Beispieldokumentation

Deutsche Beschreibung

NUTZUNGSBEDINGUNGEN

Die Verwendung der Beispielprogramme erfolgt ausschließlich unter Anerkennung folgender Bedingungen durch den Benutzer:

INSEVIS bietet kostenlose Beispielprogramme für die optimale Nutzung der S7-Programmierung und zur Zeitersparnis bei der Programmerstellung. Für direkte, indirekte oder Folgeschäden des Gebrauchs dieser Software schließt INSEVIS jegliche Gewährleistung genauso aus, wie die Haftung für alle Schäden, die aus die aus der Weitergabe der die Beispielinformationen beinhaltenden Software resultieren.

BEISPIELBESCHREIBUNG

Inhalt

Dieses Beispiel erklärt die Grundlagen einer Modbus-TCP-Anbindung.

Modbus Historie

Modbus – als ModbusRTU – war ursprünglich definiert worden, um dezentrale Peripherie über eine Serielle Schnittstelle anzubinden.
Als Datentypen wurden Digitale Eingänge ('Input Bits'), Digitale Ausgänge ('Coils' – man kannte damals nur Magnetspulen als Aktoren), Analoge Eingänge („Input Register“) mit 16 Bit und Analoge Ausgänge („Holding Register“) mit 16 Bit definiert. Zu jedem Datentyp gibt es spezifische Kommandos (function codes).

01	(eine oder mehrere) Ausgangsbits (Coils) zurücklesen
02	(ein oder mehrere) Input Bits lesen
03	(ein oder mehrere) Holding Register lesen
04	(ein oder mehrere) Input Register lesen
05	ein Ausgangsbit schreiben
06	Holding Register schreiben
$0F_{hex}$	mehrere Ausgangsbits schreiben
10_{hex}	mehrere Holding Register schreiben

Es müssen nicht alle Kommandos (function codes) unterstützt werden.
Da die HoldingRegister rücklesbar sind, könnten im Prinzip alle anderen Funktionen im Lesen und Schreiben von Holding Registern abgebildet werden. Manche Geräte nutzen das und bilden komplexe Datenstrukturen einfach in Holdingregistern ab.
Die Bits, Coils und Register werden jeweils über einen Index adressiert. Dieser beginnt bei 0 und ist 16 Bit breit.

Jede Peripheriestation wird über eine Knotennummer (Unit-Identifier UID) identifiziert.

Beispiel eines Modbus-Datenpaketes zum Lesen von Inputregistern:

1 Byte: UID
1 Byte: Kommando (function code): 04
2 Bytes: Register-Index
2 Bytes: Anzahl zu lesender Register

Modbus-TCP

Bei Modbus-TCP werden die Stationen über die IP-Adresse identifiziert und die Telegramme der Seriellen Schnittstelle unverändert in Ethernet-Datenpakete abgebildet.
Damit bleiben Beschränkungen wie z.B. die Länge der Datenpakete erhalten, aber die Vorteile der Ethernetkommunikation genutzt.
Die UID ist nun innerhalb eines TCP-Datenframes und ermöglicht nun mehrere virtuelle Modbus-Server innerhalb des schon über die IP-Adresse adressierten Gerätes. (Das wird selten genutzt, ggf. bei Modbus-TCP - ModbusRTU Gateways. Um Verwechslung auszuschließen muss er hier erwähnt werden)

