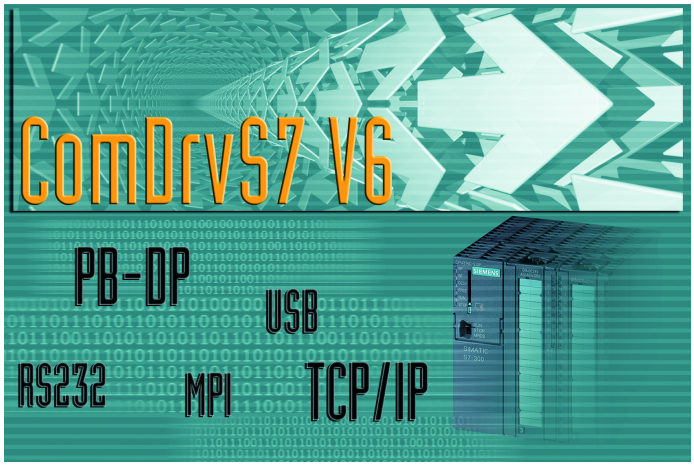


ComDrvS7 V6.2X

für LOGO!®, S7-1200®, S7-1500®, S7-300®, S7-400®,
VIPA 100V/200V/300S/SLIO

Dokumentation

für Softwareentwickler

Stand: Juli 2015

Mit Beispielen und Hinweisen zu
Visual C++ , Visual-Basic, Visual-C#, .Net, Delphi und Borland C++ Builder

---

MHJ-Software GmbH & Co. KG
Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

MHJ-Software GmbH & Co. KG

Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

Inhaltsverzeichnis

Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

1 Allgemeines zum ComDrvS7

Kurzbeschreibung

Mit der ComDrvS7-DLL hat man die Möglichkeit, Daten aus den AGs der S7-Reihe 1200/1500/300/400 von Siemens sowie den S7-kompatiblen CPUs der Reihen VIPA 100V/200V/300V/300S zu lesen und zu schreiben. Die Kommunikation kann dabei über MPI, Profibus-DP (mit NetLink, NETLink PRO oder SIMATIC®-NET) oder TCP/IP aufgebaut werden.

Ab der Version 6.25 wird die S7-1500® von Siemens unterstützt. Für die Kommunikation ist einer der beiden Open-Funktionen "MPI6_OpenTcpIp_S71500" bzw. "MPI6_OpenTcpIp_S71500Ext" zu verwenden.
Bei der Verwendung der S7-1500® von Siemens sind die im Kapitel "Notwendige Einstellung in der Hardwarekonfiguration einer S7-1500® von Siemens" angegebenen Hinweise zu beachten!

Ab der Version 6.23 ist eine 64-Bit DLL vorhanden. Mit dieser können 64-Bit Applikationen entwickelt werden. Diese Applikationen sind dann nur auf 64-Bit Betriebsystemen lauffähig. Wird eine 32-Bit Applikation mit der 32-Bit DLL entwickelt, so ist diese auf 32- und 64-Bit Betriebssystemen lauffähig.
Die 64-Bit DLL sollte verwendet werden, wenn die Speicherbegrenzungen von 32-Bit Applikationen ein Problem darstellen.
Bei der 64-Bit DLL ist der Zugriff auf SIMATIC®-NET nicht mehr möglich.

Ab der Version 6.20 können auch Daten aus einer LOGO! von Siemens gelesen und beschrieben werden. Dabei muss die Verbindung zur LOGO über Ethernet hergestellt werden. Es ist also eine LOGO ab 0BA7 notwendig.

Die verwendete DLL kann in Windows-Applikationen eingebunden werden, um mit einer SPS zu kommunizieren. Zur Kommunikation wird ein PC/MPI-Kabel (RS232, USB) bzw. der NetLink oder NETLink PRO(TCP/IP) benötigt. Weiterhin kann die Kommunikation über einen Ethernet-CP oder eine in der CPU integrierte Ethernetschnittstelle aufgebaut werden. Hierbei wird die Verbindung über ein herkömliches Ethernet-Kabel hergestellt.

Ab der Version 4 wird SIMATIC®-NET unterstützt, sofern die Treiber auf dem PC installiert sind. Damit ist es möglich, beispielsweise die CPs 5512, 5611 sowie den USB-MPI-Adapter von Siemens anzusprechen. Bei Vorhandensein des Teleservice V6 von Siemens, kann ComDrvS7 auch Daten über die Telefonleitung abfragen (z.B. mit Teleservice II-Adapter).
Ab der Version 4 kann eine CPU über Routing angesprochen werden. Somit muss eine anzusprechende CPU nicht direkt an den PC angeschlossen sein, die Anfrage kann über verschiedene Bussysteme hinweg zur CPU geroutet werden.

Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

Folgende Aktionen sind unter anderem mit der DLL möglich:

- Lesen/Schreiben von Bytes innerhalb der Operandenbereiche E, A, M und DB (E, A, M nicht in Lite-Version). Bei einer passwortgeschützten CPU muss das Passwort nicht bekannt sein.
- Lesen/Schreiben von Worten innerhalb der Operandenbereiche E, A, M und DB (E, A, M nicht in Lite-Version). Bei einer passwortgeschützten CPU muss das Passwort nicht bekannt sein.
- Lesen/Schreiben von Doppelworten innerhalb der Operandenbereiche E, A, M und DB (E, A, M nicht in Lite-Version). Bei einer passwortgeschützten CPU muss das Passwort nicht bekannt sein.
- Lesen/Schreiben von Zeiten und Zählern (nicht in Lite-Version).
- Lesen von DBs aus der CPU und speichern in einer WLD-Datei (nicht in Lite-Version).
- Lesen von DBs aus einer WLD-Datei und übertragen in die CPU (nicht in Lite-Version).
- RAM nach ROM kopieren zur Sicherung der Aktualwerte von DBs.
- Betriebszustand der CPU ermitteln und verändern (RUN, STOP schalten).
- Uhrzeit der CPU auslesen und verändern.
- Anzahl der im AG vorhandenen DBs auslesen
- Im AG vorhandene DB-Nummern ermitteln
- Länge von DBs in Bytes ermitteln
- Seriennummer der CPU und einer vorhandenen MMC auslesen
- Auslesen des Status der Fehler-LEDs SF, BF1 und BF2 einer CPU
- Abfrage der Notwendigkeit eines Passwortes für den Zugriff auf eine CPU
- Übergabe eines Passwortes an die CPU für den uneingeschränkten Zugriff auf eine passwortgeschützte CPU.
- Systembereiche ermitteln
- Stellung Betriebsartenschalter und Schutzstufen ermitteln
- Bestellnummer der CPU auslesen
- Erreichbare Teilnehmer am MPI/DP-Netz ermitteln
- Funktionen zum Wandeln von Real, DINT und INT-Operanden aus WORD- bzw. BYTE-Buffern.
- **Weitere Protokolle auf Anfrage.**

Die DLL ist in einer 32-Bit und 64-Bit-Version erhältlich.

Systemvoraussetzungen

Plattform: WinXP, Vista, Win7 (32 Bit oder 64 Bit), Win8 (32 Bit oder 64 Bit)
Plattform CE-Version: Windows CE 6.0 oder höher (ARM/x86)

1.1 Hardwarevoraussetzungen für die einzelnen Kommunikationswege

1.1.1 RS232/USB-Kommunikation

Es wird ein PC/MPI Kabel zur Verbindung zwischen dem PC und dem S7-AG der Reihe 300/400 benötigt. Des Weiteren kann ein USB-MPI-Kabel verwendet werden, sofern der USB-Treiber eine virtuelle serielle Schnittstelle anlegt.
Bestellnummern der Interface-Leitungen:

MPI-Adapter RS232: M007.001
MPI-Adapter USB: M007.005

1.1.2 Kommunikation über MHJ-Netlink oder MHJLink++ (inkl. Routing)

Hierbei dient ein **MHJ-NetLink** zur Verbindung zwischen Hub/Switch und dem S7-AG. Wird der MHJ-NetLink direkt an einer Netzwerkkarte eines PCs eingesteckt, so wird zusätzlich ein Cross-Over-Kabel benötigt.
Bestellnummern der Interface-Leitung:
MHJ-NetLink für MPI und Profibus-DP: M007.010

1.1.3 Kommunikation über TCP/IP-Direkt (inkl. Routing)

Verfügt die S7-SPS über einen Ethernet-CP oder besitzt die CPU eine integrierte Ethernet-Schnittstelle (auch Profinet), so kann die Kommunikation auch über diese Betriebsmittel hergestellt werden. Als Verbindung dient ein herkömmliches Ethernet-Kabel zwischen PC und dem CP. Der PC und der CP können ebenso über einen Hub/Switch miteinander verbunden werden. Der PC und der CP müssen sich im gleichen Subnetz befinden.

1.1.4 Kommunikation über TCP/IP-NETLink PRO (inkl. Routing)

Hierbei dient ein **NETLink PRO** als Verbindung zwischen Hub/Switch und dem S7-AG. Wird der NETLink PRO direkt an einer Netzwerkkarte eines PCs eingesteckt, so wird zusätzlich ein Cross-Over-Kabel benötigt.
Der **NETLink PRO** hat gegenüber dem normalen NetLink den Vorteil, dass dieser auf die Fernwartung über das Internet optimiert ist. Auch die Kommunikationsgeschwindigkeit des **NETLink PRO** ist höher als die des normalen NetLinks.
Bestellnummer der Interface-Leitung:
NETLink PRO (Ethernet) für MPI und Profibus-DP: M007.020

1.1.5 Kommunikation über SIMATIC®-NET (inkl. Routing)

Es müssen die Treiber SIMATIC®-NET auf dem PC installiert sein. Ist dies der Fall, so können die Siemens Interface-Adapter (z.B. CP5512, CP5611, USB-MPI-Adapter) verwendet werden. Bei Vorhandensein des Teleservice ab V6 ist auch die Fernabfrage über die Telefonleitung möglich. Dabei kann z.B. der Teleservice II-Adapter oder kompatible (siehe unter www.mhj.de) verwendet werden. Die Software Teleservice kann ebenso über www.mhj.de bezogen werden. Dieser Verbindungsweg ist nur mit der 32-Bit DLL möglich.

1.2 Installation des ComDrvS7

Sie erhalten den Treiber auf CD-ROM oder über einen Downloadlink.

Im Downloadbereich von www.mhj.de finden Sie immer die aktuelleste Version.

Nach der Installation befinden sich alle verfügbaren DLLs auf der Festplatte. Diese unterscheiden sich hinsichtlich der Programmiersprachen, in welchen diesen eingebunden werden können. Bitte benutzen Sie unbedingt nur die DLL, die für Ihre Programmiersprache vorgesehen ist.
Wenn eine DLL ohne Freischaltung verwendet wird, dann benutzen Sie automatisch eine Demoversion.

In der **Demoversion** wird bei jeder Open-Funktion eine **Demo-Meldung** angezeigt. Diese muss bestätigt werden. Die Demo-Meldung erscheint dann immer nach bestimmten Zeitabständen, bzw. beim erneuten Ausführen einer Open-Funktion.

Bitte lesen Sie die Datei README.TXT auf der CD für weitere Informationen.

1.3 Änderungen gegenüber älteren Versionen der ComDrvS7-DLL

1.3.1 Ab V6.25: ComDrvS7 unterstützt die S7-1500 von Siemens

1.3.2 Ab V6.23: ComDrvS7 ist nun auch als 64-Bit DLL erhältlich

1.3.3 Ab V6.20 Micro: Implementierung der Familie LOGO!® (ab 0BA7)

Ab der Version 6.20 können die Lese- und Schreibfunktionen auf die Operandenarten E, A, V einer LOGO!® ab 0BA7 angewendet werden. Dazu ist die spezielle Einleitungsfunktion "MPI6_OpenTcpIp_Logo" auszuführen.
Es ist zu beachten, dass der Zugriff nur in der Micro-Version von ComDrvS7 möglich ist!

1.3.4 Ab V6.1; Implementierung der Familie S7-1200®

Ab der Version 6.1 können die Lese- und Schreibfunktionen auf die Operandenarten E, A, M und DB einer S7-1200® angewendet werden. Dazu ist die spezielle Einleitungsfunktion "MPI6_OpenTcpIp_S71200" auszuführen.
Welche Funktionen für die S7-1200® anwendbar sind, ist bei den einzelnen Funktionen explizit angegeben.

1.3.5 Ab V6.0: Geschwindigkeitsoptimierte Protokolle bei Lese- und Schreibfunktionen

Ab der Version 6 werden beim Lesen und Schreiben von Operanden (E, A, M, T, Z, DB) geschwindigkeitsoptimierte Protokolle verwendet. Dadurch erhöht sich die Kommunikationsgeschwindigkeit gegenüber den früheren Versionen erheblich.

1.3.6 Ab V6.0: Neue Funktionen MixRead und MixWrite

Mit den neuen Funktionen MixRead und MixWrite können verschiedene Operandentypen mit einem Aufruf gelesen oder beschrieben werden. So kann man beispielsweise Daten aus verschiedenen Datenbausteinen, Merker und Eingänge mit einem Aufruf der Funktion MixRead lesen. Dabei werden die Datenzugriffe von der Funktion automatisch optimiert und so beispielsweise doppelte Abfragen, Überschneidungen usw. erkannt.

1.3.7 Ab V6.0: Lese- und Schreibfunktionen ohne Passwortangabe möglich

Ab der Version 6 können die Lese- und Schreibfunktionen (Byte, Wort und Doppelwort) ohne Kenntnis des Passwortes der CPU ausgeführt werden. Dies bedeutet, man kann die Funktionen ausführen ohne dass das Passwort an die CPU zu übergeben ist.

1.3.8 Ab V6.0: DBs in WLD-Dateien schreiben

Ab der Version 6 können DBs aus der CPU ausgelesen und in eine sog. WLD-Datei geschrieben werden. Eine solche Datei kann man mit S7-Programmiersystemen weiterverarbeiten. Ebenso kann damit eine Datensicherung der DBs auf dem PC vorgenommen werden.

1.3.9 Ab V6.0: DBs aus WLD-Dateien lesen und in die CPU übertragen

Ab der Version 6 können DBs aus einer WLD-Datei gelesen und in die CPU übertragen werden. Eine WLD-Datei kann man z.B. mit S7-Programmiersystemen erzeugen. Damit hat man unter anderem die Möglichkeit, DBs mit verschiedenen Einstellungen auf dem PC zu speichern und diese bei Bedarf in die CPU zu übertragen (Rezeptverwaltung und ähnliches).

1.3.10 Ab V6.0: Funktion RAM nach ROM kopieren

Mit dieser Funktion kann man die ablaufrelevanten Daten aller DBs im Arbeitsspeicher der CPU, auf dem Ladespeicher der CPU sichern. Damit bleiben die Daten auch beim Urlöschen der CPU erhalten.

1.3.11 Ab V6.0: Uhrzeit und Datum der CPU lesen/schreiben

Ab der Version 6 kann die Uhrzeit und das Datum einer CPU gelesen und neu beschrieben werden.

1.3.12 Ab V6.0: Betriebsart der CPU umschalten

Ab der Version 6 kann die Betriebsart der CPU in RUN oder STOP umgeschaltet werden.

1.3.13 Ab V5.0: Auslesen der Identifikationsdaten einer CPU (z.B. Seriennummer der CPU)

Ab der Version 5 können die Identifikationsdaten einer CPU ausgelesen werden. Dazu gehören:

- Seriennummer der CPU
- Seriennummer der MMC in der CPU
- Anlagenkennzeichnung (Kann vom Anwender in der Hardwarekonfiguration der CPU angegeben werden)
- Ortskennzeichnung (Kann vom Anwender in der Hardwarekonfiguration der CPU angegeben werden)
- Name der CPU (Kann vom Anwender in der Hardwarekonfiguration der CPU angegeben werden)
- Stationskennzeichnung (Kann vom Anwender in der Hardwarekonfiguration der CPU angegeben werden)

Die Daten können von den S7-300® CPUs von Siemens ab dem Firmwarestand 2.6 ausgelesen werden. In ComDrvS7 ist eine Funktion vorhanden mit welcher ermittelt werden kann, ob eine CPU die Daten liefert.

1.3.14 Ab V5.0: Auslesen des Status der Fehler-LEDs einer CPU

Ab der Version 5 kann der Status der Fehler-LEDs SF (Sammelfehler), BF1 (Bus-Fehler1) und BF2 (Bus-Fehler2) einer CPU ausgelesen werden. Damit ist der PC-Programmierer in der Lage zu ermitteln, ob die Bearbeitung des SPS-Programms durch einen solchen Fehler beeinträchtigt oder sogar unmöglich ist. Ein solcher Fehler kann dann auf dem PC angezeigt werden bzw. das PC-Programm kann in angemessener Weise reagieren.

1.3.15 Ab V5.0: Passwort an eine passwortgeschützte CPU übergeben

Verfügt die CPU über einen Schreibschutz, d.h. der schreibende Zugriff auf die CPU ist nur über das Passwort möglich, so kann ab der Version 5 dieses Passwort an die CPU übergeben werden. Dazu stellt ComDrvS7 zwei Funktion zur Verfügung. Über die erste Funktion kann überprüft werden, ob für einen schreibenden Zugriff ein Passwort erforderlich ist. Die zweite Funktion übergibt dann das Passwort an die CPU (das korrekte Passwort muss natürlich bekannt sein) und schaltet den Zugriff frei. Die Freischaltung gilt dann so lange, bis die Kommunikation zur CPU wieder abgebaut wird.

1.3.16 Ab V5.0: Lesen von DB-Daten aus unterschiedlichen DBs in einem Funktionsaufruf

Über die beiden Funktionen MPI_A_MixReadDBByte und MPI_A_MixReadDBWort ist es möglich, Daten aus verschiedenen Datenbausteinen über einen Funktionsaufruf zu lesen. Damit kann beispielsweise aus dem DB10 das Byte 12 und aus dem DB11 das Byte 0 gelesen werden. Die Funktionen sind dann interessant, wenn die zu lesenden Daten nicht in einem Datenbaustein gesammelt sind.

1.3.17 Ab V4.0: Unterstützung von Routing

Ab der Version 4 wird über die Kommunikationswege NetLink, NETLink PRO, TCP/IP-Direkt und SIMATIC®-NET das Routen unterstützt. Dies bedeutet, man kann auch CPUs ansprechen, welche nicht direkt mit dem PC verbunden, allerdings mit der am PC angeschlossenen CPU vernetzt sind. Der Vorteil besteht darin, dass nicht eine CPU als Datensammler fungieren muss, sondern jede CPU im Verbund angesprochen werden kann. Dabei wird nur eine CPU direkt an den PC angeschlossen, diese leitet dann die Anfrage an die anderen CPUs weiter. Die Anfrage kann dabei auch über verschiedene Bussysteme hinweg zur CPU geroutet werden. Voraussetzung ist, dass bei der Hardwarekonfiguration die Routingdaten in den CPUs abgelegt wurden.

1.3.18 Ab V4.0: Zusätzlicher Kommunikationsweg SIMATIC®-NET

Ab der Version 4 wird SIMATIC®-NET unterstützt, sofern die Treiber auf dem PC installiert sind (ist der Fall wenn z.B. der Simatic® Manager ab V5.1 oder Teleservice ab V6 installiert sind). Damit ist es möglich, beispielsweise die CPs 5512, 5611 sowie den USB-MPI-Adapter von Siemens anzusprechen.

1.3.19 Ab V4.0: Unterstützung von Fernwartungszugriffen über Teleservice von Siemens

Bei Vorhandensein des Teleservice V6 von Siemens, kann ComDrvS7 auch über die Telefonleitung mit einer CPU kommunizieren. Dabei werden auch die Siemens Teleservice II-Adapter unterstützt.

1.3.20 Ab V3.6: Zusätzlicher Kommunikationsweg NETLink PRO (TCP/IP)

Ab der Verion 3.6 kann ComDrvS7 eine CPU auch über den NETLink PRO ansprechen. Die Kommunikation läuft auf der PC-Seite über TCP/IP. Auf der CPU Seite kann der NETLink PRO für MPI oder Profibus-DP zum Einsatz kommen. Dabei werden alle Baudraten bis 12MBaud unterstützt.
Der NETLink PRO hat gegenüber dem NetLink den Vorteil, dass dieser max. 4 PC-Verbindungen verwalten kann. Weiterhin unterstützt der NETLink PRO die automatische Detektion der Baudrate auf MPI oder Profibus-DP.

1.3.21 Ab V3.5: Zusätzlicher Kommunikationsweg TCP/IP-Direkt

Ab der Version 3.5 kann ComDrvS7 auch eine CPU mit Ethernet-CP oder eine CPU mit integrierter Ethernet-Schnittstelle ansprechen. Dazu wurde die Funktion "`MPI_A_Einleitung_TCP_IP_Direct`" implementiert.

1.3.22 Ab V3.x: Keine Beschränkung auf 128 Bytes (64 Worte) pro Lese-Funktion

Ab der Version 3 sind die Lesefunktionen der ComDrvS7-DLL nicht mehr auf 128 Bytes pro Aufruf beschränkt. Dies bedeutet, es können z.B. 200 Merkerbytes mit einem Aufruf angefordert werden.

1.3.23 Ab V3.x: Neue Funktion zum Abfragen des Betriebszustands der CPU

Ab der Version 3 enthält die ComDrvS7-DLL die Funktion MPI_A_IstCPUInRun. Mit dieser Funktion kann ermittelt werden, ob sich die angeschlossene CPU im Betriebszustand RUN befindet.

1.3.24 Ab V3.x: Neue Funktionen zum Wandeln von Real-, DINT und INT-Datentypen

Ab der Version 3 enthält die ComDrvS7-DLL Hilfsfunktionen, um Real, DINT und INT-Zahlen aus BYTE- bzw- WORD-Buffern zu bilden. Des Weiteren können REAL, DINT und INT-Zahlen in BYTE- bzw. WORD-Buffer geschrieben werden.

1.3.25 Ab V3.x: Übergabe von Byte-Buffern an Lese- und Schreibfunktionen, welche auf Byte-Operanden zugreifen

Ab der Version 3 wurden die Status- und Steuern-Buffer der Lese/Schreibfunktionen auf den Datentyp BYTE geändert, sofern die Funktionen auf Byte-Operanden zugreifen. In den Vorgängerversionen waren die Buffer vom Datentyp WORD. Wenn die ComDrvS7-DLL in Anwendungen zum Einsatz kommt, welche mit einer Vorgängerversion erstellt wurden, so muss der Datentyp des übergebenen Buffers geändert werden. Der geringste Aufwand ensteht dabei, wenn vor dem Aufruf der DLL-Funktion der bisherige WORD-Buffer in einen BYTE-Buffer kopiert wird (bei Schreib-Funktionen). Bei Lese-Funktionen kann der BYTE-Buffer nach dem Aufruf der DLL-Funktion, in den bisherigen WORD-Buffer kopiert werden.
Im Allgemeinen dürfte die Umstellung allerdings kein Problem sein.
Die Umstellung wurde notwendig, um die oben angesprochenen Wandlungsfunktionen komfortabler anwenden zu können.

Wichtiger Hinweis:

Wenn auf dem PC noch andere Prozesse laufen, kann es zu Kommunikationsfehlern kommen, da die Antwortzeiten der MPI-Verbindung nicht eingehalten werden. Um dies zu vermeiden, muss die Kommunikation in einem Thread programmiert werden, welcher die Priorität "THREAD_PRIORITY_TIME_CRITICAL" besitzen muss.

2 Unterschiedliche Versionen von ComDrvS7

2.1 Unterschiede der Lite-Version zur Vollversion

Für ComDrvS7 wird eine sog. **Lite-Version** angeboten.
Die Lite-Version ist generell eine Mehrfachlizenz (Entwicklerlizenz). Mit dieser Version kann nur auf Datenbereiche innerhalb von Datenbausteinen zugegriffen werden. Dabei ist sowohl das Lesen als auch das Schreiben von Daten möglich.

Sämtliche Zugriffe auf andere Operandenbereiche (z.B. Merker, Eingänge, Ausgänge usw.) sind in der Lite-Version nicht möglich. Sollten diese Zugriffe programmiert werden, so kehren die Funktionen mit einem entprechenden Fehler zurück (Siehe auch Tabelle mit den Error-Codes).

Alle anderen Funktionen, so z.B. die Wandelfunktionen, die Funktion zum Ermitteln der erreichbaren Teilnehmer usw., sind in der Lite-Version ebenso ausführbar.
Bei den Erläuterungen zu den einzelnen Funktionen ist angegeben, wenn diese in der Lite-Version nicht verfügbar sind oder welche Einschränkungen bei der Lite-Version bestehen.

2.2 Mehrfachlizenz

Die Mehrfachlizenz kann in beliebig vielen eigenen Projekten/Anlagen verwendet werden. Bei der Registrierung wird der Firmennamen des Lizenznehmers angegeben.
Die Lizenz darf von einem Entwickler verwendet werden.

Lesen Sie dazu bitte auch die entsprechenden Angaben im Lizenzvertrag.

2.3 Besonderheit der Micro-Version

Die Micro-Version von ComDrvS7 ermöglich den Zugriff auf Geräte der Familien S7-1200® und LOGO!® von Siemens (ab 0BA7).
Die Micro-Version ist generell eine Mehrfachlizenz (Entwicklerlizenz).

2.4 Kombination der Versionsarten

Ist man im Besitz einer Mehrfachlizenz von ComDrvS7 und möchte zusätzlich auf eine LOGO!® zugreifen, so kann man eine Micro-Version von ComDrvS7 erwerben und damit die Funktionalität der Mehrfachlizenz erweitern.
Dazu wird die Freischaltfunktion "MPI6_ActivateComDrvS7" zwei mal aufgerufen. Sowohl mit der Registriernummer der Mehrfachlizenz als auch mit der Registriernummer der Micro-Version.

2.5 Extended Version

In der Extended-Version von ComDrvS7 sind Funktionen enthalten, mit denen das SPS-Programm in einer S7-300/400 komplett geladen und überschrieben werden kann. Damit kann z.B. ein Backup und Restore von S7-300/400-Systemen realisiert werden.
Weitere Informationen zu den Funktionen der Extended-Version finden Sie im Handbuch "ComDrvS7-V6-Extended-Deutsch.pdf".

2.6 Lizenzvertrag und Nutzungsbedingungen der ComDrvS7-DLL

Lesen Sie nachfolgende Lizenzbedingungen aufmerksam und sorgfältig durch, bevor Sie die "ComDrvS7-DLL" (MPIA32_V**.DLL oder MPIA64_V**.DLL) auf Ihrem Computer einsetzen. Durch Verwendung der Software, bzw. durch Öffnen der Software-Verpackung erklären Sie Ihr ausdrückliches Einverständnis mit den nachstehenden Lizenzbestimmungen. Für den Fall, dass Sie mit diesen Lizenzbedingungen nicht einverstanden sind, dürfen Sie die Software nicht verwenden. In diesem Fall können Sie das Programmpaket unverzüglich nach Erwerb oder Erhalt an den Hersteller zurücksenden und erhalten den Kaufpreis rückerstattet. Die Software wird nicht verkauft sondern lizenziert zum Zwecke der Nutzung. Eigentum erhalten Sie nur am Speichermedium (Diskette oder CD) sowie am Handbuch sowie den sonstigen zugehörigen Schriftdokumenten.

1. Mehrfachlizenz
Mit dem Erwerb der **Mehrfachlizenz** von ComDrvS7 dürfen Sie ComDrvS7 in unbegrenzt vielen Projekten ohne weitere Lizenzgebühren einsetzen. ComDrvS7 darf nicht für sich alleine verkauft werden, sondern nur in Verbindung mit einem Software-Projekt.
Die Seriennummer bzw. Registriernummer darf unter keinen Umständen weitergegeben werden.
Die **Mehrfachlizenz** von ComDrvS7 darf nicht für Softwareprodukte eingesetzt werden, die in direkter Konkurrenz zu Softwareprodukten von MHJ-Software stehen. Für solche Projekte ist ein gesondertes Lizenzmodell mit MHJ-Software auszuhandeln. Dieses Lizenzmodell muss schriftlich in Form eines Vertrags festgelegt werden. ComDrvS7 darf nicht in Projekten vertrieben werden, bei denen der Treiber die Hauptfunktionalität zur Verfügung stellt.
Die **Mehrfachlizenz** berechtigt den Lizenznehmer, ComDrvS7 in beliebig vielen eigenen Projekten einzusetzen. Wird ein Produkt des Lizenznehmers (in welchem ComDrvS7 Bestandteil ist) verkauft und vom Endkunden vervielfältigt, so muss der Endkunde ebenfalls eine Mehrfachlizenz von ComDrvS7 erwerben.
Nur der Lizenznehmer von "ComDrvS7 Mehrfachlizenz" hat das Recht, Kopien von dem Softwareprodukt anzufertigen, in welchem ComDrvS7 verwendet wird.

Die **Mehrfachlizenz** ist eine Lizenz für einen Entwickler.

2. Dauer der Lizenz
Die Einräumung der Lizenz erfolgt zeitlich unbefristet. Die Lizenz verliert automatisch ihre Wirksamkeit, ohne dass es einer Kündigung bedarf, wenn sie gegen irgendeine Bestimmung dieses Vertrages verstoßen. Im Falle der Beendigung sind sie verpflichtet, die Software sowie alle Kopien der Software zu vernichten. Sie können den Lizenzvertrag jederzeit dadurch beenden, dass sie die Software einschließlich aller Kopien vernichten.

3. Begrenzte Garantie
Die Firma MHJ-Software GmbH & Co. KG garantiert für einen Zeitraum von 6 Monaten ab Empfangsdatum, dass die Software, im wesentlichen frei von Material- und Herstellungsfehlern ist und im wesentlichen entsprechend dem begleitenden Produkthandbuch arbeitet. Im Fall einer berechtigten Mängelrüge behält sich die Firma vor, insgesamt drei Nachbesserungen durchzuführen bzw. im Falle des endgültigen Scheiterns der Nachbesserung Wandlung oder Minderung nach Wahl des Anwenders vorzunehmen. Jede weitere Gewährleistung, insbesondere dafür, dass die Software für die Zwecke des Anwenders geeignet ist, sowie für direkte oder indirekt verursachte Schäden (z. B. Gewinnverluste, Betriebsunterbrechung) sowie für Verluste von Daten oder Schäden, die im Zusammenhang mit der Wiederherstellung verloren gegangener Daten entstehen, ist ausdrücklich ausgeschlossen, es sei denn, dass der Firma MHJ-Software GmbH & Co. KG Vorsatz oder grobe Fahrlässigkeit nachgewiesen werden kann. In jedem Fall ist die Haftung der Firma auf den Betrag beschränkt, der für die Software bezahlt wurde.

