IPS7 Members

Namespace: IPS7Lnk

Assemblies: IPS7LnkNet.Advanced.dll, IPS7LnkNet.Advanced.dll

The [IPS7](#) type exposes the following members.

Constructors

IPS7()

Referenznummer auf die Verbindung in der IPS7Lnk.dll.

C#

```
public IPS7()
```

Methods

Close()

Dient zur Deinitialisierung der Verbindung, Speicher wird freigegeben und die TCP/IP-Verbindung wird getrennt.

C#

```
public int Close()
```

Returns

[Int32](#)

0 Alles OK Speicher wieder freigegeben und Verbindung, wenn vorhanden geschlossen -3 Mit der angegebenen Referenznummer wurde kein IPS7Open durchgeführt Haben Sie IPS7Open aufgerufen ?. -99 Die Referenznummer ist ungültig -----

Connect()

Führt einen Explizitenverbindungsaufbau zur SPS aus.

C#

```
public int Connect()
```

Returns

[Int32](#)

1: Die Verbindung wurde aufgebaut, < 0 siehe: [Result](#)

GetConnectStatus()

Prüft den TCP/IP Verbindungsstatus zur SPS

C#

```
public int GetConnectStatus()
```

Returns

[Int32](#)

1 Verbindung ist OKay. 0 die Verbindung ist unterbrochen

GetCopyrightEntry(ref String)

Auslesen des Copyrighteintrags in der SPS in einen String

C#

```
public int GetCopyrightEntry(ref string Str)
```

Parameters

[Str](#) [String](#)

Ref auf String für die Information

Returns

[Int32](#)

siehe: [Result](#)

GetDBLen(UInt16)

Ermitteln der DB Länge

C#

```
public int GetDBLen(ushort DBNr)
```

Parameters

[DBNr](#) [UInt16](#)

Nummer des DB

Returns

[Int32](#)

>= Länge in Bytes < 0 see Result Wenn DB nicht vorhanden: S7.Result.E_BLOCK_NOT_EXIST

GetDBList()

C#

```
public int GetDBList()
```

Returns[Int32](#)

GetLastPackets()

Liefert die bis zu 20 letzten Telegramme als ByteArray Das Array ist 3-dimensionel [x][y][z] x = 0: Sendetelegramm x = 1: Antworttelegramm y = 0 .. 19: Index auf die entsprechende Telegrammnummer z = 0 .. n: die einzelnen Bytes des Telegramms

C#

```
public byte[][][] GetLastPackets()
```

Returns[Byte\[\]\[\]\[\]](#)

3-dimensionles Byte-Array

GetLocationDesignation(ref String)

Auslesen Beschreibung für den Ort in der SPS in einen String

C#

```
public int GetLocationDesignation(ref string Str)
```

Parameters[Str](#) [String](#)

Ref auf String für die Information

Returns[Int32](#)

siehe: [Result](#)

GetMMCSNr(ref String)

Auslesen des MMC-Seriennummer in einen String

C#

```
public int GetMMCSNr(ref string Str)
```

Parameters

Str String

Ref auf String für die Information

Returns

Int32

siehe: [Result](#)

GetModuleName(ref String)

Auslesen Baugruppen Name der SPS in einen String

C#

```
public int GetModuleName(ref string Str)
```

Parameters

Str String

Ref auf String für die Information

Returns

Int32

siehe: [Result](#)

GetModuleSNr(ref String)

Auslesen der Seriennummer der SPS in einen String

C#

```
public int GetModuleSNr(ref string Str)
```

Parameters

Str String

Ref auf String für die Information

Returns

[Int32](#)

siehe: [Result](#)

GetModuleName(ref String)

Auslesen des CPU-Typs / Baugruppentyps in einen String

C#

```
public int GetModuleName(ref string Str)
```

Parameters

[Str](#) [String](#)

Ref auf String für die Information

Returns

[Int32](#)

siehe: [Result](#)

GetOEMId(ref String)

Auslesen der OEM-ID in der SPS in einen String, falls vorhanden

C#

```
public int GetOEMId(ref string Str)
```

Parameters

[Str](#) [String](#)

Ref auf String für die Information

Returns

[Int32](#)

siehe: [Result](#)

GetPlantIdName(ref String)

Auslesen des Anlagennamen in der SPS in einen String

C#

```
public int GetPlantIdName(ref string Str)
```

Parameters

Str String

Ref auf String für die Information

Returns

Int32

siehe: [Result](#)

GetPLCName(ref String)

Auslesen Name der SPS in einen String

C#

```
public int GetPLCName(ref string Str)
```

Parameters

Str String

Ref auf String für die Information

Returns

Int32

siehe: [Result](#)

GetPLCTime(ref DateTime)

Auslesen der SPS-Uhrzeit in einen DateTime

C#

```
public int GetPLCTime(ref DateTime PlcTime)
```

Parameters

PlcTime DateTime

Ref auf DateTime für die Uhrzeit

Returns

Int32

siehe: [Result](#)

GetSockErr()

Liefert den letzten Socket-Fehler, der bei der TCP/IP-Kommunikation aufgetreten ist.

