch nachträgliche Änderungen (z. B. der Node-ID oder der CAN-Nummer) zum Tragen.

7.2 SetPorttype

Durch das Setzen des Port-Typen wird dem Knoten mitgeteilt, wie die einzelnen Ports konfiguriert sind. Hier werden das erste Mal SDOs an den entsprechenden Knoten gesendet. Somit ist eine Abfrage, ob der entsprechende Knoten online ist, ratsam:

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

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

Der Aufruf für die oben definierten Ein- und Ausgänge lautet wie folgt:

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

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

Weitere Parameter sind hier nicht nötig. Die Parameter wurden bereits im Konstruktor an die entsprechende Klasse übergeben. Übertragen werden die zusätzlichen Eigenschaften (z. B. die maximale Stromstärke) allerdings erst jetzt.
Wenn dieser Schritt übersprungen wird, dann ist der entsprechende Port nicht konfiguriert und die Ein- und Ausgangswerte sowie die Parameter werden vom Knoten nicht korrekt verarbeitet.

7.3 Mapping

Nachdem den Ports ihre Parameter zugewiesen sind, können einzelne Werte gemapped werden. Dies funktioniert für alle Werte der Ein- bzw. Ausgangsklasse, die mit `I_` oder mit `O_` beginnen.
Die Verteilung der einzelnen Werte auf die verschiedenen PDOs erfolgt automatisch durch die Mobil-IO-API.

```
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();
```

Die PDOs werden aktiviert, sobald der entsprechende Teilnehmer in den Status Operational geht.
Sobald das Mapping aktiviert ist, wird die Kommunikation nicht mehr per SDO, sondern per PDO gesendet. Die Kommunikation verläuft nun asynchron. Es wird nicht mehr auf eine Antwort des Knotens gewartet. Somit kann die Steuerung den Code schneller verarbeiten.
Mappen Sie keine Werte, die nur selten geändert werden. Der CAN-Bus wird sonst unnötig belastet.

7.4 Setoperational

Um die CAN-PDOs zu aktivieren und die Ausgänge aktiv zu schalten, werden nun alle Komponenten in den Zustand Operational gesetzt. Jede Komponente kann einzeln auf Operational gesetzt werden:

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

Alternativ kann der Status aller Komponenten zusammen gesetzt werden:

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

Die Konfiguration ist beendet und die Knoten können verwendet werden.

7.5 Set/Get

Die Funktionen `Get` und `Set` der einzelnen Parameter und Werte können sowohl im Status Preoperational als auch im Status Operational des Knotens verwendet werden.
Hierüber wird die eigentliche Kommunikation mit dem Knoten gewährleistet.
Die Parameter werden dann entweder per PDO oder per SDO gesetzt oder gelesen.

Beispiel:

```
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 Beispiele

8.1 Universelle Hydraulikfunktion

Im folgenden Beispiel wird eine Klasse angelegt, die 2 Ausgänge und 2 Eingänge beinhaltet. Die Klasse kann universell in verschiedenen Hydraulikfunktionen verwendet werden.

```
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 Parameter speichern

```
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;
```