4. Sonstiges
Dieser Lizenzvertrag unterliegt dem Recht der Bundesrepublik Deutschland Für den Fall, dass Bestimmungen dieses Lizenzvertrages ganz oder teilweise unwirksam sind oder werden, so berührt dies die Wirksamkeit der übrigen Bestimmungen nicht. Die unwirksame Bestimmung ist vielmehr durch eine solche zu ersetzen, die dem Sinn und Zweck der unwirksamen Bestimmung möglichst nahekommen. Nebenabreden sind nicht getroffen. Änderungen dieser Lizenzvereinbarung bedürfen der Schriftform. Gleiches gilt für die Aufhebung dieser Schriftformklausel.

3 Hinweise zum Einsatz von ComDrvS7 in den verschiedenen Programmiersprachen

ComdrvS7 kann mit den Programmiersprachen C++, C#, VB und Delphi genutzt werden. Dabei können folgende Entwicklungsumgebungen zum Einsatz kommen:

- Visual C++ (Microsoft)
- C++ Builder (Borland, Codegear, embarcadero)
- Visual Basic (Microsoft)
- Visual C# (Microsoft)
- Delphi (Borland, Codegear, embarcadero)

Da die einzelnen Entwicklungsumgebungen in unterschiedlichen **Versionsständen** zum Einsatz kommen können, ist strengstens darauf zu achten, dass neben den Sprachen auch die für die entsprechenden Versionsständen vorgesehenen Deklarationen verwendet werden.

3.1 Windows CE

Für die Windows CE-Version des Treibers ist ein Beispiel als Visual C++ 2008-Solution vorhanden. Dieses befindet sich im Verzeichnis "EXAMPLES Windows CE". Die DLLs sind aus dem Ordner "DLL-WinCE" zu entnehmen. Hier sind jeweils Dateien für ARM und x86 vorhanden.

3.2 Visual C++

3.2.1 Was muss beachtet werden?

Es sind die Dateien aus dem Verzeichnis "DLL-Bit32\VC" bzw. "DLL-Bit64\VC" zu verwenden. Darin befinden sich die notwendigen DLL- und LIB-Dateien. Des Weiteren ist die Header-Datei vorhanden, welche in das VC-Projekt einzubinden ist.
Die Dateien können für alle VC-Eintwicklungsumgebungen ab der Version 6 (VC98) verwendet werden. Die 64-Bit DLL kann ab Visual Studio 2012 verwendet werden.

Damit die ComDrvS7-DLL im Projekt verwendet werden kann, ist in den Projekteinstellungen die LIB-Datei einzubinden. Nachfolgend ist dies am Beispiel von VC 2008 zu sehen:

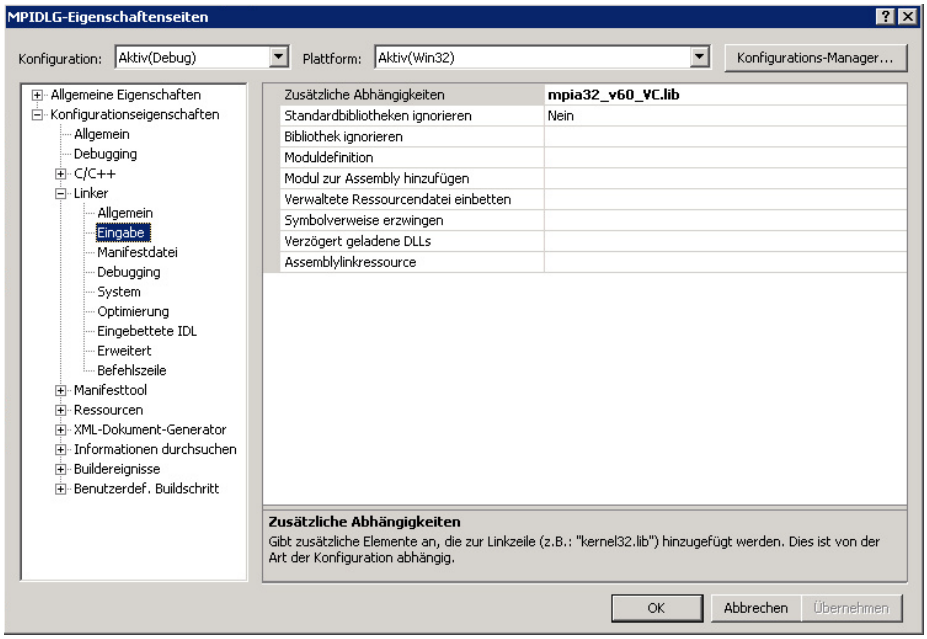


Bild: LIB-Datei in das Projekt von VC einbinden.

Für die Ausführung der Applikation in der Entwicklungsumgebung müssen die DLLs von ComDrvS7 in das Projektverzeichnis kopiert werden.

3.2.2 Beispiele zu VC++

Für VC6 befindet sich das Beispiel im Verzeichnis "Example Visual C V6". Dieses kann auch für die Versionen < VC2008 verwendet werden, dabei wird das Projekt beim Öffnen in die jeweilige Version konvertiert.
Für VC2008 oder höher befindet sich das Beispiel im Verzeichnis "Example Visual C 2008", "Example Visual C 2010", "Example Visual C 2012" usw..

Anmerkung:

Die VC-Beispiele können nicht mit der Express-Version von Visual C verwendet werden, da in der Express-Version die MFC-Klassenbibliothek nicht verfügbar ist.

3.3 C++ Builder

3.3.1 Was muss beachtet werden?

Es sind die Dateien aus dem Verzeichnis "DLL-Bit32\BC" oder "DLL-Bit64\BC" zu verwenden. Darin befinden sich die notwendigen DLL- und LIB-Dateien. Des Weiteren ist die Header-Datei vorhanden, welche in das Builder-Projekt einzubinden ist.
Die Dateien können für alle Builder-Eintwicklungsumgebungen ab der Version 5 verwendet werden.

Im Projekt des Builders wird die LIB-Datei einfach in die Projektgruppe der EXE-Datei eingebunden. Nachfolgend ist dies dargestellt.

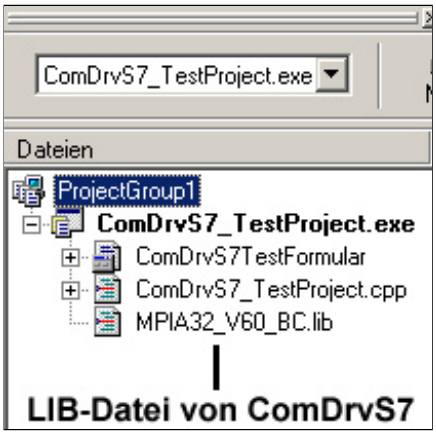


Bild: LIB-Datei wird im EXE-Projekt eingebunden.

Für die Ausführung der Applikation in der Entwicklungsumgebung müssen die DLLs von ComDrvS7 in das Projektverzeichnis kopiert werden.

Bei der Verwendung der 64-Bit DLL ist zu beachten, dass die LIB-Datei die Endung ".a" besitzt. Die 64-Bit DLL kann ab dem C++ Builder XE3 verwendet werden, da dies die erste Builder-Version mit einem 64-Bit Compiler ist.

3.3.2 Beispiele zu C++ Builder

Für den C++Builder befinden sich mehrere Beispiele im Installationsverzeichnis von ComDrvS7. Diese sind nach der Versionnummern des Builders sortiert.
Dabei sind Beispiele für die Builder Version C++Builder 5, C++Builder 2007 und C++Builder 2009 vorhanden. Für Versionen < C++Builder 2007 kann das Projekt C++Builder 5 verwendet werden.
Neuere Version des C++ Builders (>= 2010) können das Beispiel "C++ Builder 2010" verwenden.
Für die 64-Bit DLL ist das Beispiel für C++ Builder XE3 zu verwenden.

3.4 Visual Basic

3.4.1 Was muss beachtet werden?

Es sind die Dateien aus dem Verzeichnis "DLL-Bit32\VB" zu verwenden. Darin befinden sich die notwendigen DLL- und LIB-Dateien.
In diesem Verzeichnis befinden sich ebenfalls die Dateien "ComDrvS7V6_Declare_VB6.bas" und "ComDrvS7_Declare_VB2008.vb". In diesen Dateien sind die Deklarationen der Funktionen von ComDrvS7 angegeben.
Dabei ist die Datei "ComDrvS7_Declare_VB6.bas" nur für die Version 6 von Visual Basic zu verwenden!
Die Datei "ComDrvS7V6_Declare_VB2008.vb" wird ab den Visual Basic Version nach VB6 verwendet.
Nachfolgend ist dargestellt, wie diese Datei in die Projektmappe einzubinden ist:

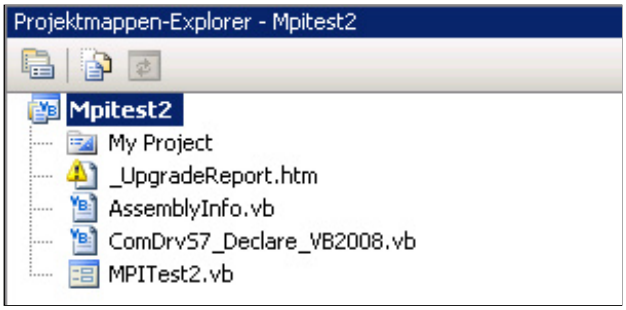


Bild: Eingebundene Datei mit der Deklaration der ComDrvS7-Funktionen.

Für die Ausführung der Applikation in der Entwicklungsumgebung müssen die DLLs von ComDrvS7 in das Windows-System32-Verzeichnis kopiert werden.

Wird die **64-Bit DLL** von ComDrvS7 verwendet, so ist die .Net-Wrapper-Klasse für die Entwicklung einzusetzen. Das Beispiel "Example Visual Basic 2012 64-Bit Wrapper" zeigt dies.

3.4.2 Beispiele zu Visual Basic

Für Visual Basic befinden sich mehrere Beispiele im Installationsverzeichnis von ComDrvS7. Diese sind nach der Versionnummern sortiert.
Das Beispiel "Example Visual Basic 6" kann für die Version VB6 verwendet werden.

Bei VB 2008 wird das Beispiel "Example Visual Basic 2008" verwendet. Für dieses Beispiel kann auch die Express-Version von VB2008 verwendet werden.
Des Weiteren ist das Beispiel "Example Visual Basic 2008 mit Wrapper" vorhanden, wobei die .Net-Wrapper-Klasse zur Anwendung kommt.

Setzen Sie VB 2010 oder höher ein, so können die Projektmappen "Example Visual Basic 2010" bzw. "Example Visual Basic 2010 mit Wrapper" verwendet werden.

Wichtiger Hinweis:
Werden die Programme innerhalb der Eintwicklungsumgebung ausgeführt, also innerhalb von Visual Studio, so müssen die notwendigen DLLs von ComDrvS7 in das jeweiligen BIN-Verzeichnis kopiert werden. Wird z.B. im Konfigurationsmanager des Projektes "Debug" und als Plattform "x86" eingestellt, so sind die DLLs in das Verzeichnis "...\bin\x86\Debug\" zu kopieren.

3.5 Visual C#

Es sind die Dateien aus dem Verzeichnis "DLL-Bit32\VC" oder "DLL-Bit64\VC" zu verwenden. Darin befinden sich die notwendigen DLL-Dateien. In diesem Verzeichnis befindet sich ein weiteres Verzeichnis mit der Bezeichnung "NET". Darin ist die DLL mit der Wrapper-Klasse abgelegt. Diese hat den Namen "ComDrvS7V6_Net.dll". Die DLLs sind in das Projektverzeichnis "..\bin\Debug" zu kopieren.
Die Wrapper-Klasse (bzw. die Datei ComDrvS7V6_Net.dll) ist in den Verweisen der Projektmappe einzufügen. Nachfolgend ist dies zu sehen:

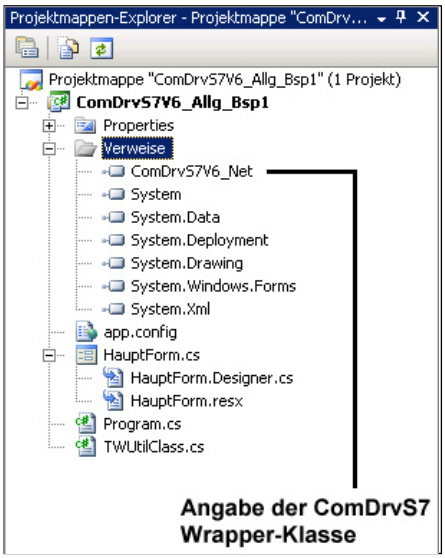


Bild: DLL mit Wrapper-Klasse eingefügt in den Verweisen der Projektmappe

Danach ist folgende using-Direktive anzugeben:

```
using MHJSW.ComDrvS7V6_Net;
```

Für weitere Infos können die umfangreichen Beispiele zu C# angesehen werden.

3.5.1 Beispiele zu Visual C#

Für Visual C# kann das Beispiel im Verzeichnis "Example Visual C# 2008" oder "Example Visual C# 2010" usw. verwendet werden. Diese befinden sich im Installationsverzeichnis von ComDrvS7.
Für die 64-Bit DLL ist das Beispiel "Example Visual C# 2012 64Bit" zu verwenden.

Kommen die PCPanel WPF-Controls zum Einsatz, so kann das Beispiel "Example Visual C# 2008 mit PCPanel WPF" angesehen werden. Die Projektmappe kann ab der Version 2008 von Visual Studio geöffnet werden.

Wichtiger Hinweis:
Werden die Programme innerhalb der Eintwicklungsumgebung ausgeführt, also innerhalb von Visual Studio, so müssen die notwendigen DLLs von ComDrvS7 in das jeweiligen BIN-Verzeichnis kopiert werden. Wird z.B. im Konfigurationsmanager des Projektes "Debug" und als Plattform "x86" eingestellt, so sind die DLLs in das Verzeichnis "...\bin\x86\Debug\" zu kopieren.

3.6 Delphi

3.6.1 Was muss beachtet werden?

Es sind die Dateien aus dem Verzeichnis "DLL-Bit32\BC" oder "DLL-Bit64\BC" zu verwenden. Darin befinden sich die notwendigen DLL- und LIB-Dateien (bzw. ".a" bei 64 Bit).
Des Weiteren befinden sich darin die beiden Dateien
"DelphiDeklarationFunktionen_V4_Bis_2006.TXT" und
"DelphiDeklarationFunktionen_Ab_Delphi2009.TXT".
In diesen Dateien sind die Deklarationen aller Funktionen von ComDrvS7 aufgeführt. Je nach der verwendeten Delphi-Version können die Daten kopiert und in den Sourcecode übernommen werden. Die Deklarationen sind auch in den jeweiligen Beispielen zu Delphi vorhanden.

Die ComdrvS7-DLL-Dateien müssen in das Projektverzeichnis kopiert werden.

3.6.2 Beispiele zu Delphi

Es sind mehrere Beispiele zu verschiedenen Delphi-Versionen vorhanden. Dabei geben die Verzeichnisnamen an, für welche Delphi-Version die Beispiele vorgesehen sind.
Es stehen Beispiele für die Version 4, 2006 und 2009 zur Verfügung. Kommt eine Delphi-Version zum Einsatz, welche zwischen der Version 4 und 2006 liegt, so lädt man das Beispiel für die Version 4, dieses wird dann entsprechend konvertiert.

Ab Version XE5, kommt das Beispiel zu XE5 zur Anwendung. Die Deklaration der Treiberfunktionen ist dabei in der Datei "ComDrvS7Functions.pas" enthalten. Hier sind sowohl die 32Bit- als auch die 64Bit-Funktionen deklariert.

4 Umstieg von älteren Versionen des ComDrvS7

Haben Sie eine Applikation mit einer der älteren Versionen von ComDrvS7 erstellt, so können Sie sehr einfach und schnell von den Neuerungen der Version 6 profitieren.

Die Lese- und Schreibfunktionen der Version 6 wurden erheblich geschwindigkeitsoptimiert. Des Weiteren können die neuen Read- und Write-Funktionen auf passwortgeschützte CPUs zugreifen, ohne dass das Passwort übergeben werden muss.
Grund genug auf die neuen Lese- und Schreibfunktionen umzustellen. Diese Umstellung kann sehr einfach durchgeführt werden.

Dabei geht man wie folgt vor:

- Aufruf der gewohnten Einleitungsfunktion.
- Aufruf der Funktion **MPI6_ChangeProtocolTypeForV5Functions**, wobei der Parameter TakeV6Protocol den Wert 1 haben muss.

Nun werden alle alten Lese- und Schreibfunktionen (z.B. MPI_A_ReadMerkerByte, MPI_A_WriteMerkerByte usw.) auf den neuen Protokolltyp umgestellt, ohne dass weitere Codeänderungen notwendig sind.

Die Funktion **MPI6_ChangeProtocolTypeForV5Functions** muss dabei nur einmalig nach der Einleitungsfunktion aufgerufen werden. Die Einstellung gilt dann bis zur Beendigung der Kommunikation.

Eine genauere Beschreibung der Funktion **MPI6_ChangeProtocolTypeForV5Functions** finden Sie in einem gesonderten Kapitel.

Manche alten Funktionen werden in diesem Handbuch nicht mehr erwähnt. Diese sind aber weiterhin aus Gründen der Kompatibilität in ComDrvS7 enthalten. Bei den Auskunftsfunktionen wurden lediglich die Funktionamen und Parameter in englisch übersetzt. Die Beschreibungen aus diesem Handbuch gelten dabei auf für diese Funktionen.

Für die Beschreibung der Funktionen bis ComDrvS7 V5, kann das bisherige Handbuch verwendet werden, dieses wird auch bei der Installation des ComDrvS7 V6 als PDF-Datei mit installiert.

5 Grundsätzliche Vorgehensweise bei ComDrvS7

Die nachfolgende Grafik zeigt die grundsätzliche Vorgehensweise bei der Verwendung des ComDrvS7.

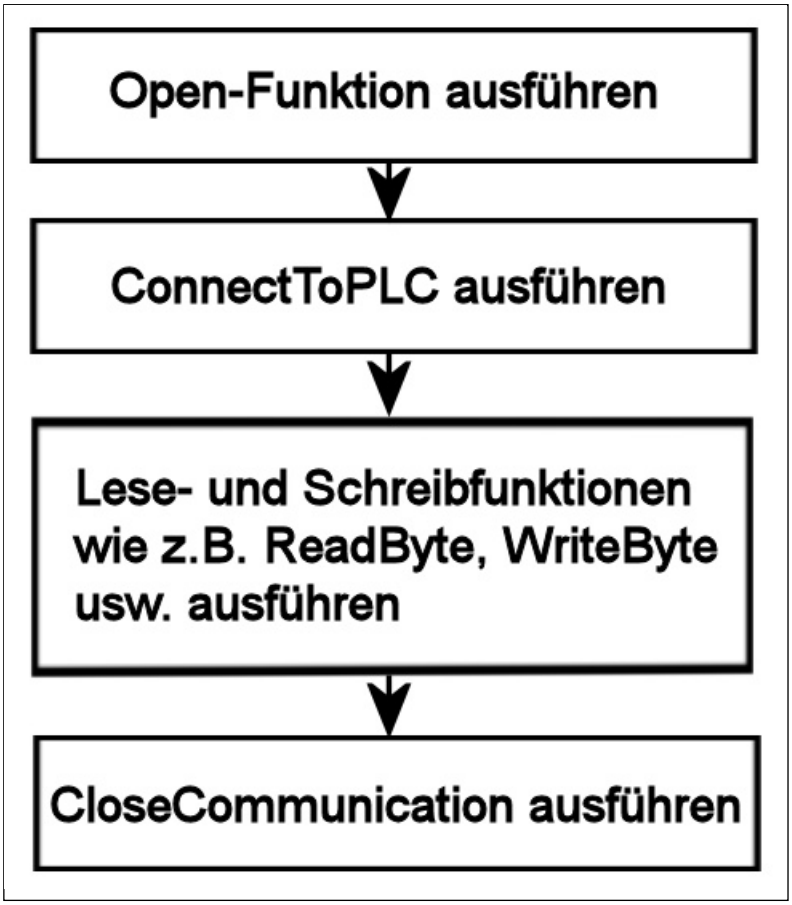


Bild: Vorgehensweise bei ComDrvS7

Zunächst wird je nach Verbindung zur CPU eine der 6 **MPI6_Open**-Funktionen aufgerufen.
Danach erfolgt der Aufruf der Funktion **MPI6_ConnectToPLC**.
Ist diese Funktion ohne Fehler ausgeführt worden, so kann man über die **Lese-, Schreibe- und Auskunftsfunktionen** auf die CPU zugreifen.
Werden die Daten zyklisch mit der CPU ausgetauscht, so kann man die Verbindung offen lassen und immer wieder auf die CPU zugreifen.
Erst beim Beenden der Kommunikation (oder im Fehlerfall) wird die Funktion **MPI6_CloseCommunication** aufgerufen um die Verbindung zur CPU abzubauen und die Instanz von ComDrvS7 freizugeben.

5.1 Umstellung auf einen anderen Zugangsweg zur CPU

Haben Sie eine Applikation entwickelt bei der z.B. die Verbindung zur CPU über einen MHJ-NetLink realisiert wurde und kommt in dem neuen Projekt eine direkte TCP/IP Verbindung zur CPU zum Einsatz, so sind die Änderungen innerhalb Ihres Codes minimal.

In ComDrvS7 wird der Zugangsweg zur CPU über die Open-Funktion festgelegt. Nach der Open-Funktion gibt es bzgl. dem Zugangsweg keine Unterschiede mehr.

Wenn also z.B. vom MHJ-NetLink auf TCP/IP-Direkt umgestellt werden muss, so ist anstatt der Funktion "MPI6_OpenNetLink" die Funktion "MPI6_OpenTcpIp" aufzurufen. Ansonsten sind keine Änderungen notwendig.
Dies gilt auch für alle anderen Zugangswege.

5.2 Verhalten bei einem Fehler in den Open-Funktionen

Passiert während der Bearbeitung der Open-Funktionen ein Fehler, so liegt meist ein Hardwarefehler vor oder die an die jeweilige Open-Funktion übergebenen Parameter sind nicht korrekt.
Kehrt eine Open-Funktion mit einem Fehler zurück, so muss keine weitere Funktion aufgerufen werden, die Kommunikations-Instanz wird bei einem Fehler in der Open-Funktion bereits beseitigt.

5.3 Verhalten bei einem Fehler nach den Open-Funktionen

Tritt bei der Funktion MPI6_ConnectToPLC oder einer der Lese-/Schreibfunktionen zur CPU hin (z.B. MPI6_ReadByte, MPI6_WriteDword usw.) ein Fehler auf, so sollte generell mit der Funktion MPI6_CloseCommunication die Kommunikationsinstanz geschlossen und danach die jeweilige Open-Funktion neu aufgerufen werden.

Wichtig dabei ist, dass die Funktion MPI6_CloseCommunication aufgerufen wird, da ansonsten die Kommunikationsinstanz im Speicher verbleibt!

Anmerkung für Fehler die kein Kommunikationsproblem als Ursache haben

Eine Ausnahme bilden die Fehler, welche aufgrund von fehlerhaften Parameterübergaben usw. auftreten. Ein Bespiel ist der Fehler, welcher auftritt, wenn bei eine Aktion mit einer WLD-Datei, diese Datei nicht an dem angegeben Pfad vorhanden ist. In diesem Fall genügt es natürlich den entsprechenden Pfad zu ändern ohne dass die Funktion MPI6_CloseCommunication aufgerufen werden muss.
Diese Fehler treten meist nur in der Entwicklungsphase der Applikation auf.

Eine Liste der Error-Codes finden Sie im Kapitel "Fehlermeldungen".

6 Beschreibung der einzelnen Funktionen

In diesem Kapitel werden die einzelnen Funktionen des ComDrvS7 mit deren Aufgaben erläutert. Nach den meisten Beschreibungen folgt ein kleines Beispiel, in welchem die jeweilige Funktion verwendet wird.

6.1 Grundsätzliches zur Erläuterung der einzelnen Funktionen von ComDrvS7

Die einzelnen Funktionen werden in der C-Syntax dargestellt. Da die Beispiele sehr einfach gehalten sind, dürfte dies aber auch für VB, Delphi und C#-Programmierer kein Problem darstellen. Die Erläuterungen für die Parameter sind für alle Programmiersprachen gleichbedeutend. Abweichungen werden dabei bei der Erklärung des jeweiligen Parameters angegeben.

Bei der Verwendung der Wrapper-Klasse unter .Net entfällt generell die Angabe des Handles für die Kommunikationsinstanz, da dieses Handle in der Wrapper-Klasse verwaltet wird.

6.2 Die Funktion: MPI_A_GetDLLError bzw. MPI_A_GetDLLErrorEng

Kurzbeschreibung

Die Funktion MPI_A_GetDLLError kann aufgerufen werden um einen String (nullterminiert) zu erhalten, welcher den übergebenen Error näher beschreibt. Jede Funktion in der DLL liefert den Wert '0' (FALSE) als Funktionswert zurück, wenn ein Fehler bei der Ausführung aufgetreten ist. In diesem Fall enthält die Variable ErrorCode, welche jeder Funktion übergeben werden muss, den Fehlercode.
Will man diesen Fehler beschreiben, so kann die Funktion MPI_A_GetDLLError verwendet werden.
Eine Beschreibung der Errorcodes ist im Abschnitt "Fehlermeldungen" zu finden.
Die Funktion MPI_A_GetDLLErrorEng liefert die Fehlermeldungen in Englisch.
In der Wrapper-Klasse für .Net haben die Funktionen die Namen "MPI6_GetDLLError" bzw. "MPI6_GetDLLErrorEng"

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird. (Entfällt bei .Net-Wrapper-Klasse)
ErrorString	CHAR*	Enthält den String für den übergebenen Errorcode.
ErrorCode	WORD	Error-Code für den der String geliefert werden soll.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

In den Beispielen zu den einzelnen Funktionen, wird MPI_A_GetDLLError häufig verwendet.

6.3 Die Funktion: MPI6_OpenRS232

Kurzbeschreibung

Die Funktion MPI6_OpenRS232 muss zwingend aufgerufen werden, um mit einer CPU zum ersten Mal zu kommunizieren, sofern die Kommunikation über eine **serielle Schnittstelle** bzw. **einen virtuellen COM-Port eines USB-Adapters** aufzubauen ist.
Die Funktion öffnet die angegebene PC-Schnittstelle mit den ebenfalls angegebenen Kommunikationsparametern. Des Weiteren sind der Funktion MPI-Netzdaten zu übergeben. Die Funktion kann nur erfolgreich ausgeführt werden, wenn eine CPU an der angegebenen PC-Schnittstelle angeschlossen ist.

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (COM-Schnittstelle, Baudrate usw.) genutzt wird. Das Handle spezifiziert somit die Kommunikationsinstanz (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
ComNr	INT	Angabe der seriellen PC-Schnittstelle, an welcher das AG angeschlossen ist. Z.B. '1' für COM1
BaudRate	LONG	Angabe der Baudrate für die Kommunikation mit dem AG. Dieser Wert ist abhängig von dem verwendeten PC/MPI-Kabel. Die älteren Modelle unterstützen als Baudrate nur 19200 Baud. Bei neueren Kabeln läßt sich die Baudrate auf die Werte 19200, 38400, 57600 und 115200 Baud einstellen. **Hier kann übergeben werden: 19200, 38400, 57600,115200 Wird dieser Wert falsch angegeben, so kann keine Kommunikation aufgebaut werden.**
PGAddress	BYTE	Angabe der MPI/DP-Adresse, mit welcher sich die Kommunikationsinstanz am MPI/DP-Netz anmelden soll. **Es ist zu beachten, dass die angegebene Adresse von keinem anderen Gerät im angeschlossenen Netz verwendet werden darf.** Standardmäßig sind die Programmiergeräte auf die Adresse '0' eingestellt.
HighestAddress	BYTE	Angabe der höchsten Adresse, welche im angeschlossenen MPI-Netz verwendet werden darf. Dabei sind die Werte **15, 31, 63 oder 126** anzugeben. Es ist darauf zu achten, dass alle Geräte im angeschlossenen Netz die gleiche höchste Adresse besitzen.

ComWasAlreadyUsed	BOOL*	In diesem Parameter wird der Wert TRUE geliefert, wenn die Schnittstelle bereits von einer anderen Kommunikationsinstanz genutzt wird. In diesem Fall wurden die in der Funktion übergebenen Kommunikationsparameter (z.B. Baudrate und PGAddress) nicht berücksichtigt. Die Kommunikation kann aber erfolgen. Wird in dem Parameter der Wert FALSE geliefert, so war die Schnittstelle noch frei und die angegebenen Kommunikationsparameter wurden eingestellt.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_OpenRS232 die Schnittstelle COM2 geöffnet. Dabei wurde eine Übertragungsrate von 115200Baud gewählt.

```
//Variablen
int ComNr=2;          //Schnittstelle COM2
long BaudRate=115200; //Baudrate 115200
BYTE PGMPIAdresse=0;  //Adresse der DLL-Aplikation = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte Adresse im Netz = 31
bool SchnittstelleWarSchonAllokiert=false; //true wenn die
                                           //Schnittselle schon
                                           //genutzt wird
WORD Error=0;         //Error-Variable
char ErrorString[255];//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz

//Verbindung aufbauen
if (!MPI6_OpenRS232(&MPIHandle, ComNr, BaudRate, PGMPIAdresse,
                            HoechsteMPI,
                            &SchnittstelleWarSchonAllokiert,
                            &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung war erfolgreich.", "",
           MB_ICONINFORMATION);
```

6.4 Die Funktion: MPI6_OpenNetLink

Kurzbeschreibung

Die Funktion MPI6_OpenNetLink muss zwingend aufgerufen werden, um mit einer CPU zum ersten Mal zu kommunizieren, sofern die **Kommunikation über TCP/IP mit Hilfe eines MHJ-NetLink** bzw. **MHJ-NetLink++** aufzubauen ist.
Die Funktion baut eine Verbindung zu einem NetLink mit der angegebenen IP-Adresse auf. Des Weiteren sind der Funktion MPI/DP-Netzdaten zu übergeben.