C#

```
public int GetSockErr()
```

Returns

Int32

Socketfehler: Diese Liste erhebt keinen Anspruch auf Vollständigkeit ^ ^ NameCodeBedeutung ^ | |
WSAEINTR10004Aufruf wurde abgebrochen | | | WSAEACCESS10013Zugriffsfehler | | |
WSAEFAULT10014Parameter sind falsch | | | WSAEINVAL100221.Andere Funktion muß vorher aufgerufen werden 2. Socket ist schon an Adresse gebunden 3. Socket noch nicht an Adresse gebunden bzw. schon verbunden. | | | WSAEMFILE10024Ressourcen fehlen (Dateien, Warteschlangen) | | |
WSAEWOULDBLOCK10035Aufruf würde blockieren | | | WSAEINPROGRESS10036Parallele Aufrufe nicht erlaubt | | | WSAEALREADY10037Abgebrochene Routine trotzdem schon fertig | | |
WSAENOTSOCK10038Kein gültiger Socket angegeben | | | WSAEDESTADDRREQ10039Zieladresse benötigt | | | WSAEMSGSIZE10040Datagramm zu groß, wurde abgeschnitten | | |
WSAENOPROTOOPT10042Unbekannte Socket-Option | | | WSAEPROTONOSUPPORT10043Protokoll wird nicht unterstützt | | | WSAESOCKTNOSUPPORT10044Sockettyp wird in angegebener Adressfamilie nicht unterstützt | | | WSAEOPNOTSUPP10045Dieser Sockettyp wird nicht unterstützt | | |
WSAEPFNOSUPPORT10046Protokollfamilie wird nicht unterstützt | | |
WSAEAFNOSUPPORT10047Adressfamilie wird nicht unterstützt | | | WSAEADDRINUSE10048IP-Adresse bzw. Port werden schon/nach benutzt | | | WSAEADDRNOTAVAIL10049Port/Adresse nicht verfügbar | | |
WSAENETDOWN10050Netzwerk reagiert nicht | | | WSAENETUNREACH10051Netzwerk kann nicht erreicht werden | | | WSAENETRESET10052Verbindung durch TCP/IP zurückgesetzt | | |
WSAECONNABORTED10053Verbindung durch TCP/IP abgebrochen | | | WSAECONNRESET10054Partner hat Verbindung zurückgesetzt | | | WSAENOBUFS10055Ressourcen fehlen (Interner Pufferspeicher) | | |
WSAEISCONN10056Socket ist schon verbunden | | | WSAENOTCONN10057Socket ist noch nicht verbunden | | | WSAESHUTDOWN10058Andere Seite hat Verbindung einseitig beendet | | |
WSAETIMEDOUT10060Aufruf dauert zu lange, daher Abbruch | | | WSAECONNREFUSED10061Angerufener möchte keinen Verbindungsaufbau | | | WSAEHOSTUNREACH10065Host nicht erreichbar | | |
WSASYSNOTREADY10091Netzwerk nicht zur Kommunikation bereit | | |
WSAVERNOTSUPPORTED10092gewünschte Winsock-Version wird nicht unterstützt | | |
WSANOTINITIALISED10093vSocket.Initialize muß aufgerufen werden | | |
WSAHOST_NOT_FOUND11001DNS-Server nicht gefunden | | | WSATRY_AGAIN11002Gesuchter Rechner nicht gefunden | | | WSANO_RECOVERY11003Nicht behebbare Fehler | | | WSANO_DATA11004Keine Namensdaten vorhanden |

Open(String, UInt32, UInt32, UInt32, UInt32, UInt32)

Dient zur Initialisierung der Verbindung, dort wird nur Speicher vorbereitet. Erst beim ersten Aufruf der Lese- oder Schreibfunktionen wird die TCP/IP-Verbindung automatisch gestartet. Es wird eine OP-

Verbindung angelegt!

C#

```
public int Open(string IPAdr, uint Rack, uint Slot, uint RxTimeout, uint TxTimeout, uint ConnectTimeout)
```

Parameters

IPAdr String

IP-Adresse der SPS im Format: xxx.xxx.xxx.xxx. Beispiel: 192.168.0.100

Rack UInt32

Racknummer der Ziel-CPU

Slot UInt32

Slotnummer der Ziel-CPU

RxTimeout UInt32

Timeout in ms für Warten auf TCP/IP-Paket von der SPS. 0 = Standardeinstellung = 5000 ms

TxTimeout UInt32

Timeout in ms für Senden eines TCP/IP-Paketes an die SPS. 0 = Standardeinstellung = 5000 ms

ConnectTimeout UInt32

Timeout in ms für Warten auf Verbindungsaufbau mit SPS. 0 = Standardeinstellung = 5000 ms (5sec.)muss bei Bedarf verlängert werden.

Returns

Int32

Die Rückgabe ist die Referenznummer für diese Verbindung. ^ Wert ^ Beschreibung ^ | >= 0 | Alles OK. Result.E_NOERR | | E_NORESOURCE -2 | Keine Ressourcen mehr frei. Maximale Anzahl an verfügbaren Verbindungen erreicht. | | E_ALREADY_OPENED -21 | Verbindung bereits geöffnet. Nur bei .NET Es wurde bereits ein Open ausgeführt. |

OpenEx(String, UInt32, UInt32, UInt32, UInt32, UInt32, UInt32, UInt32, UInt32, Byte[], Byte[])

Dient zur Initialisierung der Verbindung, dort wird nur Speicher vorbereitet. Erst beim ersten Aufruf der Lese- oder Schreibfunktionen wird, die TCP/IP-Verbindung automatisch gestartet. Die Art der Verbindung OP/PG und der Typ der CPU werden über separate Eingangsparameter ausgewählt. Weiter kann hier festgelegt werden, ob ein Routing auf eine andere CPU über ein SubNetz erreicht werden soll. (Hierfür muß der Treiber mit der Routing-Option lizenziert sein!)