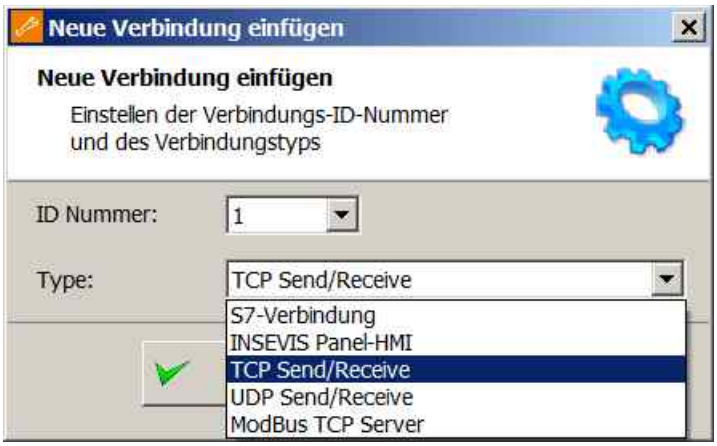
Implementierung eines Modbus-TCP Server in einer INSEVIS SPS
Im Konfigurationstool der INSEVIS-SPS „ConfigStage“ kann unter „Ethernet“ ein Modbus-TCP-Server aktiviert und konfiguriert werden. Dieser erlaubt einen externen Zugriff auf die konfigurierten S7-Operanden. Der Zugriff erfolgt eventgesteuert, auch während des Zyklus. Die Register- und Bit-Adressierung beginnt modbuskonform mit 0.
In diesem Beispiel dient dieser Server als Kommunikationspartner zum Test.

Implementierung eines Modbus-TCP Clients in einer INSEVIS SPS
Das Verhalten der Modbus-Geräte (Server, Slaves) ist genau beschrieben, der Ablauf im Master (Client) ist jedoch nicht spezifiziert. Eine feste Implementierung im Betriebssystem wäre damit nicht flexibel genug.

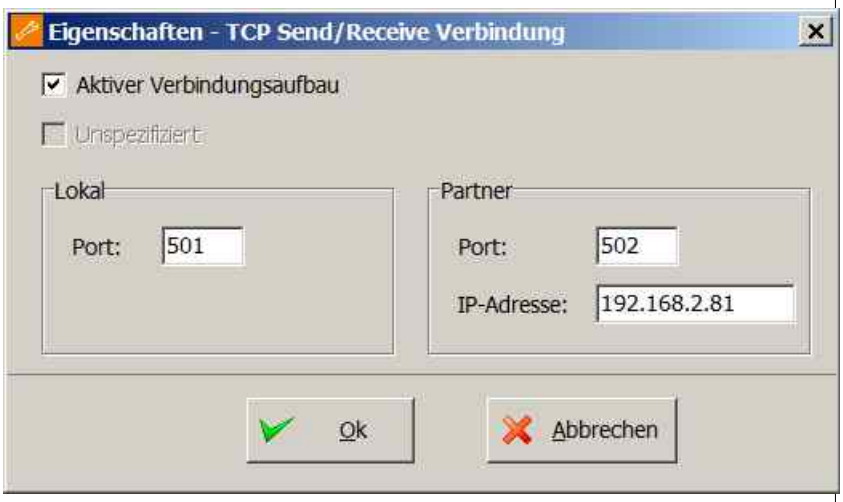
Der hier vorgestellte Modbus-TCP client wurde deshalb in S7 realisiert, um kundenspezifische Anpassungen zu ermöglichen ist er quelloffen.
.

Konfiguration
Für jeden einzubindenden Modbus-TCP-Server (Slave) muss eine TCP-Verbindung angelegt werden.
Das erfolgt bei der INSEVIS-SPS im Konfigurationstool ConfigStage.

Unter „Ethernet“ wird eine neue Verbindung vom Typ TCP Send/Receive angelegt und bearbeitet.
Die automatisch vergebene ID-Nummer wird im S7-Programm zur Zuordnung benutzt.



Die erstellte TCP-Verbindung wird mit der IP-Adresse des Kommunikations-partners konfiguriert.
Die Partner-Port-Adresse
(= die des Servers) ist bei Modbus per Standard 502.
Die lokale Portadresse kann weitgehend frei gewählt werden, sie sollte nicht 0 sein und darf nicht mit bestehenden Verbindungen kollidieren.