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddress	CHAR*	Übergabe der IP-Adresse des NetLink, mit dem die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
PGAddress	BYTE	Angabe der MPI/DP-Adresse, mit welcher sich die Kommunikationsinstanz am Netz anmelden soll. **Es ist zu beachten, dass die angegebene Adresse von keinem anderen Gerät im angeschlossenen Netz verwendet werden darf.** Standardmäßig sind die Programmiergeräte auf die Adresse '0' eingestellt. **Achtung:** Die PG-Adresse des MHJ-NetLink kann nicht über diesen Funktionsaufruf verändert werden. Diese ist mit dem mitgelieferten Konfigurationsprogramm einzustellen.
HighestAddress	BYTE	Angabe der höchsten MPI/DP-Adresse, welche im angeschlossenen Netz verwendet werden darf. Dabei sind die Werte **15, 31, 63 oder 126** anzugeben. Es ist darauf zu achten, dass alle Geräte im angeschlossenen Netz die gleiche höchste Adresse besitzen. **Achtung:** Die höchste Adresse des MHJ-NetLink kann nicht über diesen Funktionsaufruf verändert werden. Diese ist mit dem mitgelieferten Konfigurationsprogramm einzustellen.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_OpenNetLink eine Verbindung zu einem NetLink mit der IP-Adresse 172.16.130.84 aufgebaut.

```
BYTE PGMPIAdresse=0;  //MPI-Adresse der Kommunikationsinstanz = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_OpenNetLink(&MPIHandle, IPAdresseStr, PGMPIAdresse,
                              HoechsteMPI, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

Anmerkung:
Es sei nochmals erwähnt, dass die Angaben der PG-Adresse und der höchsten MPI/DP-Adresse nicht die im NetLink eingestellten Werte verändern. Die Einstellung im NetLink werden über das mitgelieferte Konfigurationsprogramm getätigt. Die Einstellungen müssen nur einmalig vorgenommen werden, danach sind die Daten fest im MHJ-NetLink abgelegt.

6.5 Die Funktion: MPI6_OpenTcpIp

Kurzbeschreibung

Die Funktion MPI6_OpenTcpIp muss zwingend aufgerufen werden, um mit einer CPU zum ersten Mal zu kommunizieren, sofern die **Kommunikation über TCP/IP hin zu einem Ethernet-CP oder einer CPU mit integrierter Ethernet-Schnittstelle** aufzubauen ist.
Die Funktion baut eine Verbindung zu einem Ethernet-CP mit der angegebenen IP-Adresse auf. Des Weiteren ist die Slot-Nummer der CPU zu übergeben. Bei S7-300-Systemen ist hierbei generell der Slot 2 anzugeben.

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddress	CHAR*	Übergabe der IP-Adresse des Ethernet-CPs oder der integrierten Ethernet-Schnittstelle der CPU, mit dem die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
PlcSlotNr	INT	Angabe des Steckplatzes der CPU mit welcher kommuniziert werden soll. Bei S7-300-Systemen ist hierbei generell der Steckplatz 2 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_OpenTcpIp eine Verbindung zu einem Ethernet-CP mit der IP-Adresse 172.16.130.84 aufgebaut.

```
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
int CPUSlotNr=2; //Steckplatz der anzusprechenden CPU im SPS-Rack
char IPAdresseStr[50]={0};
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_OpenTcpIp(&MPIHandle, IPAdresseStr,
                    CPUSlotNr, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

6.6 Die Funktion: MPI6_OpenTcpIp_S71500

Kurzbeschreibung

Die Funktion MPI6_OpenTcpIp_S71500 muss zwingend aufgerufen werden, um mit einer CPU der Familie S7-1500® zum ersten Mal zu kommunizieren

Die Funktion baut eine Verbindung zu einer S7-1500® mit der angegebenen IP-Adresse auf.

Bei der Verwendung der S7-1500® von Siemens sind die im Kapitel "Notwendige Einstellung in der Hardwarekonfiguration einer S7-1500® von Siemens" angegebenen Hinweise zu beachten!

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddress	CHAR*	Übergabe der IP-Adresse der Ethernet-Schnittstelle der CPU, mit der die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_OpenTcpIp_S71500 eine Verbindung zu einer S7-1500® CPU mit der IP-Adresse 172.16.130.84 aufgebaut.

```
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_OpenTcpIp_S71500(&MPIHandle, IPAdresseStr, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
          MB_ICONINFORMATION);
```

6.7 Die Funktion: MPI6_OpenTcpIp_S71500Ext

Kurzbeschreibung

Die Funktion MPI6_OpenTcpIp_S71500Ext muss zwingend aufgerufen werden, um mit einer CPU der Familie S7-1500® zum ersten Mal zu kommunizieren

Die Funktion baut eine Verbindung zu einer S7-1500® mit der angegebenen IP-Adresse auf. Zusätzlich ist die IP-Adresse der Netzwerkkarte anzugeben, an welcher die CPU angeschlossen ist. Dies ist notwendig, wenn sich mehrere Netzwerkkarten im PC befinden oder der Zugang z.B. über eine VPN-Verbindung hergestellt wird.

Bei der Verwendung der S7-1500® von Siemens sind die im Kapitel "Notwendige Einstellung in der Hardwarekonfiguration einer S7-1500® von Siemens" angegebenen Hinweise zu beachten!

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddress	CHAR*	Übergabe der IP-Adresse der Ethernet-Schnittstelle der CPU, mit der die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
IPAddressNetworkAdapter	CHAR*	Übergabe der IP-Adresse der Netzwerkkarte des PCs, welche mit der CPU verbunden ist.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

6.8 Die Funktion: MPI6_OpenTcpIp_S71200 (Auch in Micro-Version)

Kurzbeschreibung

Die Funktion MPI6_OpenTcpIp_S71200 muss zwingend aufgerufen werden, um mit einer CPU der Familie S7-1200® zum ersten Mal zu kommunizieren.
Ab **Firmwareversion 4** muss die Hardwarekonfiguration der CPU wie im Kapitel "Notwendige Einstellung in der Hardwarekonfiguration einer S7-1500® und S7-1200® (ab Firmware V4) von Siemens" beschrieben, vorgenommen werden.

Die Funktion baut eine Verbindung zu einer S7-1200® mit der angegebenen IP-Adresse auf.

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddress	CHAR*	Übergabe der IP-Adresse der Ethernet-Schnittstelle der CPU, mit dem die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_OpenTcpIp_S71200 eine Verbindung zu einer S7-1200® CPU mit der IP-Adresse 172.16.130.84 aufgebaut.

```
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_OpenTcpIp_S71200(&MPIHandle, IPAdresseStr, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
          MB_ICONINFORMATION);
```

6.9 Die Funktion: MPI6_OpenTcpIp_Logo (Nur in Micro-Version)

Kurzbeschreibung

Die Funktion MPI6_OpenTcpIp_Logo muss zwingend aufgerufen werden, um mit einer CPU der Familie LOGO!® (ab 0BA7) von Siemens zum ersten Mal zu kommunizieren

Die Funktion baut eine Verbindung zu einer LOGO!® (ab 0BA7) mit der angegebenen IP-Adresse auf. **Wie die LOGO!® für die Ethernetkommunikation zu parametrieren ist, wird im Kapitel "Ethernet-Verbindung in LOGO-Programmiersoftware parametrieren" beschrieben. Beachten Sie dabei auch die Besonderheiten ab einer LOGO!® ab 0BA8**

Mit dieser Funktion wird auch eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddress	CHAR*	Übergabe der IP-Adresse integrierten Ethernet-Schnittstelle des LOGO, mit dem die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_OpenTcpIp_Logo eine Verbindung zu einer LOGO!® mit der IP-Adresse 172.16.130.84 aufgebaut.

```
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_OpenTcpIp_Logo(&MPIHandle, IPAdresseStr, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

6.10 Die Funktion: MPI6_Open_NetLinkPro_TCP_AutoBaud

Kurzbeschreibung

Die Funktion **MPI6_Open_NetLinkPro_TCP_AutoBaud** muss zwingend aufgerufen werden, um mit einer CPU zum ersten Mal zu kommunizieren, sofern die **Kommunikation über einen NETLink PRO hin zu einer MPI oder Profibus-DP-Schnittstelle einer CPU** aufzubauen ist. Die Funktion baut eine Verbindung zu einem NETLink PRO mit der angegebenen IP-Adresse auf. Des Weiteren ist die PG-Adresse sowie die höchste im Netz (MPI oder Profibus) vorhandene Adresse anzugeben. Im Gegensatz zur Funktion **MPI6_Open_NetLinkPro_TCP_SelectBaud** kommt diese Funktion zum Einsatz, wenn der NETLink PRO selbständig die im Bus vorhandene Baudrate detektieren soll. Es ist zu beachten, dass hierbei der Kommunikationsaufbau ein paar Sekunden länger dauert, da die Baudraten-Detektion einige Zeit beansprucht. Wird die Kommunikation häufig auf- und wieder abgebaut (z.B. weil mit vielen verschiedenen CPUs kommuniziert wird), dann sollte die Funktion **MPI6_Open_NetLinkPro_TCP_SelectBaud** verwendet werden.

Mit der Funktion **MPI6_Open_NetLinkPro_TCP_AutoBaud** wird eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddressStr	CHAR*	Übergabe der IP-Adresse des NETLink PRO, mit dem die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
PGAddress	BYTE	Angabe der MPI/DP-Adresse, mit welcher sich die Kommunikationsinstanz am MPI/DP-Netz anmelden soll. **Es ist zu beachten, dass die angegebene MPI/DP-Adresse von keinem anderen Gerät im angeschlossenen MPI/DP-Netz verwendet werden darf.** Standardmäßig sind die Programmiergeräte auf die MPI/DP-Adresse '0' eingestellt.
HighestAddress	BYTE	Angabe der höchsten MPI/DP-Adresse, welche im angeschlossenen Netz verwendet werden darf. Dabei sind die Werte **15, 31, 63 oder 126** anzugeben. Es ist darauf zu achten, dass alle Geräte im angeschlossenen MPI/DP-Netz die gleiche höchste MPI/DP-Adresse besitzen.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_Open_NetLinkPro_TCP_AutoBaud eine Verbindung zu einem NETLink PRO mit der IP-Adresse 172.16.130.84 aufgebaut.

```
BYTE PGMPIAdresse=0;  //MPI-Adresse der Kommunikationsinstanz = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_Open_NetLinkPro_TCP_AutoBaud(&MPIHandle, IPAdresseStr,
                                       PGMPIAdresse, HoechsteMPI,
                                       &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

Anmerkung:

Vor dem ersten Einsatz des NETLink PRO muss dieser einmalig auf die notwendigen Kommunikationsparametern eingestellt werden. Zu diesem Zweck kommt eine Konfigurationssoftware zum Einsatz, welche mit ComDrvS7 geliefert wird. Diese Software befindet sich nach der Installation von ComDrvS7 im Verzeichnis "NETLink PRO Konfigurator". Wurden die Einstellungen (wie z.B. IP-Adresse, Subnetzmaske usw.) vorgenommen, so werden diese fest im NETLink PRO gespeichert. Die Einstellungen sind also auch nach der Abschaltung der Versorgungsspannung des NETLink PRO weiterhin verfügbar.

Für den Betrieb des NETLink PRO mit ComDrvS7 ist kein weiterer Treiber notwendig!

6.11 Die Funktion: MPI6_Open_NetLinkPro_TCP_SelectBaud

Kurzbeschreibung

Die Funktion **MPI6_Open_NetLinkPro_TCP_SelectBaud** muss zwingend aufgerufen werden, um mit einer CPU zum ersten Mal zu kommunizieren, sofern die **Kommunikation über einen NETLink PRO hin zu einer MPI oder Profibus-DP-Schnittstelle einer CPU** aufzubauen ist. Die Funktion baut eine Verbindung zu einem NETLink PRO mit der angegebenen IP-Adresse auf. Des Weiteren ist die PG-Adresse sowie die höchste im Netz (MPI oder Profibus) vorhandene Adresse anzugeben. Im Gegensatz zur Funktion **MPI6_Open_NetLinkPro_TCP_AutoBaud** kommt diese Funktion zum Einsatz, wenn die Baudrate des MPI/DP-Bus bekannt ist. In diesem Fall entfällt die Zeit, welche für das Detektieren der Baudrate benötigt wird, was den Einleitungsvorgang beschleunigt. Wird die Kommunikation häufig auf- und wieder abgebaut (z.B. weil mit vielen verschiedenen CPUs kommuniziert wird), dann sollte diese Funktion verwendet werden. Voraussetzung ist allerdings, dass die Baudrate des MPI/DP-Bus bekannt ist. Dies sollte aber in den häufigsten Fällen gegeben sein.

Mit der Funktion **MPI6_Open_NetLinkPro_TCP_SelectBaud** wird eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg (IP-Adresse) genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
IPAddressStr	CHAR*	Übergabe der IP-Adresse des NETLink PRO, mit dem die Kommunikation ausgeführt werden soll. Die Adresse ist in der Form "172.16.130.84" anzugeben.
PGAddress	BYTE	Angabe der MPI/DP-Adresse, mit welcher sich die Kommunikationsinstanz am MPI/DP-Netz anmelden soll. **Es ist zu beachten, dass die angegebene MPI/DP-Adresse von keinem anderen Gerät im angeschlossenen MPI/DP-Netz verwendet werden darf.** Standardmäßig sind die Programmiergeräte auf die MPI/DP-Adresse '0' eingestellt.
HighestAddress	BYTE	Angabe der höchsten MPI/DP-Adresse, welche im angeschlossenen MPI/DP-Netz verwendet werden darf. Dabei sind die Werte **15, 31, 63 oder 126** anzugeben. Es ist darauf zu achten, dass alle Geräte im angeschlossenen Netz die gleiche höchste MPI/DP-Adresse besitzen.
IsProfibusDP	BOOL	Wenn die Kommunikation mit der CPU über Profibus-DP erfolgt, dann ist hierbei der Wert '1' (TRUE) zu übergeben. Bei einer Kommunikation über MPI ist der Wert '0' (FALSE) zu übergeben.

BaudrateUsed	WORD	Übergabe der im MPI/DP-Netz eingestellten Baudrate. Dabei sind folgende Werte definiert: 9,6 kBaud: MPIA_BAUD_96 entspricht Wert 0 19,2 kBaud: MPIA_BAUD_19_2 entspricht Wert 1 45,45 kBaud: MPIA_BAUD_45_45 entspricht Wert 2 93,74 kBaud: MPIA_BAUD_93_75 entspricht Wert 3 187,5 kBaud: MPIA_BAUD_187_5 entspricht Wert 4 500 kBaud: MPIA_BAUD_500 entspricht Wert 5 1500kBaud: MPIA_BAUD_1500 entspricht Wert 6 3000 kBaud: MPIA_BAUD_3000 entspricht Wert 7 6000 kBaud: MPIA_BAUD_6000 entspricht Wert 8 12000 kBaud: MPIA_BAUD_12000 entspricht Wert 9
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird mit der Funktion MPI6_Open_NetLinkPro_TCP_SelectBaud eine Verbindung zu einem NETLink PRO mit der IP-Adresse 172.16.130.84 aufgebaut. Der NETLink PRO ist dabei mit der Profibus-DP-Schnittstelle der CPU verbunden. Das DP-Netz ist auf 1,5MBaud eingestellt.

```
BYTE PGMPIAdresse=0;  //DP-Adresse der Kommunikationsinstanz = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte DP-Adresse im Netz = 31
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
bool IstProfibusDP=true;//Es ist ein DP-Netz
WORD VorgabeBaudrate=MPIA_BAUD_1500;//Baudrate ist auf 1,5MBaud eingestellt
//IP-Adresse eintragen
strcpy(IPAdresseStr, "172.16.130.84");
//Verbindung aufbauen
if (!MPI6_Open_NetLinkPro_TCP_SelectBaud(&MPIHandle, IPAdresseStr,
                                         PGMPIAdresse, HoechsteMPI,
                                           IstProfibusDP, VorgabeBaudrate
                                          &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

Anmerkung:

Vor dem ersten Einsatz des NETLink PRO muss dieser einmalig auf die notwendigen Kommunikationsparametern eingestellt werden. Zu diesem Zweck kommt eine Konfigurationssoftware zum Einsatz, welche mit ComDrvS7 geliefert wird. Diese Software befindet sich nach der Installation von ComDrvS7 im Verzeichnis "NETLink PRO Konfigurator". Wurden die Einstellungen (wie z.B. IP-Adresse, Subnetzmaske usw.) vorgenommen, so werden diese fest im NETLink PRO gespeichert. Die Einstellungen sind also auch nach der Abschaltung der Versorgungsspannung des NETLink PRO weiterhin verfügbar.

Für den Betrieb des NETLink PRO mit ComDrvS7 ist kein weiterer Treiber notwendig!

6.12 Die Funktion: MPI6_Open_SimaticNet (Nicht bei 64-Bit und CE)

Kurzbeschreibung

Die Funktion **MPI6_Open_SimaticNet** muss zwingend aufgerufen werden, um mit einer CPU zum ersten Mal zu kommunizieren, sofern die **Kommunikation über den SIMATIC® NET Treiber erfolgen soll.**
Über diesen Kommunikationsweg können der USB-MPI-Adapter von Siemens, sowie die CPs 5511, 5612 usw. angesprochen werden. Auch die Kommunikation über einen TS-Adapter II ist möglich.

Voraussetzung ist, dass der SIMATIC® NET Treiber auf dem PC installiert ist. Dies ist z.B. der Fall, wenn auf dem PC die Software Simatic®-Manager (ab V5.1), der Treiber des SIEMENS-USB-Adapters oder der Teleservice ab V6 installiert sind. Das dabei verwendete Interface ist auf dem Dialog "PG/PC-Schnittstelle einstellen" zu selektieren. Dieser Dialog kann über die Datei "s7epatsx.exe" im System32-Verzeichnis von Windows aufgerufen werden. Kommt der Teleservice ab V6 von Siemens zum Einsatz, dann kann mit Hilfe eines TS-Adapter II eine Fernabfrage über die Telefonleitung realisiert werden.

Die Funktion **MPI6_Open_SimaticNet** baut eine Verbindung zu dem im Dialog "PG/PC-Schnittstelle einstellen" selektierten Gerät auf.

Mit der Funktion **MPI6_Open_SimaticNet** wird eine Kommunikationsinstanz angelegt. Die "Kennung" dieser Instanz wird in der Variablen "Handle" geliefert. Diese Kennung muss den anderen Funktionen der DLL übergeben werden, damit der hier angegebene Kommunikationsweg genutzt wird (entfällt bei .Net-Wrapper-Klasse).

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT*	Hier wird das Handle der neu gebildeten Kommunikationsinstanz zurückgeliefert (entfällt bei .Net-Wrapper-Klasse).
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel zu MPI6_Open_SimaticNet:

Im folgenden Beispiel wird mit der Funktion MPI6_Open_SimaticNet eine Verbindung zu dem im Dialog "PG/PC-Schnittstelle einstellen" selektierten Interface hergestellt.

```
WORD Error=0;          //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;      //Handle der neuen Kommunikationsinstanz

if (!MPI6_Open_SimaticNet(&DLLHandle, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

Anmerkung:

Die Verbindung SIMATIC®-NET sollte nicht für die Verbindungen MHJ-NetLink, NetLink PRO oder TCP/IP-Direkt verwendet werden. Für diese Kommunikationswege verwenden Sie bitte die entsprechenden Einleitungsfunktionen (z.B. MPI6_Open_NetLinkPro_TCP_SelectBaud, MPI6_OpenTcpIp usw.).

6.13 Die Funktion: MPI6_CloseCommunication

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen MPI6_OpenXXXX muss erfolgreich ausgeführt worden sein.

Kurzbeschreibung

Die Funktion MPI6_CloseCommunication muss als letztes aufgerufen werden, um die Kommunikation mit einem AG endgültig zu beenden. Bei dieser Funktion wird auch die über die Funktion "MPI6_OpenRS232" geöffnete Schnittstelle oder der über MPI6_OpenNetLink, MPI6_OpenTcpIp, MPI6_Open_SimaticNet, MPI6_Open_NetLinkPro_TCP_AutoBaud oder MPI6_Open_NetLinkPro_TCP_SelectBaud geöffnete Socket geschlossen. Des Weiteren wird die Kommunikationsinstanz, welche über das anzugebende Handle identifiziert wird, beseitigt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel zu MPI6_CloseCommunication:

Im folgenden Beispiel wird eine zuvor aufgebaute Kommunikation mit einer CPU wieder beendet, die Schnittstelle bzw. der Socket geschlossen und die Kommunikationsinstanz beseitigt.

```
WORD Error=0;
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers

if (!MPI6_CloseCommunication(MPIHandle, &Error)){
      MPI_A_GetDLLError(ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
else {
     MessageBox(AppHandle, "Kommunikation ohne Fehler beendet.", "",
                MB_ICONEXCLAMATION);
}//ende else
```

Anmerkung:
Auch wenn die Funktion einen Fehler meldet ist die serielle Schnittstelle bzw. der Socket geschlossen und die Kommunikationsinstanz beseitigt. Eine Ausnahme stellt der Fehler ERROR_MPIA_PARAMETER_ERROR (Nummer 510) dar, wird dieser geliefert, so ist das übergebene Handle nicht korrekt. In diesem Fall kann keine Aktion ausgeführt werden (entfällt bei .Net-Wrapper-Klasse).

6.14 Die Funktion: MPI6_GetAccessibleNodes

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen MPI6_OpenXXXX muss erfolgreich ausgeführt worden sein.
Wichtig:
Wurde die Einleitung über die eine TCP/IP-Funktion z.B. "MPI6_OpenTcpIp" oder "MPI6_OpenTcpIp_S71200" ausgeführt, so kann die Funktion nicht ausgeführt werden.

Kurzbeschreibung

Die Funktion MPI6_GetAccessibleNodes kann dazu verwendet werden, die MPI/DP-Adressen der angeschlossenen Geräte eines MPI-Netzes oder im Profibus-DP zu ermitteln.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Addresses	INT*	Integer-Array in welchem die MPI/DP-Adressen der einzelnen Teilnehmer im angeschlossenen MPI/DP-Netz aufgelistet werden. Das Array sollte für 127 Integer-Felder ausgelegt sein.
countAddresses	INT*	Liefert die Anzahl der Teilnehmer im angeschlossenen MPI/DP-Netz.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel MPI6_GetAccessibleNodes:

Im folgenden Beispiel werden die Teilnehmer am angeschlossenen MPI-Netz ermittelt.

```
int ComNr=2;          //Schnittstelle COM2
long BaudRate=115200; //Baudrate 115200
BYTE PGMPIAdresse=0;  //MPI-Adresse der DLL-Aplikation = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
bool SchnittstelleWarSchonAllokiert=false;
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
int Teilnehmer[130]={0}; //Array für die Teilnehmer
int AnzahlTeilnehmer=0;
char AusgabeText[255]={0};
//Verbindung aufbauen
if (!MPI6_OpenRS232(&MPIHandle, ComNr, BaudRate,
                  PGMPIAdresse,
                   HoechsteMPI,
                   &SchnittstelleWarSchonAllokiert,
                   &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung war erfolgreich.", "",
           MB_ICONINFORMATION);
//Erreichbare Teilnehmer ermitteln
if (!MPI6_GetAccessibleNodes(MPIHandle, Teilnehmer,
                             &AnzahlTeilnehmer,
                              &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
else {
     //Anzahl der Teilnehmer anzeigen
     wsprintf(AusgabeText, "Es sind %i Teilnehmer am MPI-Netz
              angeschlossen", AnzahlTeilnehmer);
     MessageBox(AppHandle, AusgabeText, "", MB_ICONEXCLAMATION);
}//ende else
//Kommunikation beenden
if (!MPI6_CloseCommunication(MPIHandle, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
```

6.15 Die Funktion: MPI6_SetRoutingData

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (außer MPI6_OpenRS232) (z.B. MPI6_OpenNetLink, MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.

Kurzbeschreibung

Die Funktion MPI6_SetRoutingData muss aufgerufen werden, bevor man über die Funktion MPI6_ConnectToPLCRouting eine nicht direkt angeschlossene CPU ansprechen möchte. Über die Parameter der Funktion wird die CPU angegeben, zu welcher geroutet werden soll. Routing bedeutet, man kommuniziert nicht mit der direkt am PC angeschlossenen CPU, sondern mit einer anderen CPU, welche mit dieser vernetzt ist.

Wichtige Ausnahme:

Das Routen ist über die Kommunikationswege MHJ-NetLink, NetLink PRO, TCP/IP-Direkt und SIMATIC®-NET möglich. Routen ist nicht möglich, wenn die Funktion MPI6_OpenRS232 verwendet wird.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beispiel zu Routing:

Bei der Erklärung der Funktion MPI6_ConnectToPLCRouting wird ein Beispiel zum Thema Routing vorgestellt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
TargetSlotNr	BYTE	Hier ist die Slotnummer (der Steckplatz) der anzusprechenden CPU anzugeben. Bei S7-300-Systemen ist hierbei 2 anzugeben. Bei S7-400-Systemen entnimmt man den Steckplatz der Hardwarekonfiguration.
TargetRackNr	BYTE	Angabe der Racknummer auf der die anzusprechende CPU steckt. Im Normalfall steckt die CPU auf dem Rack 0, weshalb hierbei 0 anzugeben ist,
TargetMPI_DP_Address	BYTE	Wird die Ziel-CPU über ein MPI oder DP-Netz erreicht, so ist an diesem Parameter die MPI/DP-Adresse anzugeben, welche die Ziel-CPU in diesem Netz besitzt.
TargetIPAddressStr	char*	Wird die Ziel-CPU über ein TCP/IP-Netz erreicht, so ist an diesem Parameter die IP-Adresse der CPU bzw. des Ethernet-CPs (der auf dem Rack der CPU sitzt) anzugeben.
TargetSubnetID_High	WORD	High-Wort der Subnetz-ID des S7-Netzes, über welches die CPU angesprochen wird. Die Subnetz-ID wird bei der Hardwarekonfiguration festgelegt.
TargetSubnetID_Low	WORD	Low-Wort der Subnetz-ID des S7-Netzes, über welches die CPU angesprochen wird. Die Subnetz-ID wird bei der Hardwarekonfiguration festgelegt.
TargetNetIsMPI_DP_Net	BYTE	Wird die CPU über ein MPI/DP-Netz angesprochen, so ist hierbei der Wert 1 zu übergeben. In diesem Fall wird der Parameter ZielBaugruppeIPAdresseStr nicht beachtet. Im Parameter ZielBaugruppeMPI_DP_Adresse muss die korrekte Adresse angegeben sein. Wird der Wert 0 übergeben, so ist die CPU über die IP-Adresse spezifiziert, die MPI/DP-Adresse wird in diesem Fall ignoriert und sollte mit 2 vorgegeben werden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

6.16 Die Funktion: MPI6_ConnectToPLC

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.

Kurzbeschreibung

Die Funktion MPI6_ConnectToPLC muss aufgerufen werden, bevor man die gewünschte CPU über Schreib- oder Lesezugriffe ansprechen kann. Über diese Funktion wird festgelegt, welcher Teilnehmer am angeschlossenen MPI- bzw. Profibus-DP-Netz angesprochen wird. Der Kommunikationspartner wird hierbei über die MPI-Adresse bzw. die Profibusadresse spezifiziert.

Wichtige Ausnahme:

Wurde die Einleitung über die Funktion MPI6_ConnectToPLC ausgeführt, dann ist die Angabe der MPI-Adresse ein Dummy-Wert, da die CPU über die IP-Adresse und die CPU-Slot-Nummer bereits angegeben ist. Hierbei ist als MPI/DP-Adresse immer der Wert 2 anzugeben.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PlcAddress	BYTE	Enthält die MPI/DP-Adresse des gewünschten Kommunikationspartners. Der Wert der Adresse kann zwischen 0 bis 126 liegen. Beim Kommunikationsweg TCP-IP-Direct ist generell der Wert 2 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Liefert die Funktion als Error-Code einen der folgenden Werte (dezimal):
1046: CPU ist keine S7-1500
1045: CPU ist keine LOGO
1044: CPU ist keine S7-1200
dann wurde intern bereits die Funktion "MPI6_CloseCommunication" aufgerufen und die Verbindung beendet.
Diese Fehler treten auf, wenn z.B. die Open-Funktion für die S7-1200® aufgerufen wird, in Wirklichkeit aber eine S7-1500® angeschlossen ist.

Beispiel

Im folgenden Beispiel wird die Kommunikation mit einem AG aufgebaut. Die CPU hat hierbei die MPI-Adresse 10.

```
int ComNr=2;           //Schnittstelle COM2
long BaudRate=115200; //Baudrate 115200
BYTE PGMPIAdresse=0;  //MPI-Adresse der DLL-Aplikation = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
bool SchnittstelleWarSchonAllokiert=false; //true wenn die
                                           //Schnittselle schon
                                           //genutzt wird
WORD Error=0;          //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;      //Handle der neuen Kommunikationsinstanz
BYTE AGMPIAdresse=10; //Die MPI-Adresse der anzusprechenden CPU
//Verbindung aufbauen
if (!MPI6_OpenRS232(&MPIHandle, ComNr, BaudRate, PGMPIAdresse,
                           HoechsteMPI,
                           &SchnittstelleWarSchonAllokiert,
                           &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung war erfolgreich.", "",
           MB_ICONINFORMATION);
//Kommunikation aufbauen
if (!MPI6_ConnectToPLC(MPIHandle, AGMPIAdresse, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
else {
     MessageBox(AppHandle, "Kommunikation erfolgreich aufgebaut!", "",
                MB_ICONINFORMATION);
}//ende else
//Kommunikation beenden
if (!MPI6_CloseCommunication(MPIHandle, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
```

6.17 Die Funktion: MPI6_ConnectToPLCRouting

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (außer MPI6_OpenRS232) (z.B. MPI6_OpenNetLink, MPI6_OpenTcpIp, usw.) muss erfolgreich ausgeführt worden sein. Weiterhin müssen die Routingdaten über die Funktion MPI6_SetRoutingData übergeben worden sein.