C#

```
public int OpenEx(string IPAdr, uint Rack, uint Slot, uint SubNetId, uint DstMPIAdr, uint
AccessMode, uint RxTimeout, uint TxTimeout, uint ConnectTimeout, byte[] LocalTSAP = null,
byte[] RemoteTSAP = null)
```

Parameters

IPAdr String

IP-Adresse der SPS im Format: xxx.xxx.xxx.xxx. Beispiel: 192.168.0.100

Rack UInt32

Racknummer der Ziel-CPU

Slot UInt32

Slotnummer der Ziel-CPU

SubNetId UInt32

Subnetz-ID, wenn über ein Subnetzu zugegriffen werden soll. In der Step-S7 Software wird die Adresse z.B. so dargestellt: 0035 - 0001, geben Sie dann an 0x00350001 Wird nur bei AccessMode 10 oder 11 verwendet. Siehe AccessModes [AM](#)

DstMPIAdr UInt32

ZielMPI-Adresse, wenn über ein Subnetz die Verbindung aufgebaut werden soll. Siehe AccessMode 10 und 11! Siehe AccessModes [AM](#)

AccessMode UInt32

Siehe AccessModes [AM](#)

RxTimeout UInt32

Timeout in ms für Warten auf TCP/IP-Paket von der SPS. 0 = Standardeinstellung = 5000 ms

TxTimeout UInt32

Timeout in ms für Senden eines TCP/IP-Paketes an die SPS. 0 = Standardeinstellung = 5000 ms

ConnectTimeout UInt32

Timeout in ms für Warten auf Verbindungsaufbau mit SPS. 0 = Standardeinstellung = 5000 ms (5sec.)muss bei Bedarf verlängert werden.

LocalTSAP Byte[]

RemoteTSAP Byte[]

Returns

Int32

Die Rückgabe ist die Referenznummer für diese Verbindung. ^ Wert ^ Beschreibung ^ | >= 0 | Alles OK.[Result.E_NOERR](#) | | [E_NORESOURCE](#) -2 | Keine Ressourcen mehr frei. Maximale Anzahl an verfügbaren

Verbindungen erreicht. | | E_ALREADY_OPENED -21 | Verbindung bereits geöffnet. Nur bei .NET Es wurde bereits ein Open ausgeführt. |

OpenPG(String, UInt32, UInt32, UInt32, UInt32, UInt32)

Dient zur Initialisierung der Verbindung, dort wird nur Speicher vorbereitet. Erst beim ersten Aufruf der Lese- oder Schreibfunktionen wird, die TCP/IP-Verbindung automatisch gestartet. Es wird eine PG-Verbindung angelegt!

C#

```
public int OpenPG(string IPAdr, uint Rack, uint Slot, uint RxTimeout, uint TxTimeout, uint ConnectTimeout)
```

Parameters

IPAdr String

IP-Adresse der SPS im Format: xxx.xxx.xxx.xxx. Beispiel: 192.168.0.100

Rack UInt32

Racknummer der Ziel-CPU

Slot UInt32

Slotnummer der Ziel-CPU

RxTimeout UInt32

Timeout in ms für Warten auf TCP/IP-Paket von der SPS. 0 = Standardeinstellung = 5000 ms

TxTimeout UInt32

Timeout in ms für Senden eines TCP/IP-Paketes an die SPS. 0 = Standardeinstellung = 5000 ms

ConnectTimeout UInt32

Timeout in ms für Warten auf Verbindungsaufbau mit SPS. 0 = Standardeinstellung = 5000 ms (5sec.)muss bei Bedarf verlängert werden.

Returns

Int32

Die Rückgabe ist die Referenznummer für diese Verbindung. ^ Wert ^ Beschreibung ^ | >= 0 | Alles OK.Result.E_NOERR | | E_NORESOURCE -2 | Keine Ressourcen mehr frei. Maximale Anzahl an verfügbaren Verbindungen erreicht. | | E_ALREADY_OPENED -21 | Verbindung bereits geöffnet. Nur bei .NET Es wurde bereits ein Open ausgeführt. |

OpenS7200(String, UInt32, UInt32, UInt32, UInt32, UInt32)

Dient zur Initialisierung der Verbindung, dort wird nur Speicher vorbereitet. Erst beim ersten Aufruf der

Lese- oder Schreibfunktionen wird, die TCP/IP-Verbindung automatisch gestartet. Es wird eine S7-200-Verbindung angelegt!

C#

```
public int OpenS7200(string IPAdr, uint Rack, uint Slot, uint RxTimeout, uint TxTimeout, uint ConnectTimeout)
```

Parameters

IPAdr String

IP-Adresse der SPS im Format: xxx.xxx.xxx.xxx. Beispiel: 192.168.0.100

Rack UInt32

Racknummer der Ziel-CPU

Slot UInt32

Slotnummer der Ziel-CPU

RxTimeout UInt32

Timeout in ms für Warten auf TCP/IP-Paket von der SPS. 0 = Standardeinstellung = 5000 ms

TxTimeout UInt32

Timeout in ms für Senden eines TCP/IP-Paketes an die SPS. 0 = Standardeinstellung = 5000 ms

ConnectTimeout UInt32

Timeout in ms für Warten auf Verbindungsaufbau mit SPS. 0 = Standardeinstellung = 5000 ms (5sec.)muss bei Bedarf verlängert werden.