S7-Programm
FB1 dient als Modbus-TCP client Treiberbaustein und nutzt die Systembausteine zum Senden und Empfangen über TCP/IP und sollte **unverändert** bleiben.
Als Parameter werden VerbindungsID-Nummer, Knotennummer (UID), Modbus- Kommando (function code 1, 2, 3, 4, 6, 15 oder 16) und Nutzdatenpointer übergeben.

```
CALL  "ModbusTCP_Client" , "tcp"      // FB1 mit DB1 als Instanz-DB
  R         :=FALSE                   // Reset-Eingang, einmalig nach Hochlauf setzen
  ConnectID:=1                        //  VerbindungsID-Nummer aus ConfigStage
  UID       :=                        //  obsolete Knotennummer aus ModbusRTU
  Cmd       :=B#16#2                  // Modbus Kommando (funktion code 1,2,3,4,6,15,16)
  Index     :=0                       // Register bzw. Bit-Adresse (0...65535)
  Length    :=256          // Anzahl zu übertragender Register (1..125) bzw. Bits (1..2000)
  Data      :="Daten_TCP".Inputs      // ANY-Pointer auf Nutzdatenbereich
  Done      :="tcp1_done"             // TRUE wenn erfolgreich
  Error     :="tcp1_error"            // TRUE bei Fehler
  ErrSrc    :="tcp1_ErrorSrc"         // Fehlercode s. Tabelle
  ErrStatus:="tcp1_ErrCode"
```

Der Aufruf muss wiederholt werden, bis DONE oder ERROR zurückgegeben wird.
Die Längenangabe im Pointer DATA wird zum Kopieren der Nutzdaten benutzt und muss zu den Datenpaketen passen (Beim Senden wird sonst ggf. der Sendepuffer unvollständig gefüllt oder andere Werte überschrieben. Beim Empfang werden ggf. andere Nutzdaten überschrieben)
Alle lokalen Variablen des Kommunikationstreiberbausteins FB1 sowie Sende- und Empfangsdaten liegen in dem zugehörigen Instanzdatenbaustein.

FC1 stellt die Modbus-Kundenapplikation dar und **muss** an die jeweiligen Anforderungen **angepasst** werden. D.h. hier wird programmiert, zu welchen Teilnehmern wann mit welchen Daten kommuniziert wird.
Über einen Sprungverteiler ist eine state-machine realisiert, welche nach Initialisierung zyklisch nacheinander alle Modbus-Kommandos (function codes) anwendet. Das gibt in diesem Beispiel nicht unbedingt Sinn.

Achtung:
FC1 benutzt als state-Merker einen 0-initialisierten Merker. Dieser Merker darf nicht als remanent konfiguriert werden!

Alternative Topologien bei Kommunikation zu mehreren Servern

Mehrere Modbus-Kommunikationen zu EINEM Gerät müssen nacheinander ablaufen,
aber mehrere TCP-Verbindungen können auch parallel ablaufen.
Man kann also in einem Applikationsbaustein alle Server nacheinander bedienen oder den FC1 kopieren (den Verbindungsindex anpassen und verschiedene Instanzdatenbausteine benutzen) und parallel kommunizieren.

Wenn mehr als 16 Verbindungen erforderlich sind, muss über SFB125 „TCONFIG“ eine Verbindung dynamisch umkonfiguriert werden. Dies ist nicht im Umfang dieser Demo.

Troubleshooting
Kein Verbindungsaufbau:
Ein tückisches Problem ist, dass sich in manchen Fällen die TCP-Verbindung nicht aufbauen lässt, wenn diese zuvor hart abgebrochen wurde. (z.B. SPS aus- und einschalten und TCP-Slave weiterlaufen lassen)
Dieser Fehler verschwindet dann nach Minuten aufgrund eines TCP-Verbindungs-timeout.
Dem kann man u.U. durch gezielten Verbindungssabbau über SFB 124 abhelfen, insbesondere bei vorübergehendem Ausfall des Kommunikationspartners. Wenn die SPS „hart“ ausgeschaltet wird, muss der Timeout abgewartet werden, wenn die Gegenseite nicht irgendwie zurückgesetzt werden kann.