Kurzbeschreibung

Die Funktion MPI6_ConnectToPLCRouting muss aufgerufen werden, bevor man die gewünschte CPU über Schreib- oder Lesezugriffe ansprechen kann. Im Gegensatz zur Funktion MPI6_ConnectToPLC wird dabei nicht die direkt an den PC angeschlossene CPU angesprochen. Es wird eine mit der direkt angeschlossenen CPU vernetzte CPU angesprochen. Welche CPU angesprochen wird, kann über die Parameter der Funktion MPI6_SetRoutingData angegeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Wichtige Ausnahme:

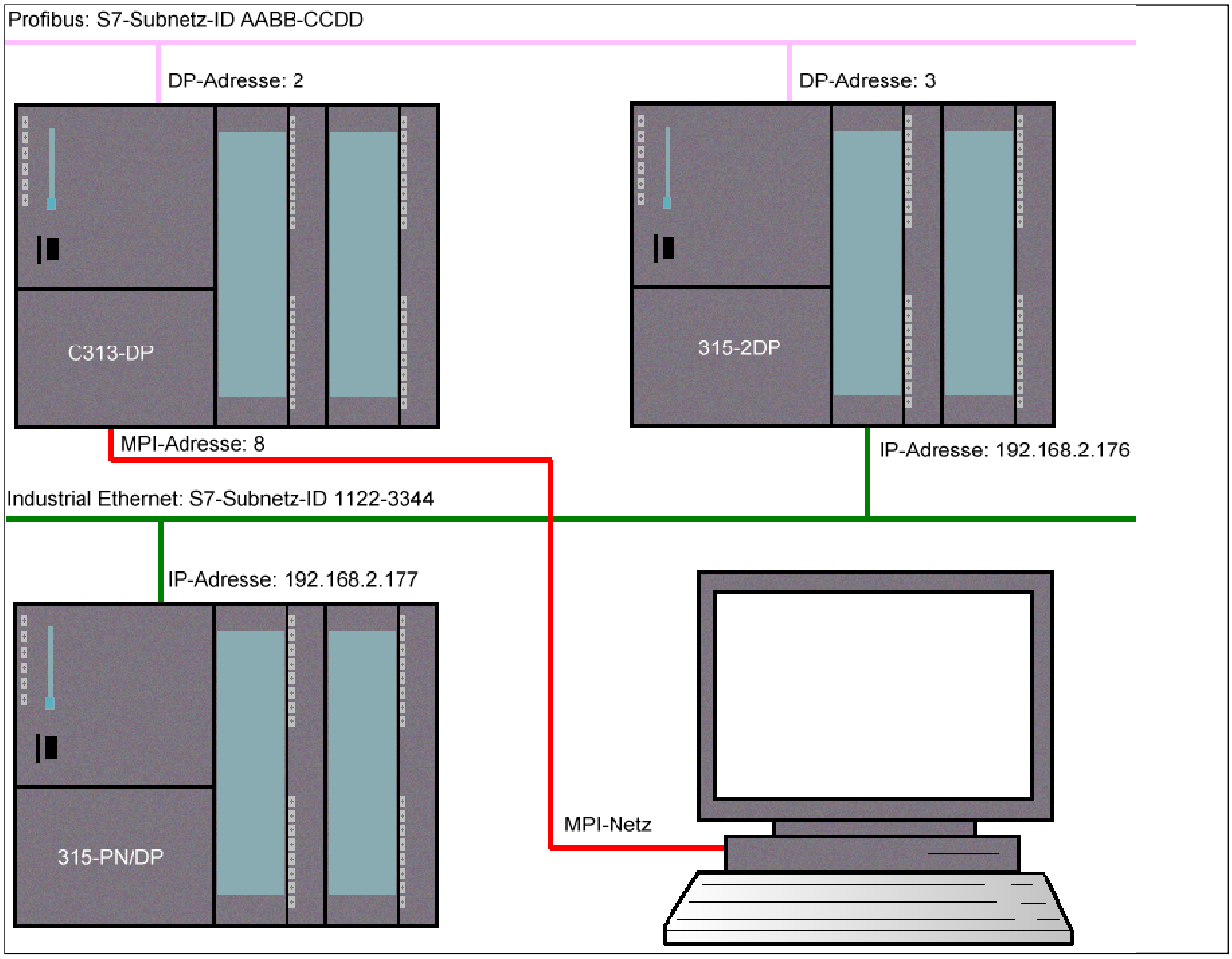
Wurde die Einleitung über die Funktion MPI6_OpenRS232 ausgeführt, dann ist das Routen nicht möglich. Die Funktion MPI6_ConnectToPLCRouting darf in diesem Fall nicht verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PlcAddress	BYTE	Enthält die MPI/DP-Adresse des direkt angeschlossenen Kommunikationspartners. Der Wert der Adresse kann zwischen 0 bis 126 liegen. Beim Kommunikationsweg TCP-IP-Direct ist generell der Wert 2 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

6.18 Beispiel zu Routing

Nachfolgend soll ein Beispiel für die beiden Funktionen MPI6_SetRoutingData und MPI6_ConnectToPLCRouting dargestellt werden. In dem Beispiel ist der PC über einen NetLink PRO mit der MPI-Schnittstelle einer CPU verbunden. Diese CPU ist mit zwei anderen CPUs vernetzt. In der folgenden Darstellung ist dies zu sehen:



Über die CPU "C313-DP" soll die CPU "315-PN/DP" angesprochen werden. Die beiden CPUs haben aber keine direkte Verbindung zueinander. Dies bedeutet, die Daten müssen von der C313-DP über Profibus an die 315-2DP weitergeleitet werden. Die 315-2DP wiederum, leitet die Daten über einen auf dem Rack befindlichen CP343 (Ethernet-CP), an die 315-PN/DP weiter. Die 315-PN/DP besitzt eine Ethernetschnittstelle im CPU-Gehäuse.

6.18.1 Einleitung über NetLink PRO

Im ersten Schritt ist die Einleitungsfunktion für den Kommunikationsweg NetLink PRO aufzurufen.

```
BYTE PGMPIAdresse=0;  //MPI-Adresse der Kommunikationsinstanz = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
char IPAdresseStr[50]={0};
//IP-Adresse des NetLink PRO eintragen
strcpy(IPAdresseStr, "192.168.2.100");
//Verbindung aufbauen
if (!MPI6_Open_NetLinkPro_TCP_AutoBaud(&MPIHandle, IPAdresseStr,
                                       PGMPIAdresse, HoechsteMPI,
                                      &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
      return;
}//ende if
MessageBox(AppHandle, "Einleitung erfolgreich.", "",
           MB_ICONINFORMATION);
```

Dabei ist kein Unterschied gegenüber einem Aufruf ohne Routing vorhanden. Beim Aufruf wird die IP-Adresse des NetLink PRO angegeben. Dieser hat im Beispiel die IP-Adresse "192.168.2.100".

6.18.2 Übergabe der Routingdaten

Nun müssen die Daten der wirklich anzusprechenden CPU über die Funktion MPI_A_SetRoutingData übergeben werden.

```
BYTE ZielBaugruppeSlotNr=2;
BYTE ZielBaugruppeRackNr=0;
BYTE ZielBaugruppeMPI_DP_Adresse=2;//Adresse nicht relevant
char ZielBaugruppeIPAdresseStr[50]={0};
strcpy(ZielBaugruppeIPAdresseStr, "192.168.2.177");//IP der 315-PN/DP
WORD ZielSubnetzID_High=0x1122;
WORD ZielSubnetzID_Low=0x3344;
BYTE ZielNetzIstMPI_DP_Netz=0;//Das Ziel-Netz ist Ethernet
//
if (!MPI6_SetRoutingData(DLLHandle, ZielBaugruppeSlotNr,
                         ZielBaugruppeRackNr, ZielBaugruppeMPI_DP_Adresse,
                         ZielBaugruppeIPAdresseStr, ZielSubnetzID_High,
                         ZielSubnetzID_Low, ZielNetzIstMPI_DP_Netz,
                         &Error)){
        //Fehler
        MPI_A_GetDLLError(DLLHandle, DLLErrorString, Error);
        MessageBox(AppHandle, DLLErrorString, "", MB_ICONSTOP);
        //
        return;
}//ende if
MessageBox(AppHandle, "Übergabe Routingdaten erfolgreich.", "",
           MB_ICONINFORMATION);
```

Erklärung der Parameter:

Als "TargetSlotNr" wird 2 übergeben, da die anzusprechende CPU auf dem Steckplatz 2 vorhanden ist (wie bei allen 300er-Systemen). Da die 315-PN/DP auf dem ersten Baugruppenträger sitzt, wird als "TargetRackNr" der Wert 0 übergeben.
Die Angabe "TargetMPI_DP_Address" ist im Beispiel nicht relevant, da die 315-PN/DP über Ethernet angesprochen wird. Die IP-Adresse ist im Parameter "TargetIPAddressStr" angegeben, dies ist die IP-Adresse der 315-PN/DP.
In den Parametern "TargetSubnetID_High" und "TargetSubnetID_Low" muss die S7-Subnetz-ID des Netzes angegeben werden, über welche die anzusprechenden CPU mit dem Verbund vernetzt ist. Im Beispiel wird die 315-PN/DP über das Ethernet angesprochen, an dem auch die 315-2DP über den Ethernet-CP angebunden ist. Dieses Ethernet hat die Subnetz-ID "1122-3344", deshalb sind diese Werte an den Parametern anzugeben.
Zuletzt ist der Parameter "TargetNetIsMPI_DP_Net" zu belegen. Im Beispiel muss dieser den Wert 0 erhalten, da das Ziel-Netz, über welches die 315-PN/DP angesprochen wird, ein Ethernet ist. Somit wird die IP-Adresse verwendet und nicht die MPI/DP-Adresse.

Damit sind die Parameter komplett, zuletzt muss nur noch die Funktion MPI6_ConnectToPLCRouting aufgerufen werden.

6.18.3 Aufruf der Funktion MPI6_ConnectToPLCRouting

Da nicht die direkt an den PC angeschlossene CPU angesprochen werden soll, sondern eine mit dieser CPU vernetzte CPU, darf nicht die Funktion MPI6_ConnectToPLC zum Einsatz kommen. Es muss die Funktion MPI6_ConnectToPLCRouting für den Aufbau der Kommunikation verwendet werden.
Nachfolgend ist der Aufruf im Beispiel zu sehen:

```
BYTE AGMPIAdresse=8; //Die MPI-Adresse der direkt verbundenen CPU
if (!MPI6_ConnectToPLCRouting(MPIHandle, AGMPIAdresse,
                              &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
else {
     MessageBox(AppHandle, "Kommunikation erfolgreich aufgebaut!", "",
                MB_ICONINFORMATION);
}//ende else
```

Als CPU-MPIAdresse ist der Wert 8 zu übergeben, denn dies ist die MPI-Adresse der CPU C313-DP, an welche der PC angeschlossen ist. Diese CPU leitet dann die Anfrage an die anzusprechenden CPU weiter.

6.18.4 Fazit zum Beispiel zu Routing

Damit sind die Aufrufe komplett, es kann nun über die Lese- und Schreibfunktionen auf die 315-PN/DP zugegriffen werden. Gegenüber dem direkten Ansprechen sind keine Unterschiede bei den Aufrufen der Funktionen vorhanden.
Das Beispiel hat gezeigt, dass beim Routing, nach der jeweiligen Einleitungsfunktion, die Funktion MPI6_SetRoutingData und danach die Funktion MPI6_ConnectToPLCRouting aufzurufen ist.

Bei der Hardwarekonfiguration der CPUs ist zu beachten, dass die Routingdaten mit übertragen werden. Im Simatic®-Manager muss z.B. eine solche Konfiguration mit NetPro ausgeführt werden. Dies ist notwendig, damit die CPUs Kenntnis davon haben, welche weiteren CPUs über sie erreichbar sind.

Weiterhin ist zu beachten, dass die Kommunikationsgeschwindigkeit beim Routing geringer ist, als bei einer direkten Verbindung.

ComDrvS7 unterstützt das Routing bei den Kommunikationswegen MHJ-NetLink, NetLink PRO, TCP/IP-Direkt und SIMATIC®-NET.

6.19 Die Funktion: MPI6_ReadByte

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadByte kann dazu verwendet werden, den Status von Eingangs-, Ausgangs-, Merker- und Datenbausteinbytes zu ermitteln.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.
In der **Micro-Version** ist auch der Zugriff auf den V-Datenbereich in einer LOGO möglich.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D, V wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version) = 86.
wAddress	WORD	Anfangsadresse ab welcher die Bytes gelesen werden sollen.
byteBufferPtr	BYTE*	In diesem Buffer werden die Statusinformationen abgelegt. Es muss darauf geachtet werden, dass der Bereich ausreichend gross dimensioniert ist, damit alle Statusinformationen abgelegt werden können. Der Buffer muss so viele Felder besitzen, wie Bytes abgefragt werden.
wCountBytes	WORD	Anzahl der zu lesenden Bytes. Es können max. 65535 Bytes mit einem Aufruf gelesen werden.
wDBNR	WORD	Handelt es sich bei den zu lesenden Bytes um den Inhalt eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden von dem Kommunikationspartner mit der MPI-Adresse 10 die Statusinformationen ab dem Merkerbyte 10 erfasst. Es sollen 30 Byte ausgelesen werden.

```
int ComNr=2;          //Schnittstelle COM2
long BaudRate=115200; //Baudrate 115200
BYTE PGMPIAdresse=0;  //MPI-Adresse der DLL-Aplikation = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
bool SchnittstelleWarSchonAllokiert=false;
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
int MPIHandle=-1;     //Handle der neuen Kommunikationsinstanz
BYTE AGMPIAdresse=10; //Die MPI-Adresse der anzusprechenden CPU
bool Fehler=false;
char AusgabeStr[255]={0};//String für Text-Ausgabe
//Verbindung aufbauen
if (!MPI6_OpenRS232(&MPIHandle, ComNr, BaudRate, PGMPIAdresse,
                           HoechsteMPI,
                           &SchnittstelleWarSchonAllokiert,
                           &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung war erfolgreich.", "",
          MB_ICONINFORMATION);
//Kommunikation aufbauen
if (!MPI6_ConnectToPLC(MPIHandle, AGMPIAdresse, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     Fehler=true;
}//ende if
else {
     MessageBox(AppHandle, "Kommunikation erfolgreich aufgebaut!",
               "", MB_ICONINFORMATION);
}//ende else
//Daten lesen
if (!Fehler){
    BYTE StatusBuffer[100]={0};
    BYTE Operand=77; //Merker ASCII-Code 77
    if (!MPI6_ReadByte(MPIHandle, Operand, 10, StatusBuffer,
                       30, 0, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       wsprintf(AusgabeStr, "Merkerbyte 11 hat den Status: %02X",
                StatusBuffer[1]);
       MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
```

```
//Kommunikation beenden
if (!MPI6_CloseCommunication(MPIHandle, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
```

6.20 Die Funktion: MPI6_ReadWord

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadWord kann dazu verwendet werden, den Status von Eingangs-, Ausgangs-, Merker- und Datenbaustein-Worten zu ermitteln.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version) = 86.
wAddress	WORD	Anfangsadresse ab welcher die Worte gelesen werden sollen.
wordBufferPtr	WORD*	In diesem Buffer werden die Statusinformationen abgelegt. Es muss darauf geachtet werden, dass der Bereich ausreichend gross dimensioniert ist, damit alle Statusinformationen abgelegt werden können. Der Buffer muss so viele Felder besitzen, wie Worte abgefragt werden.
wCountWord	WORD	Anzahl der zu lesenden worte. Es können max. 65535 Worte mit einem Aufruf gelesen werden.
wDBNR	WORD	Handelt es sich bei den zu lesenden Worten um den Inhalt eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Es ist zu beachten dass die zurückgelieferten Statuswerte folgendermaßen aufgebaut sind (Beispiel das Merkerwort MW2):
Merkerwort 2 = Merkerbyte 2 (HIBYTE) und Merkerbyte 3 (LOBYTE)
Es ist ebenfalls zu beachten, dass sich byteorientierte Wortoperanden überschneiden. Aus diesem Grund liest die Funktion entweder alle geradzahligen Wörter oder alle ungeradzahligen Wörter im angegebenen Bereich. Dies hängt ab von der Angabe im Wert "wAddress". Wird hierbei eine gerade Zahl angegeben (z.B. 10), so werden alle geradzahligen Wörter gelesen. Wird als Wert z.B. 13 übergeben, so werden alle ungeraden Wörter gelesen.
Eigentlich ist es nur sinnvoll geradzahlige Wörter zu lesen, die Option wurde offen gelassen um auch den Ausnahmen zu entsprechen.

Beispiel

Im folgenden Beispiel werden von einem Kommunikationspartner die Statusinformationen der geradzahligen Merkerwörter ab dem Merkerwort 0 ausgelesen. Es sollen 5 Wörter gelesen werden. Im StatusBuffer stehen nach der Aktion die Inhalte der Merkerwörter MW0, MW2, MW4, MW6 und MW8.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion MPI6_CloseCommunication kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
    WORD StatusBuffer[100]={0};
    BYTE Operand=77; //Merker ASCII-Code 77
    if (!MPI6_ReadWord(MPIHandle, Operand, 0, StatusBuffer,
                       5, 0, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       char AusgabeStr[255]={0};
       wsprintf(AusgabeStr, "Merkerwort 2 hat den Status (Hex):
               %04X", StatusBuffer[1]);
       MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.21 Die Funktion: MPI6_ReadDword

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadDword kann dazu verwendet werden, den Status von Eingangs-, Ausgangs-, Merker- und Datenbaustein-Doppelworten zu ermitteln.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version) = 86.
wAddress	WORD	Anfangsadresse ab welcher die Doppelworte gelesen werden sollen.
dwordBufferPtr	DWORD*	In diesem Buffer werden die Statusinformationen abgelegt. Es muss darauf geachtet werden, dass der Bereich ausreichend gross dimensioniert ist, damit alle Statusinformationen abgelegt werden können. Der Buffer muss so viele Felder besitzen, wie Doppelworte abgefragt werden.
wCountDword	WORD	Anzahl der zu lesenden Doppelworte. Es können max. 65535 Doppelworte mit einem Aufruf gelesen werden.
wDBNR	WORD	Handelt es sich bei den zu lesenden Doppelworten um den Inhalt eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Es ist zu beachten dass die zurückgelieferten Statuswerte folgendermaßen aufgebaut sind (Beispiel das Merkerdoppelwort MD2):
Das MD2 besteht aus den beiden Merkerwörtern MW2 und MW4, wobei MW2 das Hi-Wort ist. MW2 besteht wiederum aus den Bytes MB2 und MB3. MW4 besteht aus den Bytes MB4 und MB5. Somit umfasst das MD2 die 4 Bytes MB2, MB3, MB4 und MB5.
Es ist zu beachten, dass sich byteorientierte Doppelwortoperanden überschneiden. Aus diesem Grund liest die Funktion entweder alle geradzahligen Doppelwörter oder alle ungeradzahligen Doppelwörter im angegebenen Bereich. Dies hängt ab von der Angabe im Wert "wAddress". Wird hierbei eine gerade Zahl angegeben (z.B. 10), so werden alle geradzahligen Doppelwörter gelesen. Wird als Wert z.B. 13 übergeben, so werden alle ungeraden Doppelwörter gelesen. Eigentlich ist es nur sinnvoll geradzahlige Doppelwörter zu lesen, die Option wurde offen gelassen um auch den Ausnahmen zu entsprechen.

Beispiel

Im folgenden Beispiel werden von einem Kommunikationspartner die Statusinformationen der geradzahligen Merkerdoppelwörter ab dem Merkerdoppelwort 0 ausgelesen. Es sollen 5 Doppelwörter gelesen werden. Im StatusBuffer stehen nach der Aktion die Inhalte der Merkerdoppelwörter MD0, MD4, MD8, MD12 und MD16.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion MPI6_CloseCommunication kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
    DWORD StatusBuffer[100]={0};
    BYTE Operand=77; //Merker ASCII-Code 77
    if (!MPI6_ReadDword(MPIHandle, Operand, 0, StatusBuffer,
                        5, 0, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       char AusgabeStr[255]={0};
       wsprintf(AusgabeStr, "MD4 hat den Status (Hex):
                            %08X", StatusBuffer[1]);
       MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
```

6.22 Die Funktion: MPI6_ReadTimer (Nicht in Lite-Version)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadTimer kann dazu verwendet werden, den Status von Zeitbausteinen zu ermitteln.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
wAddress	WORD	Anfangsadresse ab welcher die Timer gelesen werden sollen.
wordBufferPtr	WORD*	In diesem Buffer werden die Statusinformationen abgelegt. Es muss darauf geachtet werden, dass der Bereich ausreichend gross dimensioniert ist, damit alle Statusinformationen abgelegt werden können. Der Buffer muss so viele Felder besitzen, wie Timer abgefragt werden.
wCountTimer	WORD	Anzahl der zu lesenden Timer. Es können max. 65535 Timer mit einem Aufruf gelesen werden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Das Timer-Wort ist folgendermaßen aufgebaut:
Bit 0-9: Zeitfaktor BCD-codiert
Bit 12+13: Zeitbasis (0=10ms, 1=100ms, 2=1s, 3=10s)

Beispiel

Im folgenden Beispiel werden von einem Kommunikationspartner die Statusinformationen der Timer 0 bis 4 ausgelesen.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion MPI6_CloseCommunication kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
    WORD StatusBuffer[10]={0};
    if (!MPI6_ReadTimer(MPIHandle, 0, StatusBuffer, 5, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       char AusgabeStr[255]={0};
       wsprintf(AusgabeStr, "Timer 1 hat den Status (Hex):
                            %04X", StatusBuffer[1]);
       MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.23 Die Funktion: MPI6_ReadCounter (Nicht in Lite-Version)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadCounter kann dazu verwendet werden, den Status von Zählerbausteinen zu ermitteln.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
wAddress	WORD	Anfangsadresse ab welcher die Zähler gelesen werden sollen.
wordBufferPtr	WORD*	In diesem Buffer werden die Statusinformationen abgelegt. Es muss darauf geachtet werden, dass der Bereich ausreichend gross dimensioniert ist, damit alle Statusinformationen abgelegt werden können. Der Buffer muss so viele Felder besitzen, wie Zähler abgefragt werden.
wCountCounter	WORD	Anzahl der zu lesenden Zähler. Es können max. 65535 Zähler mit einem Aufruf gelesen werden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Das Zähler-Wort ist folgendermaßen aufgebaut:
Bit 0-9: Zählerstand BCD-codiert

Beispiel

Im folgenden Beispiel werden von einem Kommunikationspartner die Statusinformationen der Zähler 0 bis 4 ausgelesen.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion MPI6_CloseCommunication kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
    WORD StatusBuffer[10]={0};
    if (!MPI6_ReadCounter(MPIHandle, 0, StatusBuffer, 5, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       char AusgabeStr[255]={0};
       wsprintf(AusgabeStr, "Zähler 1 hat den Stand (Hex):
                            %04X", StatusBuffer[1]);
       MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.24 Die Funktion: MPI6_MixRead_2

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_MixRead_2 kann dazu verwendet werden, den Status von Operanden der Bereiche E, A, M, DB, T und Z zu lesen.
Das Besondere dabei ist, dass der Status von verschiedenen Operanden gemischt in einem Aufruf gelesen werden kann. So ist es z.B. möglich, den Status des Eingangswortes EW2, des Merkerbytes MB10, dem Datenwort 2 im DB10 (DB10.DBW2) und dem Zeitbaustein T2 mit einem Aufruf der Funktion MPI6_MixRead_2 zu lesen.
Dabei optimiert die Funktion automatisch die Abfrage. Es werden Operandenüberschneidungen, Doppeltabfragen usw. erkannt und das Protokoll zur CPU entsprechend optimiert.

Die Funktion kann immer dann zum Einsatz kommen, wenn der Status von unterschiedlichen Operandenbereichen gelesen werden soll oder aber wenn unterschiedliche Adressen eines Operandenbereichs zu lesen sind (z.B. MW0, MW100, MB150 usw.).

Einschränkungen der Lite-Version

In der Lite-Version können nur Daten von Datenbausteinen gelesen werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Op_Type	DWORD*	Array mit den Operandentypen der zu lesenden Operanden. Merker = 0x0101 Eingang = 0x1101 Ausgang = 0x2101 Timer = 0x5401 Zähler = 0x6401 DB-Daten = 0x7101 V-Daten = 0xF101 (Nur Micro-Version)
Op_Address	DWORD*	Array mit den Adressen der zu lesenden Operanden.
DBNr	DWORD*	Array mit den DB-Nummern falls ein Operand ein DB-Datum ist. Bei nicht-DB-Daten ist der Inhalt des jeweiligen Index=0.
Op_LengthByte	DWORD*	Array mit den jeweiligen Längen der Operanden in Byte. Erlaubte Werte sind 1, 2 und 4.
Data	DWORD*	Array mit den gelieferten Statuswerten für die Operanden.
wCountParam	WORD	Anzahl der in den Arrays angegebenen Operanden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden von einem Kommunikationspartner die Statusinformationen der Operanden MB10, DB2.DBW0 und DB1.DBD100 angefordert.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //Anlegen der Arrays
 DWORD Op_Type[3];
 DWORD Op_Address[3];
 DWORD Op_LengthByte[3];
 DWORD DBNr[3];
 DWORD Data[3];
 //MB10 im Array-Index 0 eintragen
 Op_Type[0]=0x0101; //Merker
 Op_Address[0]=10;  //Adresse 10
 Op_LengthByte[0]=1; //Länge 1 Byte
 DBNr[0]=0; //DB-Nummer=0 da kein DB-Datum
 //DB2.DBW0 im Array-Index 1 eintragen
 Op_Type[1]=0x7101; //DB-Datum
 Op_Address[1]=0;  //Adresse 0
 Op_LengthByte[1]=2; //Länge 2 Byte = 1 Wort
 DBNr[1]=2; //DB-Nummer=2
 //DB1.DBD100 im Array-Index 2 eintragen
 Op_Type[2]=0x7101; //DB-Datum
 Op_Address[2]=100;  //Adresse 100
 Op_LengthByte[2]=4; //Länge 4 Byte = 1 Doppelwort
 DBNr[2]=1; //DB-Nummer=1
 //Anzahl der zu lesenden Operanden
 WORD wCountParam=3; //3 Operanden
 //Aufruf der MixRead-Funktion
 if (!MPI6_MixRead_2(MPIHandleV6, Op_Type, Op_Address, DBNr,
                     Op_LengthByte, Data, wCountParam, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     char AusgabeStr[255]={0};
     wsprintf(AusgabeStr, "DB2.DBW0 hat den Status(Hex): %04X",
             LOWORD(Data[1]));
     MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
 }//ende else
 //
.
.
.
```

6.25 Die Funktion: MPI6_WriteBit_2

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteBit_2 kann dazu verwendet werden, den Wert von Eingangs-, Ausgangs-, Merker- und Datenbausteinbits zu steuern. Dies bedeutet, die Operanden können auf den der Funktion übergebenen Wert gesetzt werden.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Anmerkung

Die Verwendung der Funktion MPI6_WriteBit_2 ist eigentlich uneffektiv, diese sollte nur in Ausnahmefällen verwendet werden. Die Funktion sollte nur verwendet werden, wenn es wichtig ist, wirklich nur 1 Bit zu steuern und die anderen Bits des Bytes unberührt zu lassen. In allen anderen Fällen sind die WriteByte-, WriteWord- oder WriteDword-Funktionen vorzuziehen.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version) = 86.
wByteAddress	WORD	Byteadresse des Bitoperanden. Beispiel: 10 bei M10.3
bBitAddress	BYTE	Bitadresse des Bitoperanden. Beispiel: 3 bei M10.3
bValue	BYTE	Angabe des Steuerwertes. Erlaubte Werte sind 0 und 1.
wDBNR	WORD	Handelt es sich bei dem zu beeinflußenden Bit um das Bit eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird in dem Kommunikationspartner das Merkerbit M10.3 auf den Wert 1 gesetzt.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    BYTE SteuerWert=1;
    WORD wByteAddress=10;
    BYTE bBitAddress=3;
    WORD wDBNR=0;
    BYTE Operand=77; //Merker ASCII-Code 77

    if (!MPI6_WriteBit_2(MPIHandle, Operand, wByteAddress,
                         bBitAddress, SteuerWert, wDBNR, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Das Steuern war erfolgreich!", "",
                  MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.26 Die Funktion: MPI6_WriteByte

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteByte kann dazu verwendet werden, den Wert von Eingangs-, Ausgangs-, Merker- und Datenbausteinbytes zu steuern. Dies bedeutet, die Operanden können auf die der Funktion übergebenen Werte gesetzt werden.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version)= 86.
wAddress	WORD	Anfangsadresse ab welcher die Bytes beschrieben werden sollen.
byteBufferPtr	BYTE*	In diesem Buffer werden die Steuerinformationen abgelegt. Es muss darauf geachtet werden, dass der Buffer ausreichend gross dimensioniert ist, damit alle Steuerinformationen geschrieben werden können. Der Buffer muss so viele Felder besitzen, wie Bytes zum Steuern angegeben wurden.
wCountBytes	WORD	Anzahl der zu schreibenden Bytes. Es können max. 65535 Bytes mit einem Aufruf geschrieben werden.
wDBNR	WORD	Handelt es sich bei den zu schreibenden Bytes um die Daten eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden in dem Kommunikationspartner die Merkerbytes 0 bis 9 auf den Wert FF(Hex) gesetzt.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    BYTE SteuernBuffer[10]={0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                            0xFF, 0xFF, 0xFF};
    BYTE Operand=77; //Merker ASCII-Code 77
    if (!MPI6_WriteByte(MPIHandle, Operand, 0, SteuernBuffer,
                        10, 0, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Das Steuern war erfolgreich!", "",
                  MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.27 Die Funktion: MPI6_WriteWord

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteWord kann dazu verwendet werden, den Wert von Eingangs-, Ausgangs-, Merker- und Datenbausteinwörtern zu steuern. Dies bedeutet, die Operanden können auf die der Funktion übergebenen Werte gesetzt werden.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version)= 86.
wAddress	WORD	Anfangsadresse ab welcher die Wörter beschrieben werden sollen.
wordBufferPtr	WORD*	In diesem Buffer werden die Steuerinformationen abgelegt. Es muss darauf geachtet werden, dass der Buffer ausreichend gross dimensioniert ist, damit alle Steuerinformationen geschrieben werden können. Der Buffer muss so viele Felder besitzen, wie Wörter zum Steuern angegeben wurden.
wCountWord	WORD	Anzahl der zu schreibenden Wörter. Es können max. 65535 Worte mit einem Aufruf geschrieben werden.
wDBNR	WORD	Handelt es sich bei den zu schreibenden Wörtern um die Daten eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Es ist zu beachten dass die im wordBufferPtr angegebenen Werte folgendermaßen geschrieben werden (Beispiel das Merkerwort MW2):

MB 2 = HIBYTE(wordBufferPtr[0])
MB 3 = LOBYTE(wordBufferPtr[0])

Es ist ebenfalls zu beachten, dass sich byteorientierte Wortoperanden überschneiden. Aus diesem Grund beschreibt die Funktion entweder alle geradzahligen Wörter oder alle ungeradzahligen Wörter im angegebenen Bereich. Dies hängt ab von der Angabe im Wert "wAddress". Wird hierbei eine gerade Zahl angegeben (z.B. 10), so werden alle geradzahligen Wörter beschrieben. Wird als Wert z.B. 13 übergeben, so werden alle ungeraden Wörter beschrieben.
Eigentlich ist es nur sinnvoll geradzahlige Wörter zu schreiben, die Option wurde offen gelassen um auch den Ausnahmen zu entsprechen.