Returns

Int32

Die Rückgabe ist die Referenznummer für diese Verbindung. ^ Wert ^ Beschreibung ^ | >= 0 | Alles OK. [Result.E_NOERR](#) | | E_NORESOURCE -2 | Keine Ressourcen mehr frei. Maximale Anzahl an verfügbaren Verbindungen erreicht. | | E_ALREADY_OPENED -21 | Verbindung bereits geöffnet. Nur bei .NET Es wurde bereits ein Open ausgeführt. |

Rd(Char, UInt32, UInt32, ref Double)

Liest einen einzelnen Real-Wert aus der SPS.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, ref double Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo [UInt32](#)

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start [UInt32](#)

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll.

Buf [Double](#)

Referenz auf den double im Anwenderprogramm.

Returns

[Int32](#)

siehe: [Result](#)

Rd(Char, UInt32, UInt32, ref Int16)

Liest einen einzelnen short-Wert (16 Bit) aus der SPS.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, ref short Buf)
```

Parameters

DataArea [Char](#)

Datenbereich, siehe: [DA](#)

DBNo [UInt32](#)

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start [UInt32](#)

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Buf [Int16](#)

Referenz auf den short im Anwenderprogramm.

Returns

[Int32](#)

siehe: [Result](#)

Rd(Char, UInt32, UInt32, ref Int32)

Liest einen einzelnen int-Wert (32 Bit) aus der SPS.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, ref int Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Buf Int32

Referenz auf den int im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, ref UInt16)

Liest einen einzelnen ushort-Wert (16 Bit) aus der SPS.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, ref ushort Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine! Dies gilt auch für Datenbausteine!

Buf UInt16

Referenz auf den ushort im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, ref UInt32)

Liest eines einzelnen uint-Wertes (16 Bit) aus der SPS.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, ref uint Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Buf UInt32

Referenz auf den uint im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, Byte[])

Liest byteweise Daten aus der SPS in ein byte-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, byte[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des gewünschten Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes, ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu lesenden bytes.

Buf Byte[]

Pointer auf ein byte-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, Char[])

Liest byteweise Daten aus der SPS in ein char-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, char[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des gewünschten Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes, ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu lesenden Chars.

Buf Char[]

Pointer auf ein Char-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, Double[])

Liest ein Real-Array Daten aus der SPS in ein double-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, double[] Buf)
```

Parameters**DataArea** Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu lesenden doubles (Real-Werte).

Buf Double[]

Pointer auf ein double-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, Int16[])

Liest wortweise Daten aus der SPS in ein short (16-Bit)-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, short[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu lesenden shorts.

Buf Int16[]

Pointer auf ein short-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, Int32[])

Liest doppelwortweise Daten aus der SPS in ein int(32-Bit)-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, int[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt

berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu lesenden ints (32-Bit-Werte).

Buf Int32[]

Pointer auf ein int-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, ref String)

Liest einen String aus der SPS

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, ref string Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des gewünschten Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des Bytes, ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Maximale Länge des zulesenden Strings

Buf String

ref auf string.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, UInt16[])

Liest wortweise Daten aus der SPS in ein ushort (16-Bit)-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, ushort[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu lesenden ushorts.

Buf UInt16[]

Pointer auf ein ushort-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Rd(Char, UInt32, UInt32, UInt32, UInt32[])

Liest doppelwortweise Daten aus der SPS in ein uint(32-Bit)-Array.

C#

```
public int Rd(char DataArea, uint DBNo, uint Start, uint Cnt, uint[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu lesenden uints (32-Bit-Werte).

Buf UInt32[]

Pointer auf ein uint-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdB(Char, UInt32, UInt32, UInt32, Byte*)

Liest einen Anzahl bytes aus der SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!

C#

```
public int RdB(char DataArea, uint DBNo, uint Start, uint Cnt, byte *Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Wird nicht ausgewertet, da mit dieser Funktion kein Zugriff auf Db / DX möglich ist.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Cnt UInt32

Anzahl der bytes, die gelesen werden sollen.

Buf Byte*

Pointer auf den byte-Buffer.

Returns

Int32

siehe: [Result](#)

RdBit(Char, UInt32, UInt32, UInt32, ref Byte)

Liest ein einzelnes Bit aus der SPS in eine byte Variable ein

C#

```
public int RdBit(char DataArea, uint DBNo, uint ByteNo, uint Bit, ref byte Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

ByteNo UInt32

Nummer des bytes ab welchem gelesen werden soll.

Bit UInt32

Bitnummer (0 - 7) des bytes, welches gelesen werden soll.

Buf Byte

Referenz auf das "byte" im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdBit(Char, UInt32, UInt32, UInt32, ref UInt32)

Liest ein einzelnes Bit aus der SPS in eine uint Variable ein

C#

```
public int RdBit(char DataArea, uint DBNo, uint ByteNo, uint Bit, ref uint Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

ByteNo UInt32

Nummer des bytes ab welchem gelesen werden soll.

Bit UInt32

Bitnummer (0 - 7) des bytes, welches gelesen werden soll.

Buf UInt32

Referenz auf den "uint" im Anwenderprogramm.

Returns

[Int32](#)

siehe: [Result](#)

RdL(Char, UInt32, UInt32, ref Double)

Liest einen einzelnen LReal-Wert aus der SPS.

C#

```
public int RdL(char DataArea, uint DBNo, uint Start, ref double Buf)
```

Parameters**DataArea Char**

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll.

Buf Double

Referenz auf den double im Anwenderprogramm.

Returns

[Int32](#)

siehe: [Result](#)

RdL(Char, UInt32, UInt32, ref Int64)

Liest einen einzelnen LINT-Wert (64 Bit) aus der SPS.

