IPS7 Class

Namespace: IPS7Lnk
Assemblies: IPS7LnkNet.Advanced.dll

Die Klasse IPS7 - enthält die Funktion für den Zugriff auf die Simatic S7. Unterstütz werden:
S7-200/300/400/1200 über TCP/IP mit CP-243-1/343-1/443-1 Lean-CP, ProfiNet oder S7-LAN-Modul

C#

```
[CLSCompliant(false)]
public class IPS7
```

Inheritance Object › IPS7

Attributes CLSCompliantAttribute

Constructors

Name	Description
IPS7	Referenznummer auf die Verbindung in der IPS7Lnk.dll.

Methods

Name	Description
Close	Dient zur Deinitialisierung der Verbindung, Speicher wird freigegeben und die TCP/IP-Verbindung wird getrennt.
Connect	Führt einen Explizitenverbindungsaufbau zur SPS aus.
GetConnectStatus	Prüft den TCP/IP Verbindungsstatus zur SPS
GetCopyrightEntry(String@)	Auslesen des Copyrighteintrags in der SPS in einen String
GetDBLen(UInt16)	Ermitteln der DB Länge
GetDBList	
GetLastPackets	Liefert die bis zu 20 letzen Telegramme als ByteArray Das Array ist 3-dimenisonal [x][y][z] x = 0: Sendetelegramm x = 1: Antworttelegramm y = 0 .. 19: Index auf die entprechende Telegrammnummer z = 0 .. n: die einzelnen Bytes des Telegramms
GetLocationDesignation(String@)	Auslesen Beschreibung für den Ort in der SPS in einen String
GetMMCSNr(String@)	Auslesen des MMC-Seriennummer in einen String
GetModuleName(String@)	Auslesen Baugruppen Name der SPS in einen String
GetModuleSNr(String@)	Auslesen der Seriennummer der SPS in einen String
GetModuleTypeName(String@)	Auslesen des CPU-Typs / Baugruppentyps in einen String
GetOEMId(String@)	Auslesen der OEM-ID in der SPS in einen String, falls vorhanden
GetPlantIdName(String@)	Auslesen des Anlagennamen in der SPS in einen String
GetPLCName(String@)	Auslesen Name der SPS in einen String
GetPLCTime(DateTime@)	Auslesen der SPS-Uhrzeit in einen DateTime

Name	Description
GetSockErr	Liefert den letzen Socket-Fehler, der bei der TCP/IP-Kommunikation aufgetreten ist.
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!
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 seperate 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!)
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!
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!
Rd(Char, UInt32, UInt32, Double@)	Liest einen einzelnen Real-Wert aus der SPS.
Rd(Char, UInt32, UInt32, Int16@)	Liest einen einzelnen short-Wert (16 Bit) aus der SPS.
Rd(Char, UInt32, UInt32, Int32@)	Liest einen einzelnen int-Wert (32 Bit) aus der SPS.
Rd(Char, UInt32, UInt32, UInt16@)	Liest einen einzelnen ushort-Wert (16 Bit) aus der SPS.
Rd(Char, UInt32, UInt32, UInt32@)	Liest eines einzelnen uint-Wertes (16 Bit) aus der SPS.
Rd(Char, UInt32, UInt32, UInt32, Byte)	Liest byteweise Daten aus der SPS in ein byte-Array.
Rd(Char, UInt32, UInt32, UInt32, Char)	Liest byteweise Daten aus der SPS in ein char-Array.
Rd(Char, UInt32, UInt32, UInt32, Double)	Liest ein Real-Array Daten aus der SPS in ein double-Array.
Rd(Char, UInt32, UInt32, UInt32, Int16)	Liest wortweise Daten aus der SPS in ein short (16-Bit)-Array.
Rd(Char, UInt32, UInt32, UInt32, Int32)	Liest doppelwortweise Daten aus der SPS in ein int(32-Bit)-Array.
Rd(Char, UInt32, UInt32, UInt32, String@)	Liest einen String aus der SPS
Rd(Char, UInt32, UInt32, UInt32, UInt16)	Liest wortweise Daten aus der SPS in ein ushort (16-Bit)-Array.
Rd(Char, UInt32, UInt32, UInt32, UInt32)	Liest doppelwortweise Daten aus der SPS in ein uint(32-Bit)-Array.
RdB(Char, UInt32, UInt32, UInt32, Byte*)	Liest einen Anzahl bytes aus der SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!
RdBit(Char, UInt32, UInt32, UInt32, Byte@)	Liest ein einzelnes Bit aus der SPS in eine byte Variable ein
RdBit(Char, UInt32, UInt32, UInt32, UInt32@)	Liest ein einzelnes Bit aus der SPS in eine uint Variable ein
RdL(Char, UInt32, UInt32, Double@)	Liest einen einzelnen LReal-Wert aus der SPS.
RdL(Char, UInt32, UInt32, Int64@)	Liest einen einzelnen LINT-Wert (64 Bit) aus der SPS.
RdL(Char, UInt32, UInt32, UInt64@)	Liest eines einzelnen ULINT-Wertes (46 Bit) aus der SPS.
RdL(Char, UInt32, UInt32, UInt32, Double)	Liest ein LReal-Array Daten aus der SPS in ein double-Array.
RdL(Char, UInt32, UInt32, UInt32, Int64)	Liest aus der SPS in ein LINT(64-Bit)-Array.
RdL(Char, UInt32, UInt32, UInt32, UInt64)	Liest aus der SPS in ein ULINT(64-Bit)-Array.