Beispiel

Im folgenden Beispiel werden in dem Kommunikationspartner die Merkerworte 30 bis 38 auf den Wert 1F00(Hex) gesetzt.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    WORD SteuernBuffer[5]={0x1F00, 0x1F00, 0x1F00, 0x1F00, 0x1F00};
    BYTE Operand=77; //Merker ASCII-Code 77
    if (!MPI6_WriteWord(MPIHandle, Operand, 30, SteuernBuffer,
                        5, 0, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Das Steuern war erfolgreich!", "",
                  MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.28 Die Funktion: MPI6_WriteDword

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteDword kann dazu verwendet werden, den Wert von Eingangs-, Ausgangs-, Merker- und Datenbausteindoppelwörtern zu steuern. Dies bedeutet, die Operanden können auf die der Funktion übergebenen Werte gesetzt werden.
In der **Lite-Version** ist nur der Zugriff auf Datenbausteine möglich.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
operand	BYTE	Über den ASCII-Code für die Buchstaben E, A, M, D wird der Operandenbereich angegeben in welchem gelesen werden soll. Eingänge = 69, Ausgänge = 65, Merker = 77, Datenbausteine = 68, V-Bereich (Nur Micro-Version)= 86.
wAddress	WORD	Anfangsadresse ab welcher die Doppelwörter beschrieben werden sollen.
dwordBufferPtr	DWORD*	In diesem Buffer werden die Steuerinformationen abgelegt. Es muss darauf geachtet werden, dass der Buffer ausreichend gross dimensioniert ist, damit alle Steuerinformationen geschrieben werden können. Der Buffer muss so viele Felder besitzen, wie Doppelwörter zum Steuern angegeben wurden.
wCountDword	WORD	Anzahl der zu schreibenden Doppelwörter. Es können max. 65535 Doppelworte mit einem Aufruf geschrieben werden.
wDBNR	WORD	Handelt es sich bei den zu schreibenden Wörtern um die Daten eines Datenbausteins, so muss hier die Nummer des DBs (1-65535) angegeben werden. Anderenfalls ist 0 anzugeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Es ist zu beachten dass die im dwordBufferPtr angegebenen Werte folgendermaßen geschrieben werden (Beispiel das Merkerdoppelwort MD2):

MB 2 = HIBYTE(HIWORD(dwordBufferPtr[0]))
MB 3 = LOBYTE(HIWORD(dwordBufferPtr[0]))
MB 4 = HIBYTE(LOWORD(dwordBufferPtr[0]))
MB 5 = LOBYTE(LOWORD(dwordBufferPtr[0]))

Es ist ebenfalls zu beachten, dass sich byteorientierte Wortoperanden überschneiden. Aus diesem Grund beschreibt die Funktion entweder alle geradzahligen Doppelwörter oder alle ungeradzahligen Doppelwörter im angegebenen Bereich. Dies hängt ab von der Angabe im Wert "wAddress". Wird hierbei eine gerade Zahl angegeben (z.B. 10), so werden alle geradzahligen Doppelwörter beschrieben. Wird als Wert z.B. 13 übergeben, so werden alle ungeraden Doppelwörter beschrieben.
Eigentlich ist es nur sinnvoll geradzahlige Doppelwörter zu schreiben, die Option wurde offen gelassen um auch den Ausnahmen zu entsprechen.

Beispiel

Im folgenden Beispiel werden in dem Kommunikationspartner die Merkerdoppelworte 10 und 14 auf den Wert 11223344(Hex) gesetzt.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    DWORD SteuernBuffer[2]={0x11223344, 0x11223344};
    BYTE Operand=77; //Merker ASCII-Code 77
    if (!MPI6_WriteDword(MPIHandle, Operand, 10, SteuernBuffer,
                         2, 0, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Das Steuern war erfolgreich!", "",
                  MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.29 Die Funktion: MPI6_WriteTimer (Nicht in der Lite-Version)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteTimer kann dazu verwendet werden, den Wert von Zeitbausteinen zu steuern. Dies bedeutet, die Timer können auf die der Funktion übergebenen Werte gesetzt werden.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
wAddress	WORD	Anfangsadresse ab welchem Timer gesteuert werden sollen.
wordBufferPtr	WORD*	In diesem Buffer werden die Steuerinformationen abgelegt. Es muss darauf geachtet werden, dass der Buffer ausreichend gross dimensioniert ist, damit alle Steuerinformationen geschrieben werden können. Der Buffer muss so viele Felder besitzen, wie Timer zum Steuern angegeben wurden.
wCountTimer	WORD	Anzahl der zu schreibenden Timer. Es können max. 65535 Timer mit einem Aufruf geschrieben werden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Das Timer-Wort ist folgendermaßen aufgebaut:
Bit 0-9: Zeitfaktor BCD-codiert
Bit 12+13: Zeibasis (0=10ms, 1=100ms, 2=1s, 3=10s)

Beispiel

Im folgenden Beispiel werden in dem Kommunikationspartner die Timer 3 bis 9 mit dem Wert 100 (Hex) beschrieben.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    WORD SteuernBuffer[7]={0x100, 0x100, 0x100, 0x100, 0x100, 0x100,
                           0x100};
    if (!MPI6_WriteTimer(MPIHandle, 3, SteuernBuffer, 7, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Das Steuern war erfolgreich!", "",
                  MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.30 Die Funktion: MPI6_WriteCounter (Nicht in der Lite-Version)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteCounter kann dazu verwendet werden, den Wert von Zählerbausteinen zu steuern. Dies bedeutet, die Zähler können auf die der Funktion übergebenen Werte gesetzt werden.
Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
wAddress	WORD	Anfangsadresse ab welchem Zähler gesteuert werden sollen.
wordBufferPtr	WORD*	In diesem Buffer werden die Steuerinformationen abgelegt. Es muss darauf geachtet werden, dass der Buffer ausreichend gross dimensioniert ist, damit alle Steuerinformationen geschrieben werden können. Der Buffer muss so viele Felder besitzen, wie Zähler zum Steuern angegeben wurden.
wCountCounter	WORD	Anzahl der zu schreibenden Zähler. Es können max. 65535 Zähler mit einem Aufruf geschrieben werden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Das Zähler-Wort ist folgendermaßen aufgebaut:
Bit 0-9: Zählerwert BCD-codiert

Beispiel

Im folgenden Beispiel werden in dem Kommunikationspartner die Zähler 10 bis 21 mit dem Wert 10 (Hex) beschrieben.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    WORD SteuernBuffer[12]={0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
                            0x10, 0x10, 0x10, 0x10, 0x10, 0x10};
    if (!MPI6_WriteCounter(MPIHandle, 10, SteuernBuffer, 12
                           &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Das Steuern war erfolgreich!", "",
                  MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.31 Die Funktion: MPI6_MixWrite_2

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_MixWrite_2 kann dazu verwendet werden, Operanden der Bereiche E, A, M, DB, T und Z zu steuern also auf einen vorgegeben Wert zu setzen.
Das Besondere dabei ist, dass das Steuern von verschiedenen Operanden gemischt in einem Aufruf erledigt werden kann. So ist es z.B. möglich das Eingangswortes EW2, das Merkerbytes MB10, das Datenwort 2 im DB10 (DB10.DBW2) und den Zeitbaustein T2 mit einem Aufruf der Funktion MPI6_MixWrite_2 zu steuern.
Dabei optimiert die Funktion automatisch die Steueranforderung. Es werden Operandenüberschneidungen, Doppeltangaben usw. erkannt und das Protokoll zur CPU entsprechend optimiert.

Die Funktion kann immer dann zum Einsatz kommen, wenn das Steuern von unterschiedlichen Operandenbereichen erledigt werden soll oder aber wenn unterschiedliche Adressen eines Operandenbereichs zu beschreiben sind (z.B. MW0, MW100, MB150 usw.).

Einschränkungen der Lite-Version

In der Lite-Version können nur Daten von Datenbausteinen gesteuert werden.

Ist die CPU über ein Passwort geschützt, so muss dieses nicht übergeben werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann auch bei den CPUs der Reihe S7-1500®, S7-1200® und LOGO!® verwendet werden. Bei S7-1200/1500 dürfen DBs allerdings nicht über die Option "Nur symbolisch adressierbar" erzeugt worden sein. Bei LOGO!® sind keine DBs vorhanden, hier können Eingänge, Ausgänge und der V-Bereich angegeben werden. Der Zugriff auf LOGO!® ist nur in der Micro-Version möglich.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Op_Type	DWORD*	Array mit den Operandentypen der zu steuernden Operanden. Merker = 0x0101 Eingang = 0x1101 Ausgang = 0x2101 Timer = 0x5401 Zähler = 0x6401 DB-Daten = 0x7101 V-Bereich = 0xF101 (Nur Micro-Version)
Op_Address	DWORD*	Array mit den Adressen der zu steuernden Operanden.
DBNr	DWORD*	Array mit den DB-Nummern falls ein Operand ein DB-Datum ist. Bei nicht-DB-Daten ist der Inhalt des jeweiligen Index=0.
Op_LengthByte	DWORD*	Array mit den jeweiligen Längen der Operanden in Byte. Erlaubte Werte sind 1, 2 und 4.
Data	DWORD*	Array mit den Steuerwerten für die Operanden.
wCountParam	WORD	Anzahl der in den Arrays angegebenen Operanden.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden in der CPU die Operanden MB10, DB2.DBW0 und DB1.DBD100 auf Vorgabewerte gesteuert.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //
 DWORD Op_Type[3];
 DWORD Op_Address[3];
 DWORD Op_LengthByte[3];
 DWORD DBNr[3];
 DWORD Data[3];
 //MB10: Daten im Arrayindex 0 eintragen
 Op_Type[0]=0x0101; //Merker
 Op_Address[0]=10;  //Adresse 10
 Op_LengthByte[0]=1; //Länge 1 Byte
 DBNr[0]=0; //DB-Nummer=0 da kein DB-Datum
 Data[0]=0x33; //Steuerwert
 //DB2.DBW0: Daten im Arrayindex 1 eintragen
 Op_Type[1]=0x7101; //DB-Datum
 Op_Address[1]=0;  //Adresse 0
 Op_LengthByte[1]=2; //Länge 2 Byte = 1 Wort
 DBNr[1]=2; //DB-Nummer=2
 Data[1]=0x1122; //Steuerwert
 //DB1.DBD100: Daten im Arrayindex 2 eintragen
 Op_Type[2]=0x7101; //DB-Datum
 Op_Address[2]=100;  //Adresse 100
 Op_LengthByte[2]=4; //Länge 4 Byte = 1 Doppelwort
 DBNr[2]=1; //DB-Nummer=1
 Data[2]=0x55667788; //Steuerwert
 //Anzahl der zu steuernden Operanden
 WORD wCountParam=3; //3 Operanden
 //Aufruf der MixWrite-Funktion
 if (!MPI6_MixWrite_2(MPIHandleV6, Op_Type, Op_Address, DBNr,
                      Op_LengthByte, Data, wCountParam, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(AppHandle, "Steuern war erfolgreich.", "",
                MB_ICONINFORMATION);
 }//ende else
 //
.
.
.
```

6.32 Die Funktion: MPI6_WriteBit (Nicht in Lite-Version)

Diese Funktion ist nur aus kompatibilitätsgründen vorhanden. Bei neuen Applikationen sollte die Funktion MPI6_WriteBit_2 verwendet werden!

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteBit kann dazu verwendet werden, den Inhalt eines Bits der Operandenbereiche Eingang, Ausgang, Merker und Daten zu verändern.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Anmerkung

Die Verwendung der Funktion MPI6_WriteBit ist eigentlich uneffektiv, diese sollte nur in Ausnahmefällen verwendet werden. Die Funktion sollte nur verwendet werden, wenn es wichtig ist, wirklich nur 1 Bit zu steuern und die anderen Bits des Bytes unberührt zu lassen. In allen anderen Fällen sind die WriteByte-, WriteWord- oder WriteDword-Funktionen vorzuziehen.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
ByteAddress	WORD	Gibt an, ab welche Byteadresse der Bitoperand besitzt. Angabe z.B. '0'
BitAddress	WORD	Gibt an, welche Bitadresse angesprochen wird. Bereich 0 - 7.
DBNr	WORD	Bei Beeinflussung eines Datenbits, wird über DBNummer die Nummer des Datenbausteins angegeben, in welchem sich das Datenbit befindet. Handelt es sich bei der Angabe nicht um ein Datenbit, so ist als DBNummer die Zahl '0' zu übergeben.
Operand	char	An diesen Parameter ist der Operandtyp in Großbuchstaben zu übergeben, dabei gilt: "E" = Eingänge, "A" = Ausgänge, "M" = Merker, "D" = Datenbits in DB
WriteBuffer	WORD*	In diesem Buffer wird der Steuerwert (0 oder 1) angegeben.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird das Mekerbit M10.1 in dem Kommunikationspartner Wert '1' gesetzt. Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten schreiben
if (!Fehler){
    WORD SteuerWert=1;
    if (!MPI6_WriteBit(MPIHandle, 10, 1, 0, 'M', &SteuerWert,
                       &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
    }//ende if
    else {
       MessageBox(AppHandle, "Der Operand M10.1 wurde erfolgreich "
                  "gesteuert!", "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.33 Die Funktion: MPI6_WriteDBFromWldToPlc (Nicht in den Lite- und CE-Versionen)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_WriteDBFromWldToPlc kann dazu verwendet werden,
einen Datenbaustein welcher sich in einer WLD-Datei befindet, in die CPU zu übertragen.
WLD-Dateien können von S7-Programmiersystemen (z.B. Simativ-Manager und WinSPS-S7) erzeugt und mit Bausteinen gefüllt werden. Ebenso können die S7-Programmiersysteme Bausteine aus WLD-Dateien lesen und bearbeiten.
Zusammen mit der Funktion MPI6_ReadDBFromPlcAndWriteToWld kann man mit ComDrvS7 Datenbausteine aus der CPU laden, auf dem PC sichern und bei Bedarf wieder in die CPU schreiben.
Neben dem Sichern von Daten kann damit auch eine Art Rezeptverwaltung realisiert werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
WldFileWithPath	char*	Angabe der WLD-Datei mit Pfad aus welcher der DB entnommen und in die CPU übertragen werden soll. Angabe z.B. "C:\ProjektDBs.WLD"
DBNr	WORD	Nummer des DBs der in die CPU zu übertragen ist.
OverwriteIfExist	BYTE	Mit diesem Parameter kann bestimmt werden ob ein DB, der bereits in der CPU vorhanden ist, überschrieben werden soll. Bei der Angabe von 1 wird der DB überschrieben, bei der Angabe von 0 wird ein in der CPU vorhandener DB nicht überschrieben und die Funktion kehrt mit dem Error-Wert 713 zurück.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird aus einer WLD-Datei die sich im Pfad "D:\MPI6_Testprojekt\" befindet und die Bezeichnung "Test_Datei.wld" besitzt, der DB2 gelesen und in die CPU übertragen. Sollte der DB2 in der CPU bereits vorhanden sein, so ist dieser zu überschreiben.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 WORD Error=0;
 char ErrorString[255]={0};
 char WldFileWithPath[555]={0};
 BYTE OverwriteIfExist=1; //einen in der CPU vorhandenen DB
                          //überschreiben
 WORD wDBNR=2; //Es soll der DB2 übertragen werden
 //Angabe der WLD-Datei mit Pfad
 strcpy(WldFileWithPath, "D:\\MPI6_Testprojekt\\Test_Datei.wld");
 //Übertragen des DB aus der WLD-Datei in die CPU ausführen
 if (!MPI6_WriteDBFromWldToPlc(MPIHandleV6, WldFileWithPath, wDBNR,
                               OverwriteIfExist, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(AppHandle, "Schreiben des DBs war erfolgreich.", "",
                MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.34 Die Funktion: MPI6_ReadDBFromPlcAndWriteToWld (Nicht in den Lite- und CE-Versionen)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadDBFromPlcAndWriteToWld kann dazu verwendet werden, einen Datenbaustein aus einer CPU zu laden und auf dem PC in einer sog. WLD-Datei zu speichern. S7-Programmiersysteme (z.B. Simatic®-Manager oder WinSPS-S7) können Bausteine aus WLD-Dateien lesen und bearbeiten.
Zusammen mit der Funktion MPI6_WriteDBFromWldToPlc kann man mit ComDrvS7 Datenbausteine aus der CPU laden, auf dem PC sichern und bei Bedarf wieder in die CPU schreiben.
Neben dem Sichern von Daten kann damit auch eine Art Rezeptverwaltung realisiert werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
WldFileWithPath	char*	Angabe der WLD-Datei mit Pfad in welche der DB aus der CPU geschrieben werden soll. Der Pfad muss existieren, die WLD-Datei wird erzeugt, wenn diese nicht vorhanden ist. Angabe z.B. "C:\ProjektDBs.WLD". Bei einer vorhandenen WLD-Datei darf sich noch kein DB mit der gleichen Nummer in der Datei befinden. Ist dies der Fall kehrt die Funktion mit der Error-Wert 717 zurück.
DBNr	WORD	Nummer des DBs der aus der CPU geladen und in die WLD-Datei geschrieben werden soll.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird aus der CPU der DB2 gelesen und in eine WLD-Datei welche sich im Pfad "D:\MPI6_Testprojekt\" befindet und die Bezeichnung "Test_Datei.wld" besitzt, geschrieben.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 WORD Error=0;
 char ErrorString[255]={0};
 char WldFileWithPath[555]={0};
 WORD wDBNR=2; //Es soll der DB2 aus der CPU geholt werden
 //Angabe der WLD-Datei mit Pfad
 strcpy(WldFileWithPath, "D:\\MPI6_Testprojekt\\Test_Datei.wld");
 //Lesen des DBs aus der CPU und ablegen in die WLD-Datei ausführen
 if (!MPI6_ReadDBFromPlcAndWriteToWld(MPIHandleV6, WldFileWithPath,
                                      wDBNR, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(Application->Handle, "Lesen des DBs aus der CPU"
                "und ablegen in der WLD war erfolgreich.", "",
               MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.35 Die Funktion: MPI6_GetDBNrInWldFile (Nicht in den Lite- und CE-Versionen)

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetDBNrInWldFile kann dazu verwendet werden, die in einer WLD-Datei vorhandenen Datenbausteine zu ermitteln.
WLD-Dateien können von S7-Programmiersystemen (z.B. Simativ-Manager und WinSPS-S7) erzeugt und mit Bausteinen gefüllt werden. Ebenso können die S7-Programmiersysteme Bausteine aus WLD-Dateien lesen und bearbeiten.
Mit den Funktionen MPI6_ReadDBFromPlcAndWriteToWld und MPI6_WriteDBFromWldToPlc kann man mit ComDrvS7 Datenbausteine aus einer CPU laden, auf dem PC sichern und bei Bedarf wieder in die CPU schreiben.
Neben dem Sichern von Daten kann damit auch eine Art Rezeptverwaltung realisiert werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
WldFileWithPath	char*	Angabe der WLD-Datei mit Pfad aus welcher die vorhandenen DBs ermittelt werden sollen. Die Datei muss existieren.
DBNrArray	WORD*	In diesem Array werden die Nummern der DBs innerhalb der WLD-Datei geliefert. Das Array muss mind. so viele Felder besitzen, wie im Parameter MaxDBNumbers angegeben.
wcountDB	WORD*	Anzahl der in der WLD-Datei gefundenen Datenbausteine.
MaxDBNumbers	WORD	Maximale Anzahl der zu liefernden DB-Nummern. Für diese Anzahl muss das Array DBNrArray mindestens ausgelegt sein.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden die vorhandenen Datenbausteine aus der WLD-Datei welche sich im Pfad "D:\MPI6_Testprojekt\" befindet und die Bezeichnung "Test_Datei.wld" hat, ermittelt.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde.

```
.
.
.
 //
 WORD Error=0;
 char ErrorString[255]={0};
 char WldFileWithPath[555]={0};
 //Angabe der WLD-Datei mit Pfad
 strcpy(WldFileWithPath, "D:\\MPI6_Testprojekt\\Test_Datei.wld");
 //Lesen des DBs aus der CPU und ablegen in die WLD-Datei ausführen
 WORD wcountDB=0; //In dieser Variablen wird die Anzahl der DBs
                  //geliefert
 WORD DBNrArray[100]; //Array für die vorhandenen DB-Nummern
 WORD MaxDBNumbers=100; //Das Array hat Platz für max. 100 DB-Nummern
 if (!MPI6_GetDBNrInWldFile(MPIHandleV6, WldFileWithPath, DBNrArray,
                            &wcountDB, MaxDBNumbers, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     char AusgabeStr[255]={0};
     wsprintf(AusgabeStr, "In der WLD-Datei sind %u DBs vorhanden.",
              wcountDB);
     MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
 }//ende else
 //
.
.
.
```

6.36 Die Funktion: MPI6_ReadPlcClock

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_ReadPlcClock kann das eingestellte Datum und die Uhrzeit der CPU ausgelesen werden.
Das Datum und die Uhrzeit wird dabei im Format Date-and-Time geliefert. Des Weiteren wird ein String mit der Angabe von der Funktion bereit gestellt.

Das Format Date-and-Time hat folgenden Aufbau:

Byte-Stelle	Bedeutung
n	Jahr 0-99
n+1	Monat 1-12
n+2	Tag 1-31
n+3	Stunde 0-23
n+4	Minute 0-59
n+5	Sekunde 0-59
n+6	Millisekunde 0-999 + Wochentag (1-7)
n+7	

Alle Daten sind BCD-codiert. Wochentag: 1=Sonntag, 7=Samstag.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
byteBufferPtr	BYTE*	Array in welchem das Datum und die Uhrzeit der CPU im Format Date-and-Time geliefert wird. Das Array muss mind. 8 Bytes lang sein.
DtString	char*	Hier wird das Datum und die Uhrzeit als String in der Form "2009-02-25-21:58:11.478" geliefert.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Albert-Einstein-Str. 101 • 75015 Bretten • Tel: 07252-84696 oder 87890 • Fax: 78780

Beispiel

Im folgenden Beispiel wird aus der CPU das eingestellte Datum und die Uhrzeit ausgelesen.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //
 BYTE byteBufferPtr[8]={0}; //Buffer für Format Date-and-Time
 char DTStr[255]={0}; //Gelieferte Einstellung als String
 //
 if (!MPI6_ReadPlcClock(MPIHandleV6, byteBufferPtr, DTStr, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     char AusgabeStr[255]={0};
     wsprintf(AusgabeStr, "Datum Uhrzeit in der CPU: %s", DTStr);
     MessageBox(ApplHandle, AusgabeStr, "", MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.37 Die Funktion: MPI6_WritePlcClock

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Mit der Funktion MPI6_WritePlcClock kann das Datum und die Uhrzeit in der CPU eingestellt werden.
Das Datum und die Uhrzeit muss dabei im Format Date-and-Time vorgegeben werden.

Das Format Date-and-Time hat folgenden Aufbau:

Byte-Stelle	Bedeutung
n	Jahr 0-99
n+1	Monat 1-12
n+2	Tag 1-31
n+3	Stunde 0-23
n+4	Minute 0-59
n+5	Sekunde 0-59
n+6	Millisekunde 0-999
n+7	+ Wochentag (1-7)

Alle Daten sind BCD-codiert. Wochentag: 1=Sonntag, 7=Samstag.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
byteBufferPtr	BYTE*	Array in dem das Datum und die Uhrzeit der CPU im Format Date-and-Time anzugeben ist. Das Array muss mind. 8 Bytes lang sein.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird die CPU auf das Datum 31.5.2009 und die Uhrzeit auf 12.33 Uhr 10 Sekunden und 333 Millisekunden gesetzt.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //
 BYTE byteBufferPtr[8]={0};
 byteBufferPtr[0]=0x09; //Jahr 2009
 byteBufferPtr[1]=0x05; //Monat 5
 byteBufferPtr[2]=0x31; //Tag 31
 byteBufferPtr[3]=0x12; //Stunde 12
 byteBufferPtr[4]=0x33; //Minute 33
 byteBufferPtr[5]=0x10; //Sekunde 10
 byteBufferPtr[6]=0x33; //Millisekunden 333 Angabe 1
 byteBufferPtr[7]=0x31; //Millisekunden 333 Angabe 2 Tag = 1
 //
 if (!MPI6_WritePlcClock(MPIHandleV6, byteBufferPtr, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "Datum und Uhrzeit geschrieben.", "",
                MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.38 Die Funktion: MPI6_CopyRamToRom

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Mit der Funktion MPI6_CopyRamToRom können die Aktualwerte der Datenbausteine einer CPU aus dem Arbeitsspeicher in den Ladespeicher übertragen werden. Dadurch haben diese Werte auch nach dem Urlöschen der CPU noch ihre Gültigkeit.
Die Funktion wird z.B. ausgeführt, wenn man Werte eines DBs durch SPS-Befehle oder durch Steuern über ComDrvS7 verändert hat und diese Werte auch nach dem Urlöschen Bestand haben sollen.

Die Funktion kann nur ausgeführt werden, wenn sich die CPU in STOP befindet. Sollte die CPU nicht in STOP sein, so kann diese über die Funktion MPI6_SetPLCToStop in STOP überführt werden.

Die Ausführung Funktion kann einige Minuten andauern (je nach Speicherbelegung).

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird bei der CPU die Funktion RAM-nach-ROM kopieren ausgeführt. Zuvor wird sichergestellt, dass die CPU im Zustand STOP ist.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //Ist die CPU in RUN
 bool PlcInRun=false;
 if (!MPI6_IsPLCInRunMode(MPIHandleV6, &PlcInRun, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
 }//ende if
 //
 if (PlcInRun){
     MessageBox(ApplHandle, "Aktion in RUN nicht möglich!", "",
                MB_ICONINFORMATION);
     return;
 }//ende if
 //
 if (!MPI6_CopyRamToRom(MPIHandleV6, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "Aktion wurde ausgeführt.", "",
                MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.39 Die Funktion: MPI6_PLCHotRestart bzw. MPI6_CPUWiederanlauf

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktionen MPI6_PLCHotRestart bzw. MPI6_CPUWiederanlauf lösen beide einen Wiederanlauf der CPU aus. Vorraussetzung die CPU unterstützt den Wiederanlauf (S7-400 mit entsprechender Hardwarekonfiguration) und der Betriebsartenschalter ist in der Stellung RUN.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird bei der CPU ein Wiederanlauf ausgelöst.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
 char ErrorString[255]={0};
 WORD Error=0;
 //
 if (!MPI6_CPUWiederanlauf(MPIHandleV6, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "Aktion wurde ausgeführt.", "",
                MB_ICONINFORMATION);
 }//ende else
```

6.40 Die Funktion: MPI6_PLCWarmRestart bzw. MPI6_CPUNeustart

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktionen MPI6_PLCWarmRestart bzw. MPI6_CPUNeustart lösen beide einen Neustart der CPU aus. Vorrausgesetzt der Betriebsartenschalter ist in der Stellung RUN.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird bei der CPU ein Neustart ausgelöst.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
 char ErrorString[255]={0};
 WORD Error=0;
 //
 if (!MPI6_CPUNeustart(MPIHandleV6, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "Aktion wurde ausgeführt.", "",
                MB_ICONINFORMATION);
 }//ende else
```

6.41 Die Funktion: MPI6_SetPLCToStop

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktionen MPI6_SetPLCToStop versetzt die CPU in den STOP-Zustand.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird die CPU in den Zustand STOP versetzt.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
 char ErrorString[255]={0};
 WORD Error=0;
 //
 if (!MPI6_SetPLCToStop(MPIHandleV6, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "Aktion wurde ausgeführt.", "",
                MB_ICONINFORMATION);
 }//ende else
.
```

6.42 Die Funktion: MPI6_IsPLCInRunMode

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_IsPLCInRunMode ermittelt den Betriebszustand der CPU und liefert im Parameter PlcInRun den Wert '1', wenn sich die CPU in RUN befindet.