C#

```
public int RdL(char DataArea, uint DBNo, uint Start, ref long Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Buf Int64

Referenz auf den int im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdL(Char, UInt32, UInt32, ref UInt64)

Liest einen einzelnen ULINT-Wertes (46 Bit) aus der SPS.

C#

```
public int RdL(char DataArea, uint DBNo, uint Start, ref ulong Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Buf UInt64

Referenz auf den uint im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdL(Char, UInt32, UInt32, UInt32, Double[])

Liest ein LReal-Array Daten aus der SPS in ein double-Array.

C#

```
public int RdL(char DataArea, uint DBNo, uint Start, uint Cnt, double[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu lesenden doubles (Real-Werte).

Buf Double[]

Pointer auf ein double-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdL(Char, UInt32, UInt32, UInt32, Int64[])

Liest aus der SPS in ein LINT(64-Bit)-Array.

C#

```
public int RdL(char DataArea, uint DBNo, uint Start, uint Cnt, long[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu lesenden ints (32-Bit-Werte).

Buf Int64[]

Pointer auf ein int-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdL(Char, UInt32, UInt32, UInt32, UInt64[])

Liest aus der SPS in ein ULINT(64-Bit)-Array.

C#

```
public int RdL(char DataArea, uint DBNo, uint Start, uint Cnt, ulong[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Lesevorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu lesenden uints (32-Bit-Werte).

Buf UInt64[]

Pointer auf ein uint-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

RdMulti(IPS7RdMulti[], UInt32)

ACHTUNG! Dieser Aufruf darf nur verwendet werden, wenn Ihre Applikation im Unsafe-Mode läuft und der Zielspeicher während des Aufrufs gefixed ist! Grund: Der Garbage Collector könnte während des Aufrufs die Daten verschieben. Siehe Keyword: fixed. Ansonsten ist die Methode "RdMultiBuffered" und die Methode "GetData" von IPS7RdMulti aufzurufen. Führt einen gemischten Readauftrag aus. Zuvor ist ein Array vom Typ `IPS7RdMulti` zu initialisieren. Siehe aufgeführtes Beispiel: Die Funktion führt eigenständig die Datenkonvertierung des S7-Datentyps in den PC-Datentyp durch. Beispiel: Es sollen Bits (Boolean) von der SPS gelesen werden. Der Zusatz soll in einem Array von int gespeichert werden. So ist ein Requesteintrag wie folgt zu initialisieren `int EBits[32]; Rq.Bit((char)'E', 0, 4, 0, 32, EBits);` Durch den Datentyp `int` der Variable `EBits` wird die entsprechende überladene Funktion aufgerufen. Diese setzte im Request-Eintrag den PC-Datentyp auf INT32. Somit stehen nach Aufruf der `S7RQMulti` Funktion in folgender Form in der Variablen: `E4.0 = EBits[0]` (1: wenn Bit = 1, 0: wenn Bit = 0)

`E4.1 = EBits[1]` (1: wenn Bit = 1, 0: wenn Bit = 0)

`E4.2 = EBits[2]` (1: wenn Bit = 1, 0: wenn Bit = 0)

etc.

So könnte auch ein 16-Bit-Wert der SPS in einen Realwert im PC konvertiert werden. D.h. wenn Wert der SPS "50" ist, so ist das bei einem Double im PC "50.0". Desweiteren führt `RdMulti` eine Optimierung bzglw. der Datenblöcke, die gelesen werden durch.

ACHTUNG: Die Lesereihenfolge entspricht nicht der Anordnung in den Requestblöcken. Jedoch wird auf die Konsistenz der Daten z.B. bei 16/32 Bit Werten geachtet.

C#

```
public int RdMulti(IPS7RdMulti[] pRqList, uint Cnt)
```

Parameters

pRqList IPS7RdMulti[]

Ein Array von Requestblöcken des Typs IPS7RdMulti

Cnt UInt32

Anzahl der Requestblöcke

Returns

Int32

0 alle Aufträge wurden mit Erfolg ausgeführt

!= 0 bei mindestens einem Auftrag ist ein Fehler aufgetreten. Für jeden RequestBlock muss .Result zu geprüft werden.

Siehe: [Result](#)

RdMultiBuffered(IPS7RdMulti[], UInt32)

Führt einen gemischten Readauftrag aus, die gelesenen Daten werden im sichern Zustand in den .Net - Speicher gepuffert. Der Garbage Collector wird während des Aufrufs daran gehindert die Daten verschieben. Zuvor ist ein Array vom Type `IPS7RdMulti` zu initialisieren. Im Anschluss sind die Daten über die `GetData`-Methode in den Applikationsspeicher zu holen. `IPS7RdMulti` Siehe aufgeführtes Beispiel: Die Funktion führt eigenständig die Datenkonvertierung des S7-Datentyps in den PC-Datentyp durch. Beispiel: Es sollen Bits (Boolean) von der SPS gelesen werden. Der Zusatz soll in einem Array von `int` gespeichert werden. So ist ein Requesteintrag wie folgt zu initialisieren `int EBits[32]; Rq.Bit((char)'E', 0, 4, 0, 32, EBits);` Durch den Datentyp `int` der Variable `EBits` wird die entsprechende überladene Funktion aufgerufen. Diese setzte im Request-Eintrag den PC-Datentype auf `INT32`. Somit stehen nach Aufruf der `S7RQMulti` Funktion in folgender Form in der Variablen: `E4.0 = EBits[0]` (1: wenn Bit = 1, 0: wenn Bit = 0)

`E4.1 = EBits[1]` (1: wenn Bit = 1, 0: wenn Bit = 0)

`E4.2 = EBits[2]` (1: wenn Bit = 1, 0: wenn Bit = 0)

etc.

So könnte auch ein 16-Bit-Wert der SPS in einen Realwert im PC konvertiert werden. D.h wenn Wert der SPS "50" ist, so ist das bei einem Double im PC "50.0". Desweiteren führt `RdMulti` ein Optimierung bzglw. der Datenblöcke, die gelesen werden durch.

ACHTUNG: Die Lesereihenfolge entspricht nicht der Anordnung in den Requestblöcken. Jedoch wird auf die Konsistenz der Daten z.B. bei 16/32 Bit werten geachtet.

C#