Name	Description
RdMulti(IPS7RdMulti, UInt32)	ACHTUNG! Dieser Aufruf darf nur verwendet werden, wenen Ihre Applikation im Unsafed-Mode läuft und der Zielspeicher während des Aufrufs gefixed ist! Grund: Der Garbage Collector könnte während des Aufrufs die Daten verscheiben. 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 Type 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 Zusatnd 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 auf gerufen. Diese setzte im Request-Eintrag den PC-Datentype auf INT32. Somit stehten 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 wen 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 Konsitenz der Daten z.B. bei 16/32 Bit werten geachtet.

Name	Description
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 verscheiben. 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 Zusatnd 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 auf gerufen. Diese setzte im Request-Eintrag den PC-Datentype auf INT32. Somit stehten 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 wen 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 Konsitenz der Daten z.B. bei 16/32 Bit werten geachtet.
RdMultiCalcPacketCnt(IPS7RdMulti, UInt32)	Berechnet die Anzahl der Pakete, die für diesen gemischten Readauftrag nötig sind. Zuvor ist ein Array vom Type IPS7RdMulti zu initialisieren.
RdW(Char, UInt32, UInt32, UInt32, UInt16*)	Liest einen Anzahl ushort aus der SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!
ResetBit(Char, UInt32, UInt32, UInt32)	Setzt ein einzelnes Bit in der SPS zurück.
SetBit(Char, UInt32, UInt32, UInt32)	Setzt ein einzelnes Bit in der SPS.
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 festfestellt, 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.
SetPLCTime(DateTime)	Setzen der SPS-Uhrzeit mit einen DateTime

Name	Description
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 where processed and responded. Bye default this Bug is not solved (bDoSolved = false), normaly S7 works well. (since 1.1.77.12)
Wr(Char, UInt32, UInt32, Byte)	Schreibt einen einzelnen byte-Wert (8-Bit) in die SPS.
Wr(Char, UInt32, UInt32, Double)	Schreibt einen einzelnen double-Wert (Real) in die SPS.
Wr(Char, UInt32, UInt32, Int16)	Schreibt einen einzelnen short-Wert (16-Bit) in die SPS.
Wr(Char, UInt32, UInt32, Int32)	Schreibt einen einzelnen int-Wert (32-Bit) in die SPS.
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
Wr(Char, UInt32, UInt32, UInt16)	Schreibt einen einzelnen ushort-Wert (16-Bit) in die SPS.
Wr(Char, UInt32, UInt32, UInt32)	Schreibt einen einzelnen uint-Wert (32-Bit) in die SPS.
Wr(Char, UInt32, UInt32, UInt32, Byte)	Schreibt byteweise ein byte-Array in die SPS.
Wr(Char, UInt32, UInt32, UInt32, Char)	Schreibt byteweise ein char-Array in die SPS.
Wr(Char, UInt32, UInt32, UInt32, Double)	Schreibt ein double-Array (Real-Array) in die SPS.
Wr(Char, UInt32, UInt32, UInt32, Int16)	Schreibt wortweise ein short-Array (16-Bit) in die SPS.
Wr(Char, UInt32, UInt32, UInt32, Int32)	Schreibt doppelwortweise ein int-Array (16-Bit) in die SPS.
Wr(Char, UInt32, UInt32, UInt32, UInt16)	Schreibt wortweise ein ushort-Array (16-Bit) in die SPS.
Wr(Char, UInt32, UInt32, UInt32, UInt32)	Schreibt doppelwortweise ein uint-Array (32-Bit) in die SPS.
WrB(Char, UInt32, UInt32, UInt32, Byte*)	Schreibt einen Anzahl bytes in die SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!
WrBit(Char, UInt32, UInt32, UInt32, UInt32)	Schreibt ein einzelnes Bit in die SPS.
WrL(Char, UInt32, UInt32, Double)	Schreibt einen einzelnen LReal (double) in die SPS.
WrL(Char, UInt32, UInt32, Int64)	Schreibt einen einzelnen LINT-Wert (64-Bit) in die SPS.
WrL(Char, UInt32, UInt32, UInt32, Double)	Schreibt ein LReal-Array (double-Array) in die SPS.
WrL(Char, UInt32, UInt32, UInt32, Int64)	Schreibtein LINTint-Array (64-Bit) in die SPS.
WrL(Char, UInt32, UInt32, UInt32, UInt64)	Schreibt ein ULINT-Array (64-Bit) in die SPS.
WrL(Char, UInt32, UInt32, UInt64)	Schreibt einen einzelnen ULINT-Wert (64-Bit) in die SPS.
WrMulti(IPS7WrMulti, UInt32)	
WrW(Char, UInt32, UInt32, UInt32, UInt16*)	Schreibt einen Anzahl ushorts (16-Bit) in die SPS. Diese Funktion ist nur im unsafe-Mode zu verwenden!

Table of Contents