Familie LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PlcInRun	BOOL*	Hat der Parameter den Wert '1', so befindet sich die CPU im Zustand RUN.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird überprüft ob sich die CPU im RUN-Zustand befindet.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //Ist die CPU in RUN
 bool PlcInRun=false;
 if (!MPI6_IsPLCInRunMode(MPIHandleV6, &PlcInRun, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
 }//ende if
 //
 if (PlcInRun){
     MessageBox(ApplHandle, "CPU ist in RUN!", "",
                MB_ICONINFORMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "CPU ist nicht in RUN!", "",
                MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.43 Die Funktion: MPI6_GetSystemValues

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetSystemValues kann dazu verwendet werden, die Systembereiche einer CPU auszulesen. Man erhält z.B. die Information wieviele Zeiten, Zähler und Merker usw. programmierbar sind.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
CountByteProcessImageInputs	WORD*	Hier wird die Anzahl der Byte des Prozeßabbildes der Eingänge geliefert.
CountByteProcessImageOutputs	WORD*	Hier wird die Anzahl der Byte des Prozeßabbildes der Ausgänge geliefert.
CountByteBitMemory	WORD*	Hier wird die Anzahl der verfügbaren Merkerbytes im angeschlossenen AG geliefert.
CountTimer	WORD*	Hier wird die Anzahl der verfügbaren Zeiten im angeschlossenen AG geliefert.
CountCounter	WORD*	Hier wird die Anzahl der verfügbaren Zähler im angeschlossenen AG geliefert.
CountByteWorkMemory	DWORD*	Liefert die Anzahl der Bytes des Arbeitsspeichers der CPU. **Achtung: In den alten Versionen von ComDrvS7 war dieser Parameter vom Typ WORD also nur 16 Bit breit!**
CountByteLocalData	WORD*	Liefert den gesamten Lokaldatenbereich der CPU in Byte.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden die Systembereiche des Kommunikationspartners ausgelesen. Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
     WORD AnzahlBytePAE, AnzahlBytePAA, AnzahlMerker, AnzahlZeiten;
     WORD AnzahlZaehler;
     DWORD AnzahlByteRam;
     WORD AnzahlByteLokaldaten;
     //
     if (!MPI6_GetSystemValues(MPIHandle, &AnzahlBytePAE,
                               &AnzahlBytePAA,
                                 &AnzahlMerker, &AnzahlZeiten,
                                &AnzahlZaehler,
                                &AnzahlByteRam,
                                &AnzahlByteLokaldaten,
                                 &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
          wsprintf(AusgabeStr,
                "Anzahl Byte PAE: %03u\n"
                "Anzahl Byte PAA: %03u\n"
                "Anzahl Merker  : %03u\n"
                "Anzahl Zeiten  : %03u\n"
                "Anzahl Zähler  : %03u\n",
                AnzahlBytePAE, AnzahlBytePAA, AnzahlMerker,
                AnzahlZeiten, AnzahlZaehler);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONEXCLAMATION);
    }//ende else
}//ende if
.
.
.
```

Anmerkung:

Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

6.44 Die Funktion: MPI6_GetLevelOfProtection

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetLevelOfProtection kann dazu verwendet werden, die eingestellten Schutzstufen einer CPU zu ermitteln. Es wird ebenfalls die Stellung des Schlüsselschalters auf der CPU geliefert.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
LevelModeSwitch	WORD*	Schutzstufe am Betriebsartenschalter/Schlüsselschalter (mögliche Werte: 1,2 oder 3)
ParameterizedProtectionLevel	WORD*	parametrierte Schutzstufe (mögliche Werte: 0, 1, 2 oder 3. 0 bedeutet: kein Paßwort vergeben, parametrierte Schutzstufe ist ungültig)
LevelPlc	WORD*	Gültige Schutzstufe der CPU (mögliche Werte: 1, 2 oder 3)
ModeSwitchPosition	WORD*	Liefert die Stellung des Betriebsartenschalters/Schlüsselschalters der CPU. Mögliche Inhalte sind: 0=STOP, 1=RUN, 2=RUN-P.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls

auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

Beispiel

Im folgenden Beispiel wird die Stellung des Schlüsselschalters auf dem Kommunikationspartner als String ausgegeben.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
     WORD SchutzstufeSchluesselSchalter, ParametrierteSchutzstufe;
     WORD CPUSchutzstufe;
     WORD SchluesselschalterStellung=0;
     //
     if (!MPI6_GetLevelOfProtection(MPIHandle,
                                    &SchutzstufeSchluesselSchalter,
                                    &ParametrierteSchutzstufe,
                               &CPUSchutzstufe,
                              &SchluesselschalterStellung,
                             &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         switch(SchluesselschalterStellung){
                case 0: MessageBox(AppHandle, "Schalterstellung:
                                   STOP", "", MB_ICONEXCLAMATION);
                        break;
                case 1: MessageBox(AppHandle, "Schalterstellung:"
                                   "RUN", "", MB_ICONEXCLAMATION);
                        break;
                case 2: MessageBox(AppHandle, "Schalterstellung:"
                                   "RUN-P", "", MB_ICONEXCLAMATION);
                                   break;
                default: MessageBox(AppHandle, "Schalterstellung:"
                                    "Unbekannt", "",
                                    MB_ICONEXCLAMATION);
         }//ende switch
    }//ende else
}//ende if
.
.
.
```

6.45 Die Funktion: MPI6_GetOrderNrPlc

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetOrderNrPlc kann dazu verwendet werden, die Bestellnummer einer CPU zu ermitteln.

Familie LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
OrderNrStr	char*	Hierbei wird die Bestellnummer der CPU als nullterminierter String geliefert.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

Beispiel

Im folgenden Beispiel wird die Bestellnummer einer CPU ermittelt.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
     char BestellNr[100]={0};
     //
     if (!MPI6_GetOrderNrPlc(MPIHandle, BestellNr, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "BestellNr.: %s", BestellNr);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.46 Die Funktion: MPI6_CanPlcSendIdentData

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_CanPlcSendIdentData kann dazu verwendet werden, zu ermitteln, ob die angeschlossene CPU das Liefern der Identifikationsdaten unterstützt. Die Funktion sollte somit vor der Funktion MPI6_GetPlcIdentData aufgerufen werden, wenn nicht sichergestellt ist, dass die angeschlossene CPU dieses Leistungsmerkmal unterstützt. Bei den Siemens CPUs der Reihe S7-300® wird diese Funktion ab dem Firmwarestand 2.6 unterstützt.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PlcCanSendData	bool*	Liefert den Wert 1, wenn die CPU die Abfrage der Identifikationsdaten unterstützt. Anderenfalls wird der Wert 0 geliefert.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

Beispiel

Siehe Beispiel zur Funktion MPI6_GetPlcIdentData

6.47 Die Funktion: MPI6_GetPlcIdentData

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Besteht ein Zweifel daran, ob die angeschlossene CPU die Identifikationsdaten liefern kann, so muss zuvor die Funktion MPI6_CanPlcSendIdentData ausgeführt werden. Bei den Siemens CPUs der Reihe S7-300® wird dieses Leistungsmerkmal ab dem Firmwarestand 2.6 unterstützt.

Kurzbeschreibung

Die Funktion MPI6_GetPlcIdentData liefert die eindeutige Seriennummer der angeschlossenen CPU, sowie die Seriennummer der in der CPU vorhandenen MMC-Karte. Somit ist eine eindeutige Identifizierung der angeschlossenen CPU möglich. Des Weiteren können die in der Hardwarekonfiguration der CPU definierbaren Texte für den Namen der CPU, den Namen der Station, die Anlagenkennzeichnung und die Ortskennzeichnung ausgelesen werden.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PlcName	char*	Liefert den in der Hardwarekonfiguration für die CPU definierten Namen der Station oder einen leeren String wenn dies nicht definiert wurde.
ModuleName	char*	Liefert den in der Hardwarekonfiguration für die CPU definierten Namen der CPU oder einen leeren String wenn dies nicht definiert wurde.
PlantDesignation	char*	Liefert die in der Hardwarekonfiguration für die CPU definierte Anlagenkennzeichnung oder einen leeren String wenn dies nicht definiert wurde.
LocationIdentifier	char*	Liefert die in der Hardwarekonfiguration für die CPU definierte Ortskennzeichnung oder einen leeren String wenn dies nicht definiert wurde.
SerialNrPlcStr	char*	Liefert die Seriennummer der angeschlossenen CPU.
MMCIdentNrStr	char*	Liefert die Seriennummer der in der CPU gesteckten MMC-Karte.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

Beispiel

Im folgenden Beispiel werden die Identifikationsdaten der CPU ausgelesen, sofern die CPU dies unterstützt. Die Unterstützung wird dabei zunächst abgefragt und bei einer positiven Antwort die Daten aus der CPU geladen.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
bool AbfrageMoeglich=false;
if (!MPI6_CanPlcSendIdentData(MPIHandle, &AbfrageMoeglich, &Error)){
    //Fehler anzeigen
    MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
    MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
//
if (AbfrageMoeglich){
   char NameDerStation[100]={0};
   char NameDerCPU[100]={0};
   char AnlagenKennzeichnung[100]={0};
   char OrtsKennzeichnung[100]={0};
   char SerienNummerCPU[100]={0};
   char MMCIdentNr[100]={0};
   //
   if (!MPI6_GetPlcIdentData(MPIHandle, NameDerStation, NameDerCPU,
                             AnlagenKennzeichnung, OrtsKennzeichnung,
                             SerienNummerCPU, MMCIdentNr, &Error)){
       //Fehler anzeigen
       MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
       MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
   }//ende if
   else {
       char AusgabeStr[255]={0};
       wsprintf(AusgabeStr, "Seriennummer der CPU: %s",
               SerienNummerCPU);
       MessageBox(Handle, AusgabeStr, "", MB_ICONINFORMATION);
   }//ende else
}//ende if
.
.
.
```

6.48 Die Funktion: MPI6_GetPlcErrorLED

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetPlcErrorLED liefert den Status der Fehler LEDs SF, BF1 und BF2. Damit kann ermittelt werden, ob ein Sammelfehler bzw. ein Busfehler an der CPU ansteht. Der Programmierer kann dann im PC-Programm entsprechend reagieren. Die CPU kann sich trotz dieser Fehler im RUN-Status befinden, wenn im SPS-Programm die jeweiligen Fehler-OBs vorhanden sind.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
SF_LED_Status	BYTE*	Hat den Wert 1, wenn die SF-LED auf der CPU leuchtet oder blinkt.
SF_LED_FlashingFrequency	BYTE*	0 = Dauerlicht wenn SF_LED_Status auf 1 1 = blinkt normal mit 2 Hz 2 = blinkt langsam mit 0.5 Hz
BUS1F_LED_Status	BYTE*	Hat den Wert 1, wenn die BF1-LED auf der CPU leuchtet oder blinkt.
BUS1F_LED_FlashingFrequency	BYTE*	0 = Dauerlicht wenn BUS1F_LED_Status auf 1 1 = blinkt normal mit 2 Hz 2 = blinkt langsam mit 0.5 Hz
BUS2F_LED_Status	BYTE*	Hat den Wert 1, wenn die BF2-LED auf der CPU leuchtet oder blinkt.
BUS2F_LED_FlashingFrequency	BYTE*	0 = Dauerlicht wenn BUS2F_LED_Status auf 1 1 = blinkt normal mit 2 Hz 2 = blinkt langsam mit 0.5 Hz
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:

Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

Beispiel

Im folgenden Beispiel wird der Status der Fehler-LEDs auf der CPU gelesen.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
BYTE SF_LED_Status=0;
BYTE SF_LED_BlinkFrequenz=0;
BYTE BUS1F_LED_Status=0;
BYTE BUS1F_LED_BlinkFrequenz=0;
BYTE BUS2F_LED_Status=0;
BYTE BUS2F_LED_BlinkFrequenz=0;
//
if (!MPI6_GetPlcErrorLED(MPIHandle,
                        &SF_LED_Status, &SF_LED_BlinkFrequenz,
                        &BUS1F_LED_Status, &BUS1F_LED_BlinkFrequenz,
                        &BUS2F_LED_Status, &BUS2F_LED_BlinkFrequenz,
                        &Error)){
    //Fehler anzeigen
    MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
    MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
else {
    char AusgabeStr[255]={0};
    wsprintf(AusgabeStr, "Status der SF-LED: %s", SF_LED_Status);
    MessageBox(Handle, AusgabeStr, "", MB_ICONINFORMATION);
}//ende else
.
.
.
```

6.49 Die Funktion: MPI6_IsPasswordRequired

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein. Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Über die Funktion MPI6_IsPasswordRequired kann ermittelt werden, ob für Lese- und/oder Schreiboperationen zur CPU hin, ein Passwort notwendig ist, die CPU also über einen Passwortschutz verfügt. Die Funktion muss verwendet werden, wenn man nicht sicher ist, ob ein Passwort zu übergeben ist. Wird ein Passwort an die CPU übergeben wobei eigentlich kein Passwortschutz besteht, so kommt es zu einem Fehler.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Anmerkung

Bei den ComDrvS7-Funktionen ReadByte, ReadWord, ReadDword, ReadTimer, ReadCounter, WriteByte, WriteWord, WriteDword, WriteTimer, WriteCounter, MixRead_2 und MixWrite_2 ist das Übergeben eines Passwortes nicht notwendig.

Beschreibung der Parameter

Argument	Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PasswordRequired	bool*	Liefert den Wert 1 wenn für den angegebenen Modus ein Passwort notwendig ist. Liefert 0 wenn für den Modus kein Passwort notwendig ist.
Mode	BYTE	Übergabe von 'R' (bzw. dem ASCII-Code 82 dez.) für Abfrage des Lesens. Übergabe von 'W' (bzw. dem ASCII-Code 87 dez.) für Abfrage des Schreibens.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung:
Die Daten für diese Auskunftsfunktion werden von der CPU virtuell zusammengestellt. Dazu ist die CPU nur einmal "parallel" in der Lage. Greift eine weitere Kommunikationsinstanz ebenfalls auf eine Auskunftsfunktion der gleichen CPU zu, so kann dies zu einem Kommunikationsfehler führen.

Beispiel: Siehe Beispiel zu MPI6_SendPasswordToPlc.

6.50 Die Funktion: MPI6_SendPasswordToPlc

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein. Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Über die Funktion MPI6_SendPasswordToPlc kann ein Passwort an die CPU übergeben werden um das Lesen/Schreiben bei einer passwortgeschützten CPU zu ermöglichen. Ist nicht sicher, ob bei der CPU ein Passwortschutz für den Zugriffsmodus besteht, so muss dies zunächst über die Funktion MPI6_IsPasswordRequired ermittelt werden. Ist es sicher, dass ein Passwort für die Zugriffsart besteht, so kann auf die Funktion MPI6_IsPasswordRequired verzichtet werden.

Wichtig!

Das übergebene Passwort gilt so lange, bis die Verbindung zur CPU wieder abgebaut wird. Dies bedeutet, wird beispielsweise mehrmals schreibend auf die passwortgeschützte CPU zugegriffen, so muss das Passwort nur einmalig übergeben werden. Sofern zwischen den Schreibzugriffen nicht die Kommunikation zur CPU abgebaut wird.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Anmerkung

Bei den ComDrvS7-Funktionen ReadByte, ReadWord, ReadDword, ReadTimer, ReadCounter, WriteByte, WriteWord, WriteDword, WriteTimer, WriteCounter, MixRead_2 und MixWrite_2 ist das Übergeben eines Passwortes nicht notwendig.

Beschreibung der Parameter

Argument	Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
PasswordStr	char*	Passwort (**mit max. 8 Zeichen**) mit welchem die CPU geschützt ist. Der Passwortschutz und das Passwort selbst wird in der Hardwarekonfiguration der CPU eingestellt.
PasswordIsCorrect	bool*	Liefert 1 wenn das übergeben Passwort korrekt ist. Liefert 0 bei einem falschen Passwort.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird geprüft, ob die CPU über einen Schreibschutz verfügt. Danach wird das Passwort an die CPU übergeben.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
bool PasswortUebergabeNotwendig=false;
BYTE Modus='W'; //Schreibmodus
if (!MPI6_IsPasswordRequired(MPIHandle,
                        &PasswortUebergabeNotwendig, Modus, &Error)){

    //Fehler anzeigen
    MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
    MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
if (PasswortUebergabeNotwendig){
   //
   char PasswortStr[20]={0};
   bool PasswortIstKorrekt=false;
   strcpy(PasswortStr, "Test"); //CPU ist auf das Passwort 'Test'
                                //eingestellt
   if (!MPI6_SendPasswordToPlc(MPIHandle, PasswortStr,
                            &PasswortIstKorrekt, &Error)){
      //Fehler anzeigen
      MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
      MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
   }//ende if
   else {
      if (PasswortIstKorrekt)
          MessageBox(Handle, "Das Passwort ist korrekt.", "",
                     MB_ICONINFORMATION);
      else
          MessageBox(Handle, "Das Passwort ist falsch!", "",
                     MB_ICONSTOP);
   }//ende else
}//ende if
.
.
.
```

6.51 Die Funktion: MPI6_GetCountDB

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetCountDB kann dazu verwendet werden, die Anzahl der vorhandenen DBs in einer CPU zu ermitteln.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
CountDB	int*	In diesem Parameter wird die Anzahl der in der CPU vorhandenen DBs geliefert.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird die Anzahl der Datenbausteine im angeschlossenen Kommunikationspartner ermittelt.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
     int AnzahlDB=0;
     //
     if (!MPI6_GetCountDB(MPIHandle, &AnzahlDB, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "Anzahl vorhandene DBs: %u", AnzahlDB);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.52 Die Funktion: MPI6_GetDBInPlc

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetDBInPlc kann dazu verwendet werden, die Nummern der in der CPU vorhandenen Datenbausteine zu ermitteln.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
DBNumbers	WORD*	WORD-Array, in welchem die in der CPU vorhandenen DB-Nummern geliefert werden.
CountDB	INT*	Anzahl der im WORD-Array eingetragenen DBs.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird die Nummer des ersten in der CPU vorhandenen DBs angezeigt. Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
     int AnzahlDB=0;
     WORD DBNummern[255]={0};
     //
     if (!MPI6_GetDBInPlc(MPIHandle, DBNummern, &AnzahlDB, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         if (AnzahlDB>0){
             wsprintf(AusgabeStr, "Erster vorhandener DB: %u",
                     DBNummern[0]);
             MessageBox(AppHandle, AusgabeStr, "",
                        MB_ICONINFORMATION);
         }//ende if
         else
             MessageBox(AppHandle, "Keine DBs vorhanden!", "",
                        MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.53 Die Funktion: MPI6_GetLengthDB

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.
Des Weiteren muss die Funktion MPI6_ConnectToPLC oder MPI6_ConnectToPLCRouting ebenfalls erfolgreich aufgerufen worden sein.

Kurzbeschreibung

Die Funktion MPI6_GetLengthDB kann dazu verwendet werden, die Länge eines in der CPU vorhandenen Datenbausteins zu ermitteln. Die Länge wird dabei in Bytes geliefert.

Familie S7-1500®, S7-1200® und LOGO!®

Die Funktion kann nicht bei den CPUs der Reihe S7-1500®, S7-1200® oder LOGO!® verwendet werden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
DBNr	WORD	Angabe der Nummer des DBs, von welchem die Länge ermittelt werden soll.
CountByte	WORD*	Länge des DBs in Bytes.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel wird die Länge des DB1 im Kommunikatiosnpartner ermittelt.
Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
//Daten lesen
if (!Fehler){
     WORD LaengeInByte=0;
     //
     if (!MPI6_GetLengthDB(MPIHandle, 1, &LaengeInByte, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "Länge des DB1: %u Byte", LaengeInByte);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
.
.
.
```

6.54 Die Funktion: MPI6_ChangeProtocolTypeForV5Functions

Voraussetzung zum Ausführen der Funktion

Eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp usw.) muss erfolgreich ausgeführt worden sein.

Kurzbeschreibung

Die Funktion MPI6_ChangeProtocolTypeForV5Functions kann dazu verwendet werden, die alten Lese- und Schreibfunktionen der Version ComDrvS7 V5 auf den neuen, schnelleren Protokolltyp umzustellen. Dazu wird die Funktion einmalig nach den Einleitungsfunktionen ausgeführt, wobei der Parameter TakeV6Protocol den Wert 1 haben muss. Damit können Anwender älterer ComDrvS7-Versionen nur durch den Aufruf dieser Funktion, von den Neuerungen der ComDrvS7 V6 profitieren.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Handle	INT	Das Handle der Kommunikationsinstanz welche angesprochen wird (entfällt bei .Net-Wrapper-Klasse).
TakeV6Protocol	BYTE	Bei Übergabe des Wertes 1 wird auch bei den alten Lese- und Schreibfunktionen (z.B. MPI_A_ReadMerkerByte) das neue Protokoll verwendet. Wird die Funktion aufgerufen und der Parameter hat den Wert 0, so werden wieder die alten Funktionen verwendet.
Error	WORD*	Liefert die Funktion den Wert '0' als Rückgabewert, so ist ein Fehler bei der Ausführung aufgetreten. In diesem Fall wird in dem Parameter Error ein Error-Wert geliefert.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Beispiel

Im folgenden Beispiel werden die Merkerbyte MB0 bis MB99 mit der alten ComDrvS7-Funktion gelesen. Vor dem Aufruf der Funktion MPI_A_ReadMerkerByte wird die Funktion MPI6_ChangeProtocolTypeForV5Functions aufgerufen um auf den neuen Protokolltyp umzustellen.

Im Beispiel wird davon ausgegangen, dass eine der Einleitungsfunktionen (z.B. MPI6_OpenTcpIp) erfolgreich ausgeführt wurde. Ebenso muss die Funktion **MPI6_ConnectToPLC** ohne Fehler ausgeführt worden sein. Nach der Aktion können weitere folgen. Durch Ausführen der Funktion **MPI6_CloseCommunication** kann die Kommunikation beendet und die Instanz beseitigt werden. So wie dies im Beispiel für die Funktion MPI6_ReadByte gezeigt wurde.

```
.
.
.
 char ErrorString[255]={0};
 WORD Error=0;
 //Auf neues V6-Protokoll umstellen
 BYTE TakeV6Protocol=1;
 //
 if (!MPI6_ChangeProtocolTypeForV5Functions(MPIHandleV6,
                                            TakeV6Protocol, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
 }//ende if
 //
 BYTE ByteStatusBuffer[100];
 //Alte Funktion wie gewohnt aufrufen, es wird automatisch das neue
//Protokoll verwendet
 if (!MPI_A_ReadMerkerByte(MPIHandleV6, 0, 100, ByteStatusBuffer,
                           &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandleV6, ErrorString, Error);
     MessageBox(ApplHandle, ErrorString, "", MB_ICONEXCLAMATION);
 }//ende if
 else {
     MessageBox(ApplHandle, "Daten wurden gelesen.", "",
                MB_ICONINFORMATION);
 }//ende else
.
.
.
```

6.55 Die Funktion: MPI6_GetVersionComDrvS7

Voraussetzung zum Ausführen der Funktion

Keine.

Kurzbeschreibung

Die Funktion MPI6_GetVersionComDrvS7 liefert die Versionsnummer der verwendeten ComDrvS7-DLL als String zurück. Dadurch kann einfach überprüft werden, welche Version zur Anwendung kommt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
VersionStr	char*	String mit der Versionsnummer in der Form "ComDrvS7 V6.XX"
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

6.56 Die Funktion: MPI_A_RealFromByteBuffer oder MPI6_RealFromByteBuffer

Kurzbeschreibung

Die Funktion MPI_A_RealFromByteBuffer setzt eine Real-Zahl aus einem Byte-Buffer zusammen. Die Funktion kann z.B. eingesetzt werden, wenn eine Real-Zahl aus einem DB ausgelesen wurde und dabei die Funktion ReadDBByte zum Einsatz kam.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
RealValue	float*	Die Real-Zahl welche aus den ersten 4 Bytes des Buffers ermittelt wurde.
ByteBuffer	BYTE*	Der Buffer dessen erste 4 Bytes eine Real-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Real-Zahl wird aus den Bytes ByteBuffer[0], ByteBuffer[1], ByteBuffer[2] und ByteBuffer[3] zusammengesetzt.

6.57 Die Funktion: MPI_A_RealFromWordBuffer oder MPI6_RealFromWordBuffer

Kurzbeschreibung

Die Funktion MPI_A_RealFromWordBuffer setzt eine Real-Zahl aus einem WORD-Buffer zusammen. Die Funktion kann z.B. eingesetzt werden, wenn eine Real-Zahl aus einem DB ausgelesen wurde und dabei die Funktion ReadDBWort zum Einsatz kam.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
RealValue	float*	Die Real-Zahl welche aus den ersten 2 Worten des Buffers ermittelt wurde.
WordBuffer	WORD*	Der Buffer dessen erste 2 Worte eine Real-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Real-Zahl wird aus den Worten WordBuffer[0] und WordBuffer[1] zusammengesetzt.

6.58 Die Funktion: MPI_A_IntFromByteBuffer oder MPI6_IntFromByteBuffer

Kurzbeschreibung

Die Funktion MPI_A_IntFromByteBuffer setzt eine Int-Zahl (16-Bit) aus einem BYTE-Buffer zusammen. Die Funktion kann z.B. eingesetzt werden, wenn eine Int-Zahl aus einem DB ausgelesen wurde und dabei die Funktion ReadDBByte zum Einsatz kam.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
IntValue	short*	Die Int-Zahl welche aus den ersten 2 Bytes des Buffers ermittelt wurde.
ByteBuffer	BYTE*	Der Buffer dessen erste 2 Bytes eine Int-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Int-Zahl wird aus den Bytes ByteBuffer[0] und ByteBuffer[1] zusammengesetzt.

6.59 Die Funktion: MPI_A_IntFromWordBuffer oder MPI6_IntFromWordBuffer

Kurzbeschreibung

Die Funktion MPI_A_IntFromWordBuffer setzt eine Int-Zahl (16-Bit) aus einem WORD-Buffer zusammen. Die Funktion kann z.B. eingesetzt werden, wenn eine Int-Zahl aus einem DB ausgelesen wurde und dabei die Funktion ReadDBWort zum Einsatz kam.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
IntValue	short*	Die Int-Zahl (16-Bit) welche aus dem ersten Wort des Buffers ermittelt wurde.
WordBuffer	WORD*	Der Buffer dessen erstes Wort eine Int-Zahl enthält.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Int-Zahl wird aus dem Wort WordBuffer[0] ermittelt.

6.60 Die Funktion: MPI_A_DIntFromByteBuffer oder MPI6_DIntFromByteBuffer

Kurzbeschreibung

Die Funktion MPI_A_DIntFromByteBuffer setzt eine DInt-Zahl (32-Bit) aus einem BYTE-Buffer zusammen. Die Funktion kann z.B. eingesetzt werden, wenn eine DInt-Zahl aus einem DB ausgelesen wurde und dabei die Funktion ReadDBByte zum Einsatz kam.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
DIntValue	int*	Die DInt-Zahl welche aus den ersten 4 Bytes des Buffers ermittelt wurde.
ByteBuffer	BYTE*	Der Buffer dessen erste 4 Bytes eine DInt-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die DInt-Zahl wird aus den Bytes ByteBuffer[0], ByteBuffer[1], ByteBuffer[2] und ByteBuffer[1] zusammengesetzt.

6.61 Die Funktion: MPI_A_DIntFromWordBuffer oder MPI6_DIntFromWordBuffer

Kurzbeschreibung

Die Funktion MPI_A_RealFromWordBuffer setzt eine DInt-Zahl (32-Bit) aus einem WORD-Buffer zusammen. Die Funktion kann z.B. eingesetzt werden, wenn eine Real-Zahl aus einem DB ausgelesen wurde und dabei die Funktion ReadDBWort zum Einsatz kam.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
DIntValue	int*	Die DInt-Zahl welche aus den ersten 2 Worten des Buffers ermittelt wurde.
WordBuffer	WORD*	Der Buffer dessen erste 2 Worte eine DInt-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die DInt-Zahl wird aus den Worten WordBuffer[0] und WordBuffer[1] zusammengesetzt.

6.62 Die Funktion: MPI_A_RealToWordBuffer oder MPI6_RealToWordBuffer

Kurzbeschreibung

Die Funktion MPI_A_RealToWordBuffer schreibt eine Real-Zahl in einen WORD-Buffer. Die Funktion kann z.B. eingesetzt werden, wenn eine Real-Zahl in einen DB zu schreiben ist und dabei die Funktion WriteDBWort zum Einsatz kommt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
RealValue	float	Die Real-Zahl welche in die ersten 2 Worte des Buffers geschrieben wird.
WordBuffer	WORD*	Der Buffer dessen erste 2 Worte die Real-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Real-Zahl wird in die Worte WordBuffer[0] und WordBuffer[1] geschrieben.

6.63 Die Funktion: MPI_A_RealToByteBuffer oder MPI6_RealToByteBuffer

Kurzbeschreibung

Die Funktion MPI_A_RealToByteBuffer schreibt eine Real-Zahl in einen BYTE-Buffer. Die Funktion kann z.B. eingesetzt werden, wenn eine Real-Zahl in einen DB zu schreiben ist und dabei die Funktion WriteDBByte zum Einsatz kommt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
RealValue	float	Die Real-Zahl welche in die ersten 4 Bytes des Buffers geschrieben wird.
ByteBuffer	BYTE*	Der Buffer dessen erste 4 Bytes die Real-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Real-Zahl wird in die Bytes ByteBuffer[0], ByteBuffer[1], ByteBuffer[2] und ByteBuffer[3] geschrieben.

6.64 Die Funktion: MPI_A_IntToByteBuffer oder MPI6_IntToByteBuffer

Kurzbeschreibung

Die Funktion MPI_A_IntToByteBuffer schreibt eine Int-Zahl (16-Bit) in einen BYTE-Buffer. Die Funktion kann z.B. eingesetzt werden, wenn eine Int-Zahl in einen DB zu schreiben ist und dabei die Funktion WriteDBByte zum Einsatz kommt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
IntValue	short	Die Int-Zahl welche in die ersten 2 Bytes des Buffers geschrieben wird.
ByteBuffer	BYTE*	Der Buffer dessen erste 2 Bytes die Int- Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die Int-Zahl wird in die Bytes ByteBuffer[0] und ByteBuffer[1] geschrieben.

6.65 Die Funktion: MPI_A_DIntToByteBuffer oder MPI6_DIntToByteBuffer

Kurzbeschreibung

Die Funktion MPI_A_DIntToByteBuffer schreibt eine DInt-Zahl (32-Bit) in einen BYTE-Buffer. Die Funktion kann z.B. eingesetzt werden, wenn eine DInt-Zahl in einen DB zu schreiben ist und dabei die Funktion WriteDBByte zum Einsatz kommt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
DIntValue	int	Die DInt-Zahl welche in die ersten 4 Bytes des Buffers geschrieben wird.
ByteBuffer	BYTE*	Der Buffer dessen erste 4 Bytes die DInt- Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die DInt-Zahl wird in die Bytes ByteBuffer[0], ByteBuffer[1], ByteBuffer[2] und ByteBuffer[3] geschrieben.