```
public int RdMultiBuffered(IPS7RdMulti[] RqList, uint Cnt)
```

Parameters

`RqList` `IPS7RdMulti[]`

Ein Array von Requestblöcken des Typs `IPS7RdMulti`

Cnt UInt32

Anzahl der Requestblöcke

Returns

Int32

0 alle Aufträge wurden mit Erfolg ausgeführt

!= 0 bei mindestens einem Auftrag ist ein Fehler aufgetreten. Für jeden RequestBlock muss .Result zu geprüft werden.

Siehe: [Result](#)

RdMultiCalcPacketCnt(IPS7RdMulti[], UInt32)

Berechnet die Anzahl der Pakete, die für diesen gemischten Readauftrag nötig sind. Zuvor ist ein Array vom Type [IPS7RdMulti](#) zu initialisieren.

C#

```
public int RdMultiCalcPacketCnt(IPS7RdMulti[] pRqList, uint Cnt)
```

Parameters

[pRqList](#) [IPS7RdMulti\[\]](#)

[Cnt](#) [UInt32](#)

Returns

Int32

>=0 Anzahl der Pakete, < 0 Fehler aufgetreten

RdW(Char, UInt32, UInt32, UInt32, UInt16*)

Liest einen Anzahl ushort aus der SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!

C#

```
public int RdW(char DataArea, uint DBNo, uint Start, uint Cnt, ushort *Buf)
```

Parameters

[DataArea](#) [Char](#)

Datenbereich, siehe: [DA](#)

DBNo UInt32

Wird nicht ausgewertet, da mit dieser Funktion kein Zugriff auf Db / DX möglich ist.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Cnt UInt32

Anzahl der ushort, die gelesen werden sollen.

Buf UInt16*

Pointer auf den ushort-Buffer.

Returns

Int32

siehe: [Result](#)

ResetBit(Char, UInt32, UInt32, UInt32)

Setzt ein einzelnes Bit in der SPS zurück.

C#

```
public int ResetBit(char DataArea, uint DBNo, uint ByteNo, uint Bit)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

ByteNo UInt32

Nummer des bytes, auf welches zugegriffen werden soll.

Bit UInt32

Bitnummer (0 - 7) des bytes, auf welches zugegriffen werden soll.

Returns

Int32

siehe: [Result](#)

SetBit(Char, UInt32, UInt32, UInt32)

Setzt ein einzelnes Bit in der SPS.

C#

```
public int SetBit(char DataArea, uint DBNo, uint ByteNo, uint Bit)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

ByteNo UInt32

Nummer des bytes, auf welches zugegriffen werden soll.

Bit UInt32

Bitnummer (0 - 7) des bytes, auf welches zugegriffen werden soll.

Returns

Int32

siehe: [Result](#)

SetKeepAlive(UInt32, UInt32)

Setzt individuelle TCP/IP KeepAlive Zeiten für diese Verbindung. Wenn die Standardwerte gelten sollen muss diese Funktion nicht verwendet werden. Wenn ja, sollte Sie nach den "Open"-Aufruf ausgeführt werden. Findet innerhalb der Zeit **AliveTime**(ms) kein Datenverkehr auf der TCP/IP-Verbindung statt, so wird ein KeepAlive-Telegramm gesendet, um die Verbindung zu prüfen. Wird bei dieser Prüfung ein Fehler festgestellt, sendet der IP-Stack innerhalb der Zeit **AliveInterval** (ms) ein nächstes KeepAlive-Telegramm. Dies wird einige male innerhalb der Zeit **AliveInterval** wiederholt (bei Windows 6 mal). War der Vorgang nicht erfolgreich, wird die Verbindung beendet.

C#

```
public int SetKeepAlive(uint AliveInterval, uint AliveTime)
```

Parameters

AliveInterval UInt32

Das Intervall in ms, in welchem KeepAlive Telegramme wiederholt werden. Dieses wird aktiv, wenn ein Fehler beim Senden / Empfangen eines KeepAlive-Telegramms aufgetreten ist.

AliveTime UInt32

Intervall in ms, in welchem KeepAlive Telegramme gesendet werden

Returns

[Int32](#)

0 Aufruf erfolgreich. < 0 Aufruf war nicht erfolgreich

SetPLCTime(DateTime)

Setzen der SPS-Uhrzeit mit einen DateTime

C#

```
public int SetPLCTime(DateTime PlcTime)
```

Parameters

[PlcTime](#) [DateTime](#)

DateTime mit der Uhrzeit, die su setzen ist.

Returns

[Int32](#)

siehe: [Result](#)

SolveSiemensPDUBug(Boolean)

Selects whether the Siemsn PDU-Bug should be solved. Background: some S7 400 PLCs have a problem processing the maximum count of read request sent. e.g. if PDU == 480 Byte max request count is 39. Only 38 were processed and responded. Bye default this Bug is not solved (bDoSolved = false), normaly S7 works well. (since 1.1.77.12)

C#