Mit Hilfe eines managebaren Switches (z.B. Netgear GS105E) kann über „Port mirroring“ der Datenverkehr zwischen 2 physikatlischen RJ45-Ports auf einen 3. kopiert werden. Hier kann man dann z.B. mit einem PC und dem open source Programm „wire shark“ den gesamten Ethernetdatenverkehr abhören. Ein Filter auf die passende IP-Adresse und Port 502 kann nützlich sein.

Fehlercodes
Die Rückgabewerte des FB1 sind in eine Fehlerquelle (ErrSrc) und einen StatusCode aufgeteilt.
ErrSrc entspricht zum state der statemachine in FB1, in dem der Fehler aufgetreten ist. Davon abhängig sind die jeweiligen Fehlercodes:

ErrSrc	ErrStatus	Bedeutung
0,1	Rückgabewerte des SFB 124 TDISCON:	
	8001_{hex}	Parameter ID ist nicht korrekt
	8002_{hex}	Verbindung mit ID ist nicht konfiguriert oder inkorrekter Verbindungstyp.
4,5	Rückgabewerte des SFB 122 TSEND:	
	8001_{hex}	Parameter-ID ist nicht korrekt.
	8002_{hex}	Verbindung mit ID ist nicht konfiguriert oder inkorrekter Verbindungstyp.
	8003_{hex}	Parameter DATA ist nicht korrekt. Nur E, A, M, DB Bereiche erlaubt
	8004_{hex}	Parameter DATA ist nicht korrekt. z.B. DB nicht geladen
	8005_{hex}	Parameter LEN ist 0 oder größer als angegeben unter Parameter DATA
	8006_{hex}	Keine Verbindung zu Partner aufgebaut
	8007_{hex}	Auftrag fehlgeschlagen wegen Verbindungsproblem (Kabel abgezogen, Kommunikation durch Partner zuruckgewiesen).
3	Syntax check Parameter	
	8001_{hex}	reserviert für UID > 127, Prüfung entfernt, weil manche Applikationen “FF” benutzen
	8002_{hex}	Ungültiger Modbus-CMD (function code)
	8003_{hex}	Ungültige Längenangabe (Register > 250, Bits/Coils > 2000)
	8004_{hex}	ungültiger Bereich Nutzdaten (≠ E, A, M, DB)
	sonst	Fehlercode des SFB20
6	Rückgabewerte des SFB 123 TRECV:	
	8001_{hex}	Parameter ID ist nicht korrekt.
	8002_{hex}	Verbindung mit ID ist nicht konfiguriert oder inkorrekter Verbindungstyp.
	8003_{hex}, 8004_{hex}	Parameter DATA ist nicht korrekt.
	8006_{hex}	Keine Verbindung zu Partner aufgebaut
	8007_{hex}	Auftrag fehlgeschlagen wegen Verbindungsproblem (Kabel abgezogen, Kommunikation durch Partner zuruckgewiesen).
7	Syntax check Empfangsdaten	
	9000_{hex}	Ungültige Antwort, Request zurückgewiesen
	$8xxxx_{hex}$	Returncode des SFC20
*	$CAFE_{hex}$	Timeout

Querverweise:
Der Modbus-Server kann auch gleichzeitig mit dem Client betrieben werden, welches eine „Relay“-Funktion ermöglicht; d.h. Daten per Modbus-TCP client einsammeln, vorverarbeiten und als Modbus-TCP Server publizieren. Dafür gibt es ein separates Demoprojekt.

RÜCKMELDUNGEN
Möchten Sie Erweiterungswünsche oder Fehler zu diesen Beispielen melden oder wollen Sie anderen eigene Bespielprogramme kostenlos zur Verfügung stellen? ***Bitte informieren Sie uns unter info@insevis.de***
Gern werden Ihre Programme -auf Wunsch mit Benennung des Autors- allen INSEVIS- Kunden zur Verfügung gestellt.