6.66 Die Funktion: MPI_A_DIntToWordBuffer oder MPI6_DIntToWordBuffer

Kurzbeschreibung

Die Funktion MPI_A_DIntToWordBuffer schreibt eine DInt-Zahl (32-Bit) in einen WORD-Buffer. Die Funktion kann z.B. eingesetzt werden, wenn eine DInt-Zahl in einen DB zu schreiben ist und dabei die Funktion WriteDBByte zum Einsatz kommt.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
DIntValue	int	Die DInt-Zahl welche in die ersten 2 Worte des Buffers geschrieben wird.
WordBuffer	WORD*	Der Buffer dessen erste 2 Worte die DInt-Zahl enthalten.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

Anmerkung

Die DInt-Zahl wird in die Worte WordBuffer[0] und WordBuffer[1] geschrieben.

6.67 Die Funktion: MPI6_BcdToDecimal

Kurzbeschreibung

Die Funktion MPI6_BcdToDecimal wandelt eine BCD-Zahl in eine decimale Zahl um. Die BCD-Zahl muss sich im Bereich 0-99 befinden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Bcd	BYTE	BCD-Zahl im Bereich 0-99
Dec	BYTE*	Gelieferte dezimale Zahl.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

6.68 Die Funktion: MPI6_DecimalToBcd

Kurzbeschreibung

Die Funktion MPI6_DecimalToBcd wandelt eine dezimale in eine BCD-Zahl um. Die dezimale Zahl muss sich im Bereich 0-99 befinden.

Beschreibung der Parameter

Argument	C-Typ	Beschreibung
Dec	BYTE	Dezimale Zahl im Bereich 0-99
BCD	BYTE*	Gelieferte BCD-Zahl.
Funktionswert	BOOL	Wurde die Funktion erfolgreich ausgeführt, so wird der Wert '1' (TRUE) geliefert. Bei einem Fehler ist der Rückgabewert '0' (FALSE).

7 Mehrere Teilnehmer über eine serielle Schnittstelle ansprechen.

Nachfolgend soll an einem Beispiel gezeigt werden, wie zwei CPUs, welche sich in einem MPI-Netz befinden, über eine serielle Schnittstelle angesprochen werden können.
Man muss dabei beachten, dass Kommunikationsinstanzen, welche sich eine Kommunikationsressource teilen, nicht in unterschiedlichen Threads bearbeitet werden dürfen. Die Funktionen der Kommunikationsinstanzen müssen hintereinander in einem Thread bearbeitet werden.

7.1 Einleitungen ausführen

Zunächst ist für jede Verbindung eine Einleitung auszuführen.
Nachfolgend sind auch die Variablen zu sehen. Die Schnittstellenparameter sind dabei nur ein Mal vorhanden, denn die Baudrate wird bei der ersten Einleitung festgelegt. Wird die zweite Einleitung mit der gleichen COM-Schnittstelle aber anderen Schnittstellenparametern (z.B. andere Baudrate) ausgeführt, so haben die zweiten Angaben keine Auswirkungen, da die Schnittstelle ja bereits geöffnet und eingestellt ist.

```
int ComNr=2;          //Schnittstelle COM2
long BaudRate=38400; //Baudrate
BYTE PGMPIAdresse=0;  //MPI-Adresse der DLL-Aplikation = 0
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
bool SchnittstelleWarSchonAllokiert=false;
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
//Variablen für Kommu1
int MPIHandle_1=-1;     //Handle der ersten Kommunikationsinstanz
BYTE AGMPIAdresse_1=10; //Die MPI-Adresse der ersten CPU
bool Fehler_1=false;
//Variablen für Kommu2
int MPIHandle_2=-1;     //Handle der zweiten Kommunikationsinstanz
BYTE AGMPIAdresse_2=11; //Die MPI-Adresse der zweiten CPU
bool Fehler_2=false;
//////////////////////////
//Verbindung 1 aufbauen
if (!MPI6_OpenRS232(&MPIHandle_1, ComNr, BaudRate,
                            PGMPIAdresse, HoechsteMPI,
                            &SchnittstelleWarSchonAllokiert,
                            &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "Kommu 1",
                MB_ICONEXCLAMATION);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung  1 war erfolgreich.", "",
           MB_ICONINFORMATION);
```

```
//Verbindung 2 aufbauen
if (!MPI6_OpenRS232(&MPIHandle_2, ComNr, BaudRate,
                            PGMPIAdresse, HoechsteMPI,
                            &SchnittstelleWarSchonAllokiert,
                            &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "Kommu 2",
                MB_ICONEXCLAMATION);
     //Kommu 1 wieder schliessen
     MPI6_CloseCommunication(MPIHandle_1, &Error);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung  2 war erfolgreich.", "",
           MB_ICONINFORMATION);
```

Man beachte, dass bei den Einleitungen verschiedene MPI-Handles übergeben werden. Diese Handles dienen zur Spezifizierung der verschiedenen Kommunikationsinstanzen.
Beim zweiten Aufruf der Funktion "MPI6_OpenRS232" wird der Parameter "SchnittstelleWarSchonAllokiert" mit dem Wert 'true' zurückgeliefert. Denn die Schnittstelle COM2 wurde bereits mit der ersten Einleitung von der ersten Kommunikationsinstanz geöffnet.

7.2 Kommunikationen zu den CPUs aufbauen

Nun können die beiden Kommunikationsinstanzen die Verbindung zu der jeweiligen CPU aufbauen. Dies geschieht über die Funktion "MPI6_ConnectToPLC". Die Funktion muss dabei für jede Kommunikationsinstanz aufgerufen werden.

```
//Kommunikation 1 aufbauen
if (!MPI6_ConnectToPLC(MPIHandle_1, AGMPIAdresse_1, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "Kommu 1",
                MB_ICONEXCLAMATION);
     Fehler_1=true;
}//ende if
else
     MessageBox(AppHandle, "Kommunikation 1 erfolgreich aufgebaut!",
                "", MB_ICONINFORMATION);
//Kommunikation 2 aufbauen
if (!MPI6_ConnectToPLC(MPIHandle_2, AGMPIAdresse_2, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "Kommu 2",
                MB_ICONEXCLAMATION);
     Fehler_2=true;
}//ende if
else
     MessageBox(AppHandle, "Kommunikation 2 erfolgreich aufgebaut!",
                "", MB_ICONINFORMATION);
```

Die angesprochenen AG-MPI-Adresse und die verwendeten MPI-Handles sind bei den Aufrufen der beiden Funktionen "MPI6_ConnectToPLC" unterschiedlich. Denn jede Kommunikationsinstanz soll ja eine andere CPU ansprechen.

7.3 Daten aus der CPU lesen

Jetzt, da beide Kommunikationsinstanzen die Verbindung zur jeweiligen CPU aufgebaut haben, können diese auch Daten aus den CPUs lesen. Die Kommunikationsinstanz mit dem Handle "MPIHandle_1" greift dabei immer auf die CPU mit der MPI-Adresse "AGMPIAdresse_1" zu, während die Instanz mit dem Handle "MPIHandle_2" immer die MPI-Adresse "AGMPIAdresse_2" anspricht. Nachfolgend ist das Listing zu sehen.

```
if (!Fehler_1){
     WORD LaengeInByte=0;
     //
     if (!MPI6_GetLengthDB(MPIHandle_1, 1, &LaengeInByte, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "Kommu1\nLänge des DB1: %u Byte",
                  LaengeInByte);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
if (!Fehler_2){
     WORD LaengeInByte=0;
     //
     if (!MPI6_GetLengthDB(MPIHandle_2, 1, &LaengeInByte, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "Kommu2\nLänge des DB1: %u Byte",
                  LaengeInByte);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
```

Es werden die Längen der Datenbausteine 1 in der jeweiligen CPU ermittelt und ausgegeben. Es können nun weitere Daten aus den CPUs gelesen oder geschrieben werden. Die Reihenfolge der Lese- bzw. Schreibvorgänge ist dabei unerheblich.

7.4 Abbau der Kommunikation und beseitigen der Kommunikationsinstanzen

Soll die Kommunikation zu den CPUs beendet, die Schnittstelle geschlossen und die Kommunikationsinstanzen beseitigt werden, so muss man die Funktion "MPI6_CloseCommunication" für jede Kommunikationsinstanz aufrufen.
Wird die Funktion nur für eine Kommunikationsinstanz aufgerufen, so bleibt die Kommunikation der anderen Instanz davon unberührt. Erst wenn die zweite Kommunikationsinstanz ebenfalls über die Funktion "MPI6_CloseCommunication" beseitigt wird, wird auch die Schnittstelle freigegeben.

```
//Kommunikation 1 beenden
if (!MPI6_CloseCommunication(MPIHandle_1, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation 1 ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
//Kommunikation 2 beenden
if (!MPI6_CloseCommunication(MPIHandle_2, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation 2 ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
```

7.5 Anmerkungen zum Beispiel

Würden im Beispiel unterschiedliche serielle Schnittstellen verwendet werden, z.B. COM1 für die Kommunikationsinstanz 1 und COM2 für die Kommunikationsinstanz 2, so könnten die Funktionen der Instanzen auch in unterschiedlichen Threads aufgerufen werden. Dies hat den Vorteil, dass sich die Instanzen nicht gegenseitig blockieren.

Die bei einer seriellen Verbindung zu verwendenden MPI-Adapter können meist mehrere Verbindungen verwalten (MHJ-MPI-Adapter 2 Verbindungen, SIEMENS-Adapter 4 Verbindungen). Dies bedeutet, dass z.B. bei einem MHJ-MPI-Adapter max. 2 Kommunikationsinstanzen über einen MPI-Adapter (und somit einer seriellen Schnittstelle) auf Kommunikationspartner zugreifen können.

Wird die Verbindung über TCP/IP und einer CPU mit integrierter Ethernet-Schnittstelle bzw. Ethernet-CP hergestellt, so sind die Kommunikationsinstanzen ebenfalls unabhängig. Die Vorgehensweise ist die Gleiche wie bei der seriellen Schnittstelle, nur das die Funktion "MPI6_OpenTcpIp" anstatt "MPI6_OpenRS232" verwendet wird.

8 Was ist bei der Verwendung eines MHJ-NetLink zu beachten?

MHJ-NetLinks können mind. zwei Kommunikationsverbindung verwalten. Es besteht auch die Möglichkeit mehrere MHJ-NetLinks mit unterschiedlichen IP-Adressen anzusprechen.
Im nachfolgenden Beispiel sollen zwei über MPI vernetzte CPUs über zwei MHJ-NetLinks angesprochen werden. Die CPUs haben die MPI-Adresse 10 und 12.

8.1 Konfiguration der MHJ-NetLinks

Bevor ein MHJ-NetLink über die MPI-DLL angesprochen werden kann, muss dieser über den mitgelieferten MHJ-NetLink-Konfigurator konfiguriert werden. In diesem können die IP-Adresse, die Netzeinstellungen usw. getätigt und im MHJ-NetLink abgelegt werden. Wurde ein MHJ-NetLink eingestellt, so sind die Daten fest im MHJ-NetLink vorhanden, auch nach einem Spannungsausfall. Die Konfiguration muss somit nur ein Mal vorgenommen werden, außer man möchte die Daten ändern.
Nachfolgend ist der MHJ-NetLink-Konfigurator zu sehen. Nach dem Start des Programms wurde der Button "MHJ-NetLinks ermitteln" betätigt.

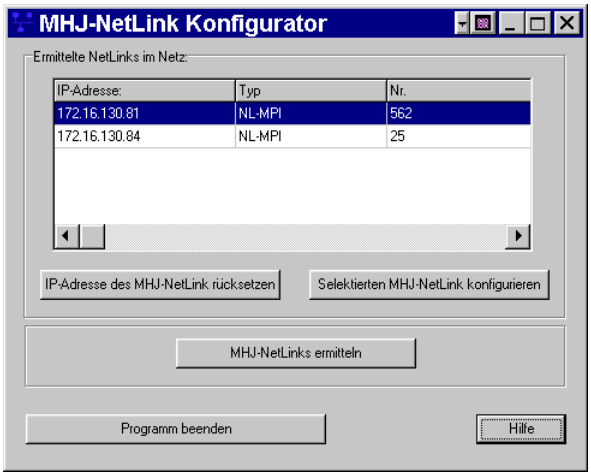


Bild: Konfigurator für MHJ-NetLink

Man erkennt, dass zwei MHJ-NetLinks im Netz gefunden wurden. Des Weiteren kann man die IP-Adresse der MHJ-NetLinks ablesen.
Es soll zunächst der MHJ-NetLink mit der IP-Adresse 172.16.130.84 konfiguriert werden. Dazu selektiert man diesen in der Liste und betätigt den Button "Selektierten MHJ-NetLink konfigurieren". Hätte der MHJ-NetLink die IP-Adresse 0.0.0.0, so wie dies im Auslieferungszustand der Fall ist, so würde zunächst ein Dialog erscheinen, auf welchem die IP-Adresse anzugeben ist. Danach ist der nachfolgend dargestellte Dialog zu sehen:

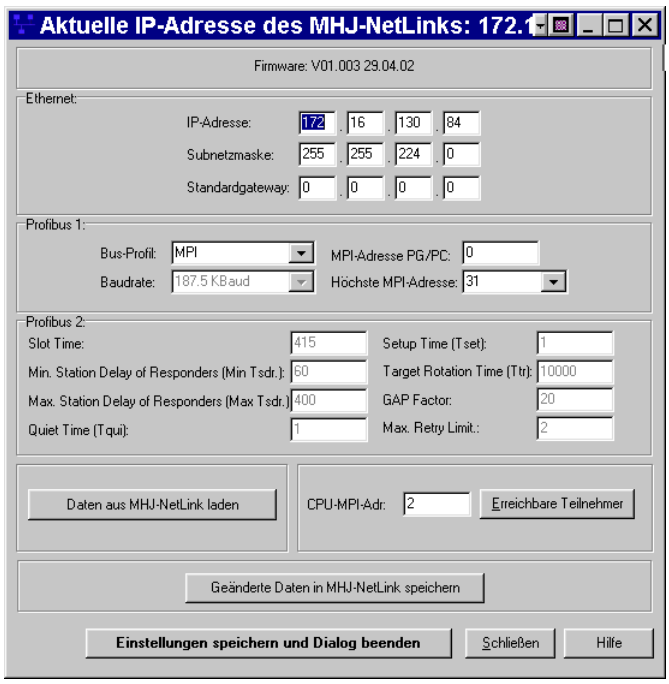


Bild: Der Konfigurationsdialog

Auf diesem können die Netzeinstellungen, die Einstellung der PG-MPI-Adresse usw. vorgenommen werden.
Im Beispiel wird als Bus-Profil "MPI" selektiert, was auch der Standardeinstellung entspricht. Weiterhin wird die "MPI-Adresse PG/PC" auf den Wert '1' eingestellt. Die MPI-Adresse der CPU muss nicht eingestellt werden.
Nun betätigt man den Button "Geänderte Daten in MHJ-NetLink speichern". Daraufhin ist eine Meldung zu sehen, welche besagt, dass die Daten in den MHJ-NetLink übertragen werden. Wird die Meldung mit "Ja" bestätigt, so wird der Vorgang gestartet.
Nachdem die Daten im MHJ-NetLink gespeichert wurden, wird dies wiederum gemeldet und man muss den MHJ-NetLink spannungslos schalten. Dies wird durch Abziehen und Aufstecken des NetLink von der PG-Schnittstelle der CPU erreicht.
Jetzt kann der Dialog über den Button "Einstellungen speichern und Dialog beenden" verlassen werden.

Der gleiche Vorgang wird nun mit dem MHJ-NetLink mit der IP-Adresse 172.16.130.81 durchgeführt. Dazu selektiert man diesen in der Liste und betätigt anschließend den Button "Selektierten MHJ-NetLink konfigurieren". Daraufhin ist der oben gezeigte Dialog zu sehen, allerdings mit der anderen IP-Adresse. Auf diesem Dialog wird ebenfalls als Bus-Profil "MPI" selektiert. Die PG-MPI-Adresse wird auf den Wert '3' eingestellt.
Anschließend betätigt man den Button "Geänderte Daten in MHJ-NetLink speichern" und führt die Schritte durch wie oben bereits beschrieben. Danach verläßt man den Dialog über den Button "Einstellungen speichern und Dialog beenden".
Hat man diese Schritte ausgeführt, so kann der Konfigurator über den Button "Programm beenden" geschlossen werden.

Weiterführende Informationen zur Inbetriebnahme eines MHJ-NetLink können der Hilfe zur Konfigurationssoftware entnommen werden.

8.2 Einleitungen ausführen

Wie bei einer Verbindung über RS232, muss auch bei einer Verbindung über einen MHJ-NetLink, die Open-Funktion aufgerufen. Allerdings ist hierbei die Funktion "MPI6_OpenNetLink" zu verwenden.
Nachfolgend ist dies zu sehen:

```
BYTE HoechsteMPI=31;  //Höchste erlaubte MPI-Adresse im Netz = 31
WORD Error=0;         //Error-Variable
char ErrorString[255]={0};//Error-String zum Anzeigen des Fehlers
//Variablen für Kommu1
char IPAdresseStr_1[50]={0};
int MPIHandle_1=-1;     //Handle der ersten Kommunikationsinstanz
BYTE PGMPIAdresse_1=1;  //MPI-Adresse der 1. Kommunikationsinstanz = 1
BYTE AGMPIAdresse_1=10; //Die MPI-Adresse der ersten CPU
bool Fehler_1=false;
//Variablen für Kommu2
char IPAdresseStr_2[50]={0};
BYTE PGMPIAdresse_2=3;  //MPI-Adresse der 2. Kommunikationsinstanz = 3
int MPIHandle_2=-1;     //Handle der zweiten Kommunikationsinstanz
BYTE AGMPIAdresse_2=12; //Die MPI-Adresse der zweiten CPU
bool Fehler_2=false;
//
strcpy(IPAdresseStr_1, "172.16.130.84");
strcpy(IPAdresseStr_2, "172.16.130.81");
//////////////////////////
//Verbindung 1 aufbauen
if (!MPI6_OpenNetLink(&MPIHandle_1, IPAdresseStr_1,
                              PGMPIAdresse_1,
                              HoechsteMPI, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung  1 war erfolgreich.", "",
           MB_ICONINFORMATION);
//Verbindung 2 aufbauen
if (!MPI6_OpenNetLink(&MPIHandle_2, IPAdresseStr_2,
                              PGMPIAdresse_2,
                              HoechsteMPI, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     //
     MPI_A_KommuBeenden(MPIHandle_1, &Error);
     return;
}//ende if
MessageBox(AppHandle, "Einleitung  2 war erfolgreich.", "",
           MB_ICONINFORMATION);
```

8.3 Kommunikation aufbauen

Die weiteren Funktionen unterscheiden sich nicht gegenüber anderen Verbindungen (z.B. NetLink PRO, TCP/IP oder RS232). Dies bedeutet, es wird für jede Kommunikationsinstanz die Funktion "MPI6_ConnectToPLC" aufgerufen, wobei die jeweilige AG-MPI-Adresse anzugeben ist (diese würde bei TCP/IP immer mit 2 vorgegeben werden).

```
//Kommunikation 1 aufbauen
if (!MPI6_ConnectToPLC(MPIHandle_1, AGMPIAdresse_1, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     Fehler_1=true;
}//ende if
else {
     MessageBox(AppHandle, "Kommunikation 1 erfolgreich aufgebaut!",
                "", MB_ICONINFORMATION);
}//ende else
//Kommunikation 2 aufbauen
if (!MPI6_ConnectToPLC(MPIHandle_2, AGMPIAdresse_2, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     Fehler_2=true;
}//ende if
else {
     MessageBox(AppHandle, "Kommunikation 2 erfolgreich aufgebaut!",
                "", MB_ICONINFORMATION);
}//ende else
```

8.4 Daten lesen

Nun sollen die Daten aus den CPUs gelesen werden. Die hierbei verwendeten Zugriffsfunktionen sind ebenfalls von der verwendeten Verbindungsressource unabhängig. Wie beim letzten Beispiel soll die Länge des DB1 in der jeweiligen CPU ermittelt werden.

```
//Daten lesen
if (!Fehler_1){
     WORD LaengeInByte=0;
     //
     if (!MPI6_GetLengthDB(MPIHandle_1, 1, &LaengeInByte, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "Kommu1\nLänge des DB1: %u Byte",
                  LaengeInByte);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if

if (!Fehler_2){
```

```
     WORD LaengeInByte=0;
     //
     if (!MPI6_GetLengthDB(MPIHandle_2, 1, &LaengeInByte, &Error)){
         //Fehler anzeigen
         MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
         MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
     }//ende if
     else {
         char AusgabeStr[255]={0};
         wsprintf(AusgabeStr, "Kommu2\nLänge des DB1: %u Byte",
                  LaengeInByte);
         MessageBox(AppHandle, AusgabeStr, "", MB_ICONINFORMATION);
    }//ende else
}//ende if
```

Es können nun weitere Daten aus den CPUs gelesen oder geschrieben werden. Die Reihenfolge der Lese- bzw. Schreibvorgänge ist dabei unerheblich.

8.5 Kommunikation abbauen

Will man die Kommunikation mit der jeweiligen CPU beendet, so erreicht man dies über die Funktion "MPI6_CloseCommunication". Gleichzeitig wird die Kommunikationsressource freigegeben und die Kommunikationsinstanz beseitigt.

```
//Kommunikation 1 beenden
if (!MPI6_CloseCommunication(MPIHandle_1, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_1, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation 1 ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
//Kommunikation 2 beenden
if (!MPI6_CloseCommunication(MPIHandle_2, &Error)){
     //Fehler anzeigen
     MPI_A_GetDLLError(MPIHandle_2, ErrorString, Error);
     MessageBox(AppHandle, ErrorString, "", MB_ICONEXCLAMATION);
}//ende if
MessageBox(AppHandle, "Kommunikation 2 ohne Fehler beendet.", "",
           MB_ICONINFORMATION);
```

8.6 Anmerkungen zum Beispiel

Da im Beispiel unterschiedliche Kommunikationsressourcen verwendet werden (zwei MHJ-NetLinks) können die Funktionen der jeweiligen Kommunikationsinstanz auch in unterschiedlichen Threads aufgerufen werden. Dies hat den Vorteil, dass sich die Kommunikationsinstanzen nicht gegenseitig blockieren.

9 Was ist bei der Verwendung eines NETLink PRO zu beachten?

Ein NETLink PRO ermöglicht die Kommunikation von TCP/IP auf der PC-Seite auf eine MPI oder DP-Schnittstelle der CPU. Dabei werden im MPI/DP-Netz alle Baudraten bis 12MBaud unterstützt. Der NETLink PRO kann dabei mind. 4 PC-Verbindungen verwalten.

9.1 Konfiguration eines NETLink PRO

Bevor ein NETLink PRO über ComDrvS7 angesprochen werden kann, muss dieser über den mitgelieferten NETLink PRO-Konfigurator konfiguriert werden. In diesem können die IP-Adresse, die Netzeinstellungen usw. getätigt und im NETLink PRO abgelegt werden. Wurde ein NETLink PRO eingestellt, so sind die Daten fest im NETLink PRO vorhanden, auch nach einem Spannungsausfall. Die Konfiguration muss somit nur ein Mal vorgenommen werden, außer man möchte die Daten ändern.
Nachfolgend ist der NETLink PRO-Konfigurator zu sehen. Nach dem Start des Programms wurde der Button "NETLink PRO suchen" betätigt.

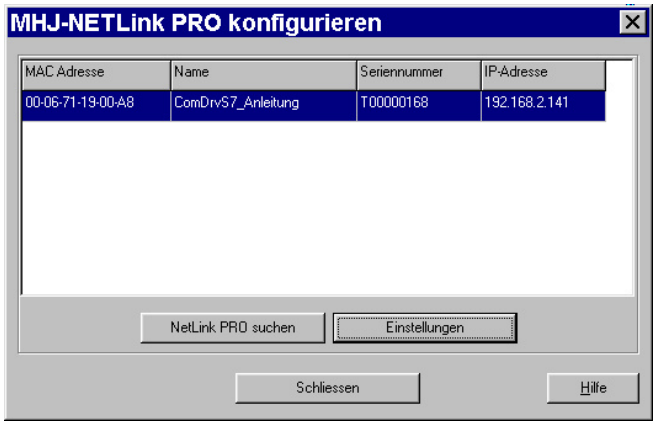


Bild: Liste der vorhandenen NETLink PRO im Netz

Nun kann der gewünschte NETLink Pro in der Liste selektiert und anschliessend der Button "Einstellungen" betätigt werden. Als Folge ist der Dialog "NETLink PRO Einstellungen" zu sehen, auf dem die für die Kommunikation bedeutenden Parameter einstellbar sind.

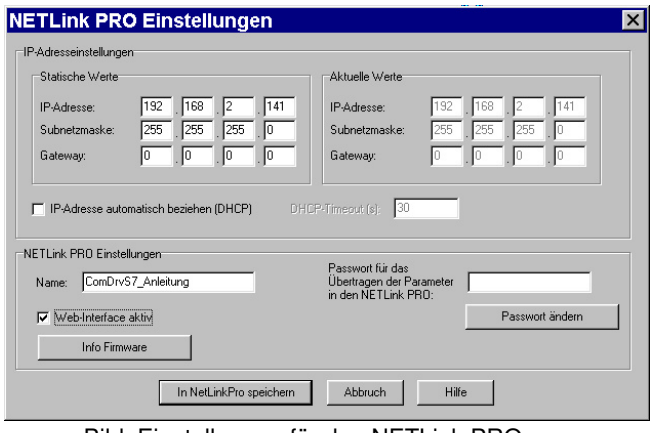


Bild: Einstellungen für den NETLink PRO

eine Hilfe zu den einzelnen Parametern kann über den Button "Hilfe" aufgerufen werden. Wurden die nötigen Einstellungen vorgenommen, so können diese über den Button "In NETLink PRO speichern" in den NETLink PRO geschrieben werden.

9.2 Die beiden Einleitungsfunktionen des NETLink PRO

Soll eine Kommunikationsinstanz für einen NETLink PRO aufgebaut werden, so stehen zwei Einleitungsfunktionen zur Verfügung. Es sind dies die Funktionen
MPI6_Open_NetLinkPro_TCP_AutoBaud
und
MPI6_Open_NetLinkPro_TCP_SelectBaud
Der Unterschied der beiden Funktionen besteht darin, dass bei der Funktion
MPI6_Open_NetLinkPro_TCP_AutoBaud
die Baudrate im MPI/DP-Netz nicht bekannt sein muss. Bei der Einleitung versucht der NETLink PRO die im Bus eingestellte Baudrate zu ermitteln und zu übernehmen. Der Nachteil dieser Variante liegt in der Zeit, welche für dieses Ermitteln benötigt wird, denn diese kann einige Sekunden betragen. Um diese Zeit wird der gesamte Einleitungsvorgang verlängert.

Ist die Baudrate bekannt, dann sollte möglichst die Funktion
MPI6_Open_NetLinkPro_TCP_SelectBaud
verwendet werden, denn hier wird die Baudrate vorgegeben, weshalb die Zeit für das Ermitteln der Baudrate gespart wird.

10 Was ist bei der Verwendung von SIMATIC®-NET zu beachten?

Ab der Version 4 von ComDrvS7 wird SIMATIC®-NET unterstützt.

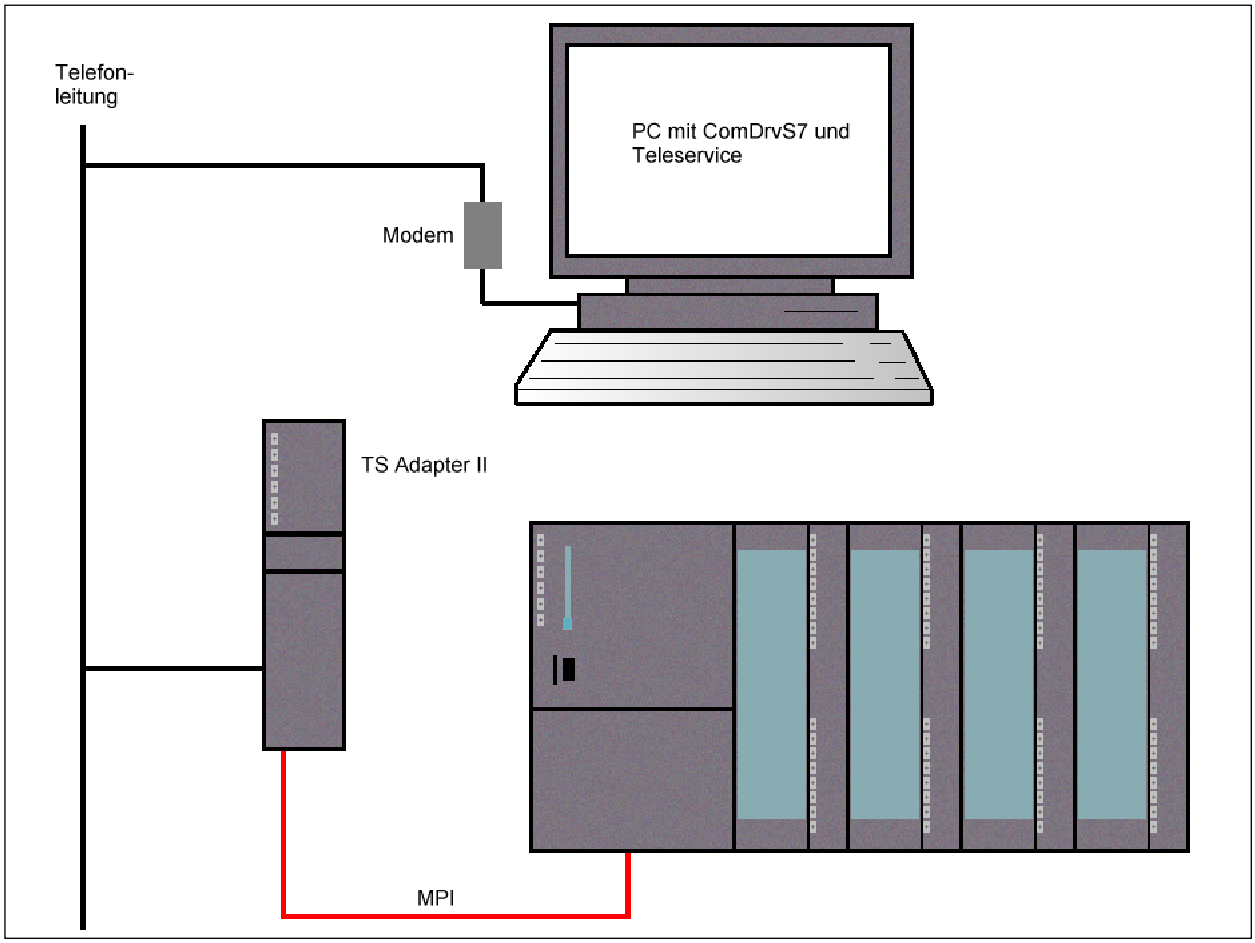
Voraussetzung ist, dass der SIMATIC® NET Treiber auf dem PC installiert ist. Dies ist z.B. der Fall, wenn auf dem PC die Software Simatic®-Manager (ab V5.1), der Treiber des SIEMENS-USB-Adapters oder der Teleservice ab V6 installiert sind. Das dabei verwendete Interface ist auf dem Dialog "PG/PC-Schnittstelle einstellen" zu selektieren. Dieser Dialog kann über die Datei "s7epatsx.exe" im System32-Verzeichnis von Windows aufgerufen werden. So können beispielsweise der USB-MPI-Adapter von Siemens, der CP5511 oder CP5612 selektiert werden.
Dieses selektierte Interface und die entsprechenden Einstellungen, werden von ComDrvS7 verwendet, wenn die Einleitung über die Funktion **MPI6_Open_SimaticNet** ausgeführt wird.