```
public void SolveSiemensPDUBug(bool bDoSolve)
```

Parameters

[bDoSolve](#) [Boolean](#)

Wr(Char, UInt32, UInt32, Byte)

Schreibt einen einzelnen byte-Wert (8-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, byte Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Value Byte

Wert als byte.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, Double)

Schreibt einen einzelnen double-Wert (Real) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, double Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Value Double

Wert als double.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, Int16)

Schreibt einen einzelnen short-Wert (16-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, short Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Value Int16

Wert als short.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, Int32)

Schreibt einen einzelnen int-Wert (32-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, int Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Value Int32

Wert als int.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, String)

Schreibt einen String in die SPS. Es werden maximal 254 Zichen geschrieben Achtung: Ist der reservierte Bereich in der SPS zu klein, so wird Speicher in der SPS überschrieben

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, string Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des Bytes, ab welchem der Lesevorgang gestartet werden soll.

Buf String

string, der geschrieben werden soll

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt16)

Schreibt einen einzelnen ushort-Wert (16-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, ushort Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Value UInt16

Der Wert als ushort.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32)

Schreibt einen einzelnen uint-Wert (32-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Value UInt32

Wert als uint.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, Byte[])

Schreibt byteweise ein byte-Array in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, byte[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes, ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu schreibenden bytes.

Buf Byte[]

Pointer auf das byte-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, Char[])

Schreibt byteweise ein char-Array in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, char[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes, ab welchem der Lesevorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu schreibenden chars.

Buf Char[]

Pointer auf das char-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, Double[])

Schreibt ein double-Array (Real-Array) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, double[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Cnt UInt32

Anzahl der zu schreibenden doubles.

Buf Double[]

Pointer auf ein double-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, Int16[])

Schreibt wortweise ein short-Array (16-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, short[] Buf)
```

Parameters**DataArea** Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu schreibenden shorts.

Buf Int16[]

Pointer auf ein short-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, Int32[])

Schreibt doppelwortweise ein int-Array (16-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, int[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu schreibenden ints.

Buf Int32[]

Pointer auf ein int-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, UInt16[])

Schreibt wortweise ein ushort-Array (16-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, ushort[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes, ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu schreibenden ushort.

Buf UInt16[]

Pointer auf ein ushort-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

Wr(Char, UInt32, UInt32, UInt32, UInt32[])

Schreibt doppelwortweise ein uint-Array (32-Bit) in die SPS.

C#

```
public int Wr(char DataArea, uint DBNo, uint Start, uint Cnt, uint[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu schreibenden uints.

Buf UInt32[]

Pointer auf ein uint-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

WrB(Char, UInt32, UInt32, UInt32, Byte*)

Schreibt einen Anzahl bytes in die SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!

C#

```
public int WrB(char DataArea, uint DBNo, uint Start, uint Cnt, byte *Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Wird nicht ausgewertet, da mit dieser Funktion kein Zugriff auf Db / DX möglich ist.

Start UInt32

Nummer des bytes (bei M,E,A), ab welchem der Schreibvorgang gestartet werden soll.

Cnt UInt32

Anzahl der bytes, die geschrieben werden sollen.

Buf Byte*

Pointer auf den byte-Buffer.

Returns

Int32

siehe: [Result](#)

WrBit(Char, UInt32, UInt32, UInt32, UInt32)

Schreibt ein einzelnes Bit in die SPS.

C#

```
public int WrBit(char DataArea, uint DBNo, uint ByteNo, uint Bit, uint Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo [UInt32](#)

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

ByteNo [UInt32](#)

Nummer des bytes, auf welches zugegriffen werden soll.

Bit [UInt32](#)

Bitnummer (0 - 7) des bytes, auf welches zugegriffen werden soll.

Value [UInt32](#)

Zustand des Bits, 0 = 0 Bit wird zurückgesetzt. != 0 Bit wird gesetzt.

Returns

[Int32](#)

siehe: [Result](#)

WrL(Char, UInt32, UInt32, Double)

Schreibt einen einzelnen LReal (double) in die SPS.

C#

```
public int WrL(char DataArea, uint DBNo, uint Start, double Value)
```

Parameters

DataArea [Char](#)

Datenbereich, siehe: [DA](#)

DBNo [UInt32](#)

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start [UInt32](#)

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Value [Double](#)

Wert als double.

Returns

[Int32](#)

siehe: [Result](#)

WrL(Char, UInt32, UInt32, Int64)

Schreibt einen einzelnen LINT-Wert (64-Bit) in die SPS.

C#

```
public int WrL(char DataArea, uint DBNo, uint Start, long Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Value Int64

Wert als int.

Returns

Int32

siehe: [Result](#)

WrL(Char, UInt32, UInt32, UInt32, Double[])

Schreibt ein LReal-Array (double-Array) in die SPS.

C#

```
public int WrL(char DataArea, uint DBNo, uint Start, uint Cnt, double[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll.

Cnt [UInt32](#)

Anzahl der zu schreibenden doubles.

Buf [Double\[\]](#)

Pointer auf ein double-Array im Anwenderprogramm.

Returns

[Int32](#)

siehe: [Result](#)

WrL(Char, UInt32, UInt32, UInt32, Int64[])

Schreibtein LINTint-Array (64-Bit) in die SPS.

C#

```
public int WrL(char DataArea, uint DBNo, uint Start, uint Cnt, long[] Buf)
```

Parameters

DataArea [Char](#)

Datenbereich, siehe: [DA](#)

DBNo [UInt32](#)

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start [UInt32](#)

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt [UInt32](#)

Anzahl der zu schreibenden ints.

Buf [Int64\[\]](#)

Pointer auf ein int-Array im Anwenderprogramm.

Returns

[Int32](#)

siehe: [Result](#)

WrL(Char, UInt32, UInt32, UInt32, UInt64[])

Schreibt ein ULINT-Array (64-Bit) in die SPS.

C#

```
public int WrL(char DataArea, uint DBNo, uint Start, uint Cnt, ulong[] Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der zu schreibenden uints.

Buf UInt64[]

Pointer auf ein uint-Array im Anwenderprogramm.

Returns

Int32

siehe: [Result](#)

WrL(Char, UInt32, UInt32, UInt64)

Schreibt einen einzelnen ULINT-Wert (64-Bit) in die SPS.

C#

```
public int WrL(char DataArea, uint DBNo, uint Start, ulong Value)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Nummer des Datenbausteins. Wird nur bei Zugriff auf DB ausgewertet.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Value UInt64

Wert als uint.

Returns

Int32

siehe: [Result](#)

WrMulti(IPS7WrMulti[], UInt32)

C#

```
public int WrMulti(IPS7WrMulti[] pRqL, uint Cnt)
```

Parameters

pRqL IPS7WrMulti[]

Cnt UInt32

Returns

Int32

WrW(Char, UInt32, UInt32, UInt32, UInt16*)

Schreibt einen Anzahl ushort (16-Bit) in die SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!

C#

```
public int WrW(char DataArea, uint DBNo, uint Start, uint Cnt, ushort *Buf)
```

Parameters

DataArea Char

Datenbereich, siehe: [DA](#)

DBNo UInt32

Wird nicht ausgewertet, da mit dieser Funktion kein Zugriff auf Db / DX möglich ist.

Start UInt32

Nummer des bytes , ab welchem der Schreibvorgang gestartet werden soll. In der SPS überlappen sich z.B MW0 und MW1. MW 0 = MB0 und MB1, MW1 = MB1 und MB2. Dieses Verhalten ist unbedingt berücksichtigen! Dies gilt auch für Datenbausteine!

Cnt UInt32

Anzahl der ushort, die geschrieben werden sollen.

Buf UInt16*

Pointer auf den ushort-Buffer.

Returns

Int32

siehe: [Result](#)

Table of Contents

Constructors	1
IPS7()	1
Methods	1
Close()	1
Connect()	1
GetConnectStatus()	2
GetCopyrightEntry(ref String)	2
GetDBLen(UInt16)	2
GetDBList()	3
GetLastPackets()	3
GetLocationDesignation(ref String)	3
GetMMCSNr(ref String)	4
GetModuleName(ref String)	4
GetModuleSNr(ref String)	4
GetModuleTypeName(ref String)	5
GetOEMId(ref String)	5
GetPlantIdName(ref String)	5
GetPLCName(ref String)	6
GetPLCTime(ref DateTime)	6
GetSockErr()	7
Open(String, UInt32, UInt32, UInt32, UInt32, UInt32)	7
OpenEx(String, UInt32, UInt32, UInt32, UInt32, UInt32, UInt32, UInt32, UInt32, Byte[], Byte[])	8
OpenPG(String, UInt32, UInt32, UInt32, UInt32, UInt32)	10
OpenS7200(String, UInt32, UInt32, UInt32, UInt32, UInt32)	10
Rd(Char, UInt32, UInt32, ref Double)	11
Rd(Char, UInt32, UInt32, ref Int16)	12
Rd(Char, UInt32, UInt32, ref Int32)	13
Rd(Char, UInt32, UInt32, ref UInt16)	13
Rd(Char, UInt32, UInt32, ref UInt32)	14
Rd(Char, UInt32, UInt32, UInt32, Byte[])	15
Rd(Char, UInt32, UInt32, UInt32, Char[])	15
Rd(Char, UInt32, UInt32, UInt32, Double[])	16
Rd(Char, UInt32, UInt32, UInt32, Int16[])	16
Rd(Char, UInt32, UInt32, UInt32, Int32[])	17
Rd(Char, UInt32, UInt32, UInt32, ref String)	18
Rd(Char, UInt32, UInt32, UInt32, UInt16[])	19
Rd(Char, UInt32, UInt32, UInt32, UInt32[])	19
RdB(Char, UInt32, UInt32, UInt32, Byte*)	20
RdBit(Char, UInt32, UInt32, UInt32, ref Byte)	21
RdBit(Char, UInt32, UInt32, UInt32, ref UInt32)	21
RdL(Char, UInt32, UInt32, ref Double)	22
RdL(Char, UInt32, UInt32, ref Int64)	22
RdL(Char, UInt32, UInt32, ref UInt64)	23
RdL(Char, UInt32, UInt32, UInt32, Double[])	24
RdL(Char, UInt32, UInt32, UInt32, Int64[])	24
RdL(Char, UInt32, UInt32, UInt32, UInt64[])	25
RdMulti(IPS7RdMulti[], UInt32)	26
RdMultiBuffered(IPS7RdMulti[], UInt32)	27
RdMultiCalcPacketCnt(IPS7RdMulti[], UInt32)	28
RdW(Char, UInt32, UInt32, UInt32, UInt16*)	28