Neben ComDrvS7 können auch weiterhin die Siemens-Softwareprodukte auf die CPU zugreifen, sofern die Kommunikationsressourcen der CPU nicht erschöpft sind. Es ist also beispielsweise möglich, dass ihre Applikation mit ComDrvS7 auf eine CPU zugreift (über SIMATIC®-NET) und gleichzeitig ein Zugriff über den Simatic®-Manager erfolgt.

11 Vorgehensweise bei Fernabfrage über Telefonleitung mit Hilfe von ComDrvS7

Ab der Version 4 von ComDrvS7 kann mit Hilfe des Teleservice (ab Version 6) von Siemens eine Fernabfrage über die Telefonleitung durchgeführt werden.

Auf der PC-Seite kann dabei ein beliebiges Modem zum Einsatz kommen. Auf der Anlagenseite werden unter anderem die Teleservice II-Adapter von Siemens unterstützt.
Nachfolgend ist eine Darstellung zu sehen, bei denen die notwendigen Komponenten benannt sind:



Auf dem PC muss der Teleservice ab V6 von Siemens installiert sein. Der PC ist über ein Modem mit der Telefonleitung verbunden. Auf der Anlagenseite kommt beispielsweise der Teleservice II-Adapter von Siemens zum Einsatz. Innerhalb von ComDrvS7 ist der Kommunikationsweg SIMATIC®-NET zu verwenden, d.h. es wird die Einleitungsfunktion **MPI6_Open_SimaticNet** aufgerufen. Als Interface im SIMATIC®-NET-Treiber wird beim im Beispiel eingesetzten TS-Adapter II die Einstellung "TS Adapter" selektiert.
Hierbei ist auch das Routing möglich.

12 Allgemeine Hinweise zur ComDrvS7-DLL

12.1 Was muss man beachten, wenn man mit mehreren Kommunikationsinstanzen auf eine CPU zugreift?

Es kommt zu Kommunikationsfehlern, wenn mehrere Kommunikationsinstanzen Auskunftsfunktionen einer CPU in Anspruch nehmen. Wird z.B. von mehreren Instanzen unmittelbar hintereinander oder sogar in unterschiedlichen Threads quasi "parallel" die Betriebsartenstellung abgefragt, so wird es zu einem Kommunikationsfehler kommen. Denn die CPU muss bei einer solchen Anfrage die Antwort virtuell zusammenstellen und dazu ist diese nicht andauernd in der Lage.
Statusabfragen können parallel erfolgen allerdings auch nur in einer begrenzten Anzahl. Wie hoch diese Anzahl ist, hängt vom verwendeten CPU-Typ ab. Diese Anzahl kann dem CPU-Handbuch entnommen werden.

12.2 Was muss man beachten, wenn neben der ComDrvS7-DLL noch weitere Applikationen auf dem PC ablaufen?

Kommen neben dem ComDrvS7 noch weitere Applikationen zur Ausführung, so müssen die Funktionen des ComDrvS7 in einem oder mehreren Threads aufgerufen werden. Wichtig dabei ist, dass der oder die Threads die Priorität "THREAD_PRIORITY_TIME_CRITICAL" erhalten, damit die Antwortzeiten auf dem MPI/DP-Bus bzw. über TCP/IP eingehalten werden können.

12.3 Wann können die Funktionen der einzelnen Kommunikationsinstanzen in verschiedenen Threads aufgerufen werden?

Werden mehrere Kommunikationsinstanzen verwendet, d.h wird eine Verbindung zu mehreren CPUs aufgebaut und werden dabei unterschiedliche Kommunikationsressourcen genutzt, so ist die Voraussetzung gegeben, dass die Funktionen der einzelnen Instanzen in unterschiedlichen Threads aufgerufen werden. Jeder der Threads muss dabei die Priorität "THREAD_PRIORITY_TIME_CRITICAL" besitzen.
Es ist allerdings nicht möglich, Funktionen einer Kommunikationsinstanz in unterschiedlichen Threads aufzurufen.

Wird eine Kommunikationsressource von mehreren Instanzen verwendet, z.B. wenn über die Schnittstelle COM1 auf 2 CPUs zugegriffen wird, dann müssen die Funktionen der beiden Instanzen in einem Thread hintereinander ausgeführt werden.

13 Fehlermeldungen

Nachfolgend sind die möglichen Fehlernummern aufgelistet. Einer dieser Werte kann in der Variablen "Error" einer DLL-Funktion enthalten sein, wenn diese Funktion '0' (FALSE) als Funktionswert liefert.
Über die Funktionen MPI_A_GetDLLError, MPI_A_GetDLLErrorEng, MPI6_GetDLLError oder MPI6_GetDLLErrorEng kann ein beschreibender String (nullterminierter) für den jeweiligen Fehler angefordert werden.

Wert (dez)	Beschreibung
54.273	Die angeforderten Informationen sind auf dem AG nicht verfügbar!
53.825	Schutzstufen-Fehler der CPU!
53.409	Aktion wegen eingestellter Schutzstufe nicht möglich!
53.377	Stellen Sie die für diese Funktion nötige Betriebsart ein!
53.298	Die dem AG übergebenen Parameter sind fehlerhaft!
33.794	Aktion kann wegen eines falschen Zustands des AGs nicht durchgeführt werden!
33.540	Meldung von der Baugruppe. Es liegt ein Ressourcen-Engpaß vor!
19.718	Vorübergehender Ressourcenmangel im AG. Wiederholen Sie die Anfrage.
16.997	Die MPI-Adresse des PG ist im Netz schon vergeben!
16.949	Ein angeschlossenes AG hat eine zu hohe Teilnehmeradresse!
16.662	Der Partner verweigert die Kommunikation!
1.046	CPU ist keine S7-1500
1.045	CPU ist keine LOGO
1.044	CPU ist keine S7-1200
1.043	Fehler beim internen Setzen der S7-1500
1.030	V-Bereich kann nur in der Micro-Version angesprochen werden.
1.029	Funktion ist mit S7-1200 nicht möglich.
1.028	Funktion ist mit LOGO nicht möglich.
1.027	Funktion ist nur in Micro-Version möglich.
1.026	Funktion ist in Micro-Version nicht möglich.
1.004	Funktion nur in Extended-Version vorhanden.
1.003	Bausteintyp nicht bekannt!
1.002	DB0 ist nicht erlaubt!
1.001	Diese Funktion ist in der Lite-Version nicht ausführbar!
1.000	Fehler beim Anlegen einer DLL-Instanz!
733	Das übergebene Handle ist nicht gültig-
732	Aktion mit mind. einem Baustein nicht möglich.
731	Fehler beim Ausführen von Komprimieren.
729	Fehler, die WLD-Datei ist bereits vorhanden.
728	Fehler beim Einstellen des Modus für die Familie S7-1200®.

727	Falsche Stellung des Betriebsartenschalters oder die CPU befindet sich schon in der Betriebsart.
726	Die Aktion ist bei dieser Betriebsart nicht möglich.
725	Aktion ist in der Betriebsart RUN nicht möglich.
724	Der Dienst wird von der CPU nicht unterstützt.
723	Anzahl der Bytes bei Mix-Funktion ist ungültig.
722	Fehler beim Übernehmen der Statusdaten bei Mix-Funktion.
721	Interner Fehler Mix-Funktion: Pointeranzahl zu hoch.
720	Ein oder mehrere Steuerwerte sind fehlerhaft.
719	Die Anzahl der DBs in der WLD-Datei übersteigt den angegebenen Max-Parameter.
718	WLD-Aktion: Baustein ist in der CPU nicht vorhanden.
717	Der DB ist bereits in der WLD vorhanden. Überschreiben nicht möglich.
716	Die Datei ist keine korrekte WLD-Datei.
715	Fehler beim Übertragen des Bausteins aus der WLD-Datei.
714	WLD-Aktion: Fehler beim Test, ob der zu übertragende Baustein bereits in der CPU vorhanden ist.
713	WLD-Aktion: Baustein ist bereits in der CPU vorhanden.
712	WLD-Datei ist nicht vorhanden.
711	Baustein ist in der WLD-Datei nicht vorhanden.
710	WLD-Aktion: Fehler bei Dateioperation.
709	WLD-Aktion: Baustein ist für die Aktion zu gross.
708	Status einer oder mehrere Operanden kann nicht geliefert werden. Mögliche Ursache: Einer oder mehrere Operanden sind in der CPU nicht vorhanden.
707	Angegebener Operand ist bei dieser Funktion nicht erlaubt.
706	Das Steuern von einem oder mehreren Operanden ist nicht möglich. Mögliche Ursache: Einer oder mehrere Operanden sind in der CPU nicht vorhanden.
614	NetLink PRO: Baudrate im Bus kann nicht ermittelt werden!
604	Statusdaten eines oder mehrerer angeforderter Operanden nicht vorhanden.
603	Fehler beim Auswerten der Statusdaten.
602	Fehler Passwort enthält zu viele Zeichen.
601	Fehler beim Wandeln des Passwortes.
518	Es ist ein unbekannter Fehler bei der Einleitung aufgetreten!
517	Fehler beim Auswerten der erreichbaren Teilnehmer!
513	Fehler beim Schliessen der Schnittstelle!
512	Fehler beim Öffnen der Schnittstelle!
511	Fehler beim Auswerten der Info-Daten!
510	Übergebene Parameter sind fehlerhaft!
509	Kommunikationsfehler aufgetreten!
508	Speicherfehler. Speicher konnte nicht allokiert werden!

500	Demobeschränkung erreicht. Diese Meldung erscheint nur in der Demo-Version der DLL auf, sobald der Bereich der Demo-Version überschritten wurde. Bitte beachten Sie in diesem Fall, die der Demo-Version beigelegten Zusatzinfos über die erlaubten Bereiche der Operanden. Ab Version 6 hinfällig.

14 Ethernet-Verbindung in der LOGO!®-Programmiersoftware parametrieren

14.1 Einstellung der IP-Adressdaten des LOGO!®

Um das LOGO über Ethernet ansprechen zu können, muss dieses mit einer IP-Adresse versehen werden. Das Gerät muss dabei für den PC mit der LOGO-Programmiersoftware bzw. des ComDrvS7 erreichbar sein. Die LOGO-Steuerung kann direkt an die Netzwerkkarte des PCs oder aber über einen Switch im Netzwerk angeschlossen sein.

Im Beispiel wird davon ausgegangen, dass der PC mit der Programmiersoftware und ComDrvS7 die IP-Adresse 192.168.1.90 besitzt. Die LOGO-Steuerung soll sich im gleichen Subnetz befinden und unterscheidet sich der Einfachheit halber nur in der letzten Stelle der IP-Adresse. Die LOGO-Steuerung soll aus diesem Grund die IP-Adresse 192.168.1.196 erhalten. Dabei ist zu bedenken, dass diese IP-Adresse noch nicht belegt sein darf.

Die IP-Adresse und Subnetzmaske der LOGO-Steuerung kann sehr schnell über das Display des LOGO eingestellt werden. Dazu betätigt man die Taste ESC auf dem LOGO, um in das Menü zu gelangen. Danach wird über die Pfeiltasten Auf und Ab der Menüpunkt "Network" selektiert und über die OK-Taste bestätigt. Im Untermenü wird der Menüpunkt "IP address" über OK ausgewählt. Daraufhin ist die momentan eingestellte IP-Adresse zu sehen.

Ein weiteres mal wird die Taste OK betätigt, um über die Pfeiltasten die einzelnen Adressen einstellen zu können. Im Beispiel wird, wie schon erwähnt, die Adresse 192.168.1.196 eingestellt. Wurde diese Adresse eingestellt, so kann über OK die Eingabe übernommen werden.
Jetzt zur Subnetzmaske. Diese hat im Beispiel die Einstellung 255.255.255.0. Von der momentanen Stelle aus kann die Subnetzmaske über das Betätigen der Pfeiltaste nach unten ausgewählt und über OK das Verändern gestartet werden. Hat man 255.255.255.0 eingestellt wird die Eingabe über OK abgeschlossen.Für weitere Informationen zur Einstellung der IP-Daten im LOGO-Gerät lesen Sie bitte die entsprechenden Abschnitte in der Hilfe der LOGO-Programmiersoftware oder dem Handbuch der LOGO-Steuerung.

Damit sind die IP-Daten in der LOGO-Steuerung eingestellt und man kann über die LOGO-Taste "ESC" zum Hauptmenü des LOGO zurück kehren.

14.2 Besonderheiten bei einer LOGO!® ab 0BA8

Ab den 0BA8 Geräten ist eine Besonderheit bei der Kommunikation zu beachten. Wird eine Kommunikation zu einer 0BA8 geöffnet und findet binnen 5 Sekunden keine Kommunikation (lesen/schreiben) zur LOGO statt, dann wird die Verbindung von der LOGO geschlossen.
Dies bedeutet, findet nur alle 5 Sekunden eine Abfrage statt, dann muss in ComDrvS7 immer eine neue Verbindung geöffnet und nach dem Lesen/Schreiben wieder geschlossen werden. Anderenfalls kommt es zu einem Kommunikationsfehler.

Werden in kürzeren Abständen Daten aus der LOGO gelesen oder in diese geschrieben, dann kann die Verbindung offen gehalten werden.

14.3 Die Ethernet-Verbindung in der LOGO!®-Programmiersoftware parametrieren

Im nächsten Schritt wird in der LOGO-Programmiersoftware der Menüpunkt "Extras->Ethernetverbindungen" ausgeführt. Als Folge erscheint der Dialog "Adresse und Verbindungen konfigurieren". Im Beispiel wird dieser wie folgt ausgefüllt:

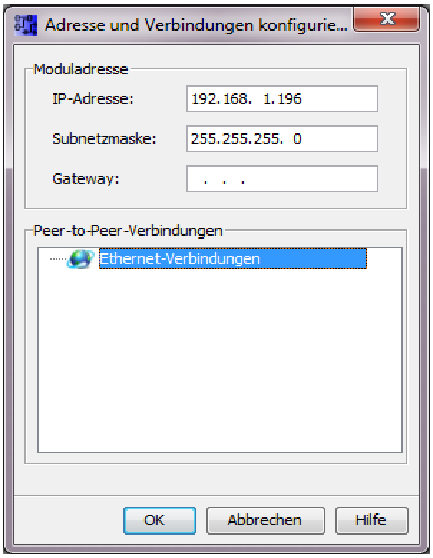


Bild: Dialog "Adresse und Verbindungen konfigurieren"

Im Feld "Moduladresse->IP-Adresse" wird die IP-Adresse des anzusprechenden LOGO eingetragen, im Beispiel somit die 192.168.1.196. Wurde diese IP-Adresse angegeben, so erfolgt die Vorgabe der Subnetzmaske automatisch. Darunter ist das Feld "Peer-to-Peer-Verbindungen" zu sehen. In diesem muss eine neue Ethernetverbindung erzeugt werden. Dazu wählt man den Punkt "Ethernet-Verbindungen" aus und betätigt die rechte Maustaste.

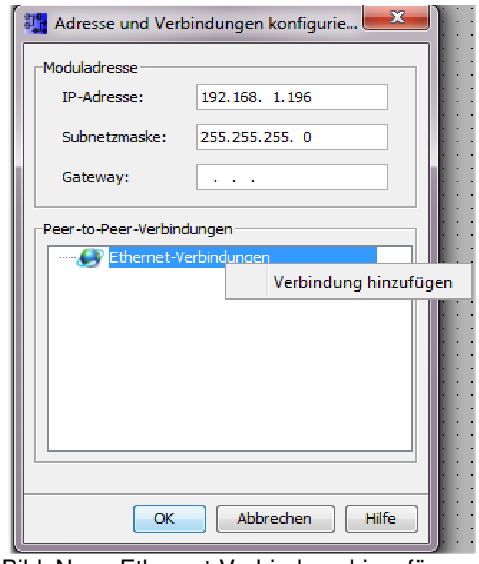


Bild: Neue Ethernet-Verbindung hinzufügen

Jetzt wird der Menüpunkt "Verbindung hinzufügen" ausgewählt. Daraufhin stellt sich der Dialog wie folgt dar:

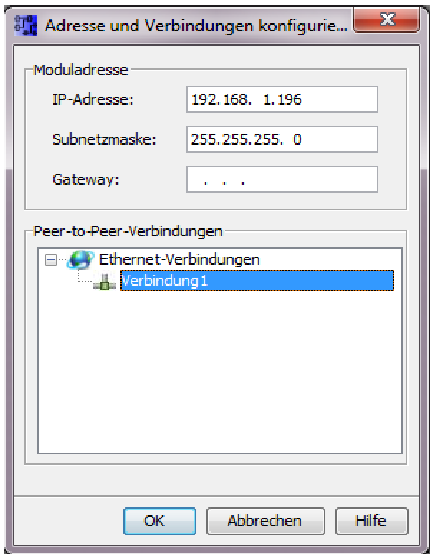


Bild: Neue Verbindung

Nun wird ein Doppelklick auf dem neuen Eintrag "Verbindung1" ausgeführt. Als Folge erscheint der Dialog "Verbindung1", dessen Optionen wie folgt auszufüllen sind:

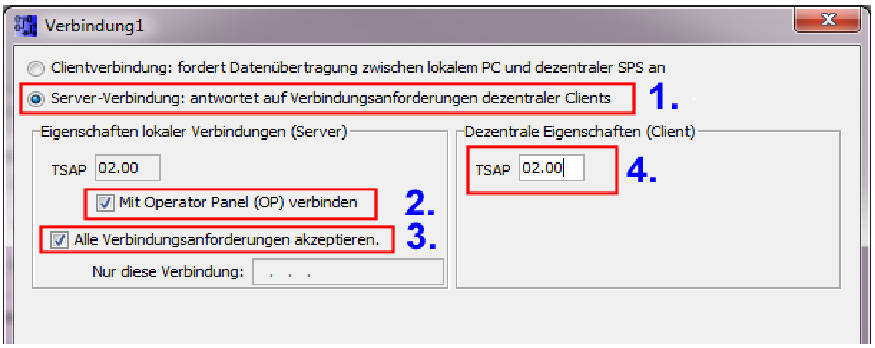


Bild: Einstellungen für die Verbindung

Die Einstellungen sind in der Reihenfolge vorzunehmen, wie auf dem Bild dargestellt.
Anschließend kann der Dialog über OK verlassen werden.
Auch der nun wieder aktive Dialog "Adresse und Verbindungen konfigurieren" wird über OK bestätigt und verlassen.
Damit sind die IP-Einstellungen komplett.

Die Konfiguration muss nun in die LOGO!®-Steuerung übertragen werden. Danach kann man mit ComDrvS7 auf die LOGO!® zugreifen.

15 Der Zugriff auf Operanden in einem LOGO!® von Siemens

Mit jeder neuen Geräteklasse der LOGO!® ändert sich auch der Speicherbereich für die Operanden im V-Bereich der LOGO!®. Nachfolgend sind die Bereiche für die beiden Geräteklassen 0BA7 und 0BA8 aufgeführt.
Die Speicherinformationen können auch in der Hilfe der LOGO!®Soft-Comfort abgerufen werden. Diese befinden sich auf der Hilfeseite mit der Bezeichnung "Extras -> Parameter-VM-Zuordnung"

15.1 Digitale Eingänge

Die digitalen Eingänge eines LOGO können direkt gelesen werden. Dazu ist bei den Lesefunktionen von ComDrvS7 (z.B. MPI6_ReadByte) der Operandenbereich "E" oder "I" anzugeben.
Das Schreiben von Eingängen ist nicht möglich, da der Status sofort wieder von dem Status der physikalisch vorhandenen Eingänge überschrieben wird.

15.1.1 Adressierung der digitalen Eingänge

Die Zuordnung der Eingänge I1 bis I24 (0BA7) bzw. I64 (0BA8) ist in der folgenden Tabelle zu sehen:

LOGO Eingang	Adressierung in ComDrvS7 bei 0BA7/0BA8
I1	E0.0
I2	E0.1
I3	E0.2
I4	E0.3
I5	E0.4
I6	E0.5
I7	E0.6
I8	E0.7
I9	E1.0
I10	E1.1
I11	E1.2
I12	E1.3
I13	E1.4
I14	E1.5
I15	E1.6
I16	E1.7
...	...

15.2 Analoge Eingänge

Der Lese-Zugriff auf die analogen Eingänge eines LOGO erfolgt über den sog. Variablenspeicher. (V-Bereich)
Bei den Lesefunktionen von ComDrvS7 (z.B. MPI6_ReadWord) ist dabei der Operandenbereich "V" anzugeben.

15.2.1 Adressierung der analogen Eingänge

Die Zuordnung der Eingänge AI1 bis AI8 ist in der folgenden Tabelle zu sehen:

LOGO Eingang	Adressierung in ComDrvS7 bei 0BA7	Adressierung in ComDrvS7 bei 0BA8
AI1	VW926	VW1032
AI2	VW928	VW1034
AI3	VW930	VW1036
AI4	VW932	VW1038
AI5	VW934	VW1040
AI6	VW936	VW1042
AI7	VW938	VW1044
AI8	VW940	VW1046
AI8-AI16	-	VW1048-VW1062

Beispiel 0BA7:

Soll der Status des analogen Eingangs AI6 eingelesen werden, so ist als Adresse 936 und der Operandenbereich "V" anzugeben.

Beispiel 0BA8:

Soll der Status des analogen Eingangs AI6 eingelesen werden, so ist als Adresse 1042 und der Operandenbereich "V" anzugeben.

15.3 Digitale Ausgänge

Die digitalen Ausgänge eines LOGO können direkt gelesen oder beschrieben werden. Dazu ist bei den Lese- und Schreibfunktionen von ComDrvS7 (z.B. MPI6_ReadByte oder MPI6_WriteByte) der Operandenbereich "A" oder "Q" anzugeben.
Das Schreiben von Ausgängen ist möglich, allerdings wird der Status eines gesteuerten Ausgangs unter Umständen vom Schaltprogramm im LOGO überschrieben.

15.3.1 Adressierung der digitalen Ausgänge

Die Zuordnung der Eingänge Q1 bis Q16 (0BA7) bzw. Q64 (0BA8) ist in der folgenden Tabelle zu sehen:

LOGO Ausgang	Adressierung in ComDrvS7
Q1	A0.0
Q2	A0.1
Q3	A0.2
Q4	A0.3
Q5	A0.4
Q6	A0.5
Q7	A0.6
Q8	A0.7
Q9	A1.0.
Q10	A1.1
Q11	A1.2
Q12	A1.3
Q13	A1.4
Q14	A1.5
Q15	A1.6
Q16	A1.7

15.4 Analoge Ausgänge

Der Lese- und Schreibzugriff auf die analogen Ausgänge eines LOGO erfolgt über den sog. Variablenspeicher.
Bei den Funktionen von ComDrvS7 (z.B. MPI6_ReadWord oder MPI6_WriteWord) ist dabei der Operandenbereich "V" anzugeben.

15.4.1 Adressierung der analogen Ausgänge

Die Zuordnung der Ausgänge ist in der folgenden Tabelle zu sehen:

LOGO Ausgang	Adressierung in ComDrvS7 0BA7	Adressierung in ComDrvS7 0BA8
AQ1	VW944	VW1072
AQ2	VW946	VW1074
AQ3-AQ16	-	VW1076-VW1102

15.5 Digitale Merker

Die digitalen Merker eines LOGO können über den V-Bereich gelesen oder beschrieben werden. Dazu ist bei den Lese- und Schreibfunktionen von ComDrvS7 (z.B. MPI6_ReadByte oder MPI6_WriteByte) der Operandenbereich "V" anzugeben.
Das Schreiben von Merkern ist möglich, allerdings wird der Status eines gesteuerten Merkers unter Umständen vom Schaltprogramm im LOGO überschrieben.

15.5.1 Adressierung der digitalen Merker

Die Zuordnung der Merker M1-M27 (0BA7) und M1-M111 (0BA8) ist in der folgenden Tabelle zu sehen:

LOGO Merker	Adressierung in ComDrvS7 0BA7	Adressierung in ComDrvS7 0BA8
M1	V948.0	V1104.0
M2	V948.1	V1104.1
M3	V948.2	V1104.2
M4	V948.3	V1104.3
M5	V948.4	V1104.4
M6	V948.5	V1104.5
M7	V948.6	V1104.6
M8	V948.7	V1104.7
M9	V949.0	V1105.0
M10	V949.1	V1105.1
M11	V949.2	V1105.2
M12	V949.3	V1105.3
M13	V949.4	V1105.4
M14	V949.5	V1105.5
...	...	..

15.6 Analoge Merker

Die analogen Merker eines LOGO können über den V-Bereich gelesen oder beschrieben werden. Dazu ist bei den Lese- und Schreibfunktionen von ComDrvS7 (z.B. MPI6_ReadWord oder MPI6_WriteWord) der Operandenbereich "V" anzugeben.
Das Schreiben von analogen Merkern ist möglich, allerdings wird der Status eines gesteuerten analogen Merkers unter Umständen vom Schaltprogramm im LOGO überschrieben.

15.6.1 Adressierung der analogen Merker

Die Zuordnung der analogen Merker ist in der folgenden Tabelle zu sehen:

LOGO Merker	Adressierung in ComDrvS7 0BA7	Adressierung in ComDrvS7 0BA8
AM1	VW952	VW1118
AM2	VW954	VW1120
AM3	VW956	VW1122
AM4	VW958	VW1124
AM5	VW960	VW1126
AM6	VW962	VW1128
AM7	VW964	VW1130
AM8	VW966	VW1132
AM9	VW968	VW1134
AM10	VW970	VW1136
AM11	VW972	VW1138
AM12	VW974	VW1140
AM13	VW976	VW1142
AM14	VW978	VW1144
AM15	VW980	VW1146
AM16	VW982	VW1148
AM17-AM63	-	VW1244

15.7 Analoge und digitale Netzwerk-Eingänge und Netzwerk-Ausgänge

In den LOGO Geräten ab 0BA7 können digitale und analoge Netwerkeingänge und Netzwerkausgänge im Schaltprogramm verwendet werden. Jeder dieser Blöcke wird dabei einer Adresse im Variablenspeicher (V-Bereich) zugewiesen. Dabei stehen die Bytes VB0 bis VB850 zur Verfügung.
Mit den Lese- und Schreibfunktionen von ComDrvS7 (z.B. MPI6_ReadWord oder MPI6_WriteWord) kann dieser Bereich gelesen oder beschrieben werden. Dabei ist bei den Funktionen der Operandenbereich "V" anzugeben.

Der Zugriff kann über die Bit-, Byte-, Word- oder DWord-Funktionen erfolgen.

16 Notwendige Einstellung in der Hardwarekonfiguration einer S7-1500® und S7-1200 (ab Firmware V4) von Siemens

Damit ComDrvS7 auf eine CPU der Reihe S7-1500 (und S7-1200 ab Firmware V4) von Siemens zugreifen kann, muss in der Hardwarekonfiguration der CPU die Option **"Zugriff über PUT/GET-Kommunikation durch entfernten Partner (PLC, HMI, OPC..) erlauben"**
selektiert sein.
Diese Option ist in den CPU-Eigenschaften innerhalb des Registers "Allgemein" und der Rubrik "Schutz" zu finden.
Die "höchste" einstellbare Schutzstufe ist der "HMI-Zugriff". Dabei können PG-Zugriffe über ein Passwort geschützt werden.
Nachfolgend ist eine Ansicht mit diesen Einstellungen zu sehen:

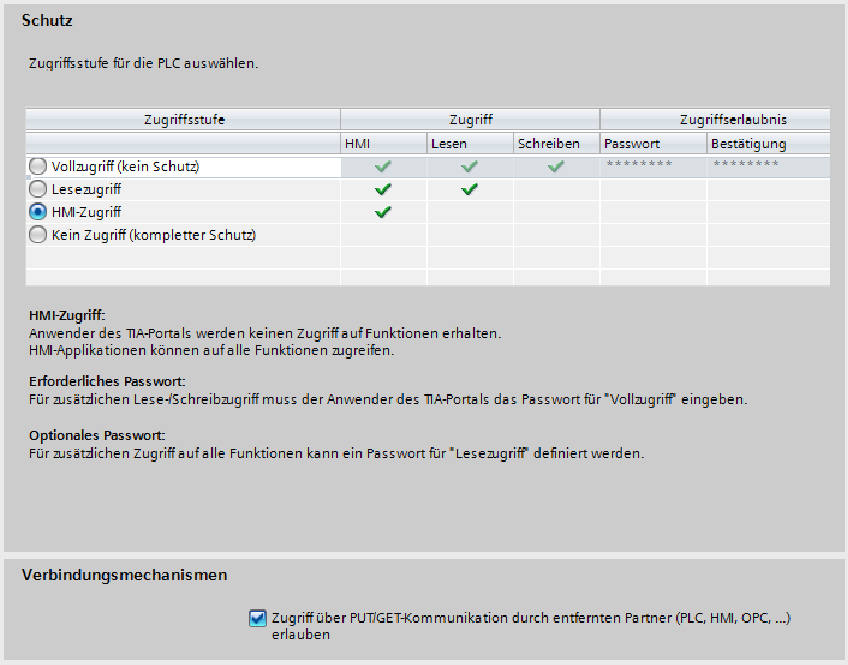


Bild: Notwendige Einstellungen in einer CPU-1500 und S7-1200 ab Firmware V4