English description

TERMS OF USE

The use of this sample programs is allowed only under acceptance of following conditions by the user:
The present software which is for guidance only aims at providing customers with sampling information regarding their S7-programs in order to save time. As a result, INSEVIS shall not be held liable for any direct, indirect or consequential damages respect to any claims arising from the content of such software and/or the use made by customers of this sampling information contained herin in connection with their own programs.

SAMPLE DESCRIPTION

Abstract
This example explains the basics of a Modbus TCP client and server interface.

Modbus History
Modbus – former as ModbusRTU – was primary defined to connect decentral periphery via a serial link.
As data typa Digital Inputs (called 'Input Bits'), Digitale Outputs (called 'Coils'), Analog Inputs („Input Register“) with 16 Bit and Analog Outputs („Holding Register“) with 16 Bit were defined.
For every data type specific commands (function codes) are defined:

01	read (one or several) outputbits (Coils)
02	read (one or several) input bits
03	read (one or several) Holding Register
04	read (one or several) Input Register
05	set a output bit (Coil)
06	write a holding registers
$0F_{hex}$	write several outputbits
10_{hex}	write several Holding Register

Not all commands (function codes) must be supported.
Due to HoldingRegister can read back, in theory all other functions could be done by reading and writing of holding registers. Some devices utilise this mapping complex data structures into holding registers.
Bits, coils and register are addressed by an index (0 ... 65535).

Every station has an unique node number (Unit-Identifier UID).

Example of a Modbus-data packet to Read input register:

1 Byte: UID
1 Byte: command (function code): 04
2 Bytes: Register-Index
2 Bytes: number of registers to read

Modbus-TCP
Modbus-TCP identifies the units via IP-address and sends the former serial data packes via Ethernet.
This way some limitations are kept e.g. length of data packts, but advantages of Ethernetcommunication are used.
The UID now is inside of a TCP-data frame and allows several virtual Modbus-Server inside the device already address by an IP-address. (e.g. used by Modbus-TCP - ModbusRTU Gateways. Keep it in mind to understand the next)

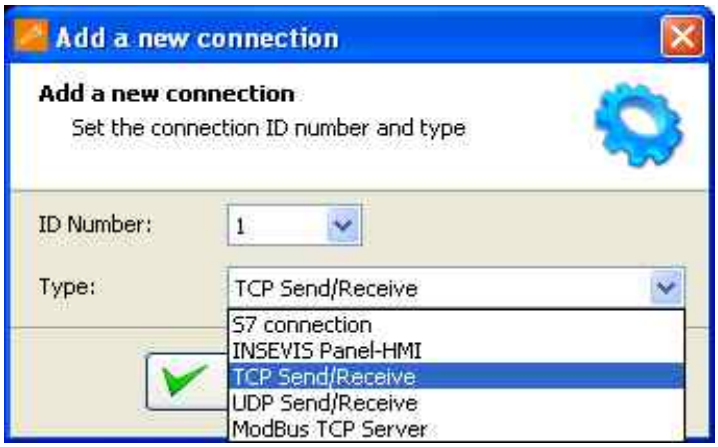
Implementation of a Modbus-TCP Server in an INSEVIS SPS
The configurationtool of INSEVIS „ConfigStage“ allows under „Ethernet“ to enable and configure a Modbus-TCP-Server. This server realizes external access to the configured S7-Operands. The access works event driven, also during OB1 cycle. The register- and bit-addressing starts at 0.
In this example the Modbus-TCP-server is used as communication partner for test.

Implementation of a Modbus-TCP Client in an INSEVIS SPS
The behavior of Modbus-devices (Server, Slaves) is specified exactly, but the master (Client) is a "matter of application". A fixed implementation in the operating system would be not flexible.

That's why the introduced Modbus-TCP client was written in S7. To allow customer adaptions it is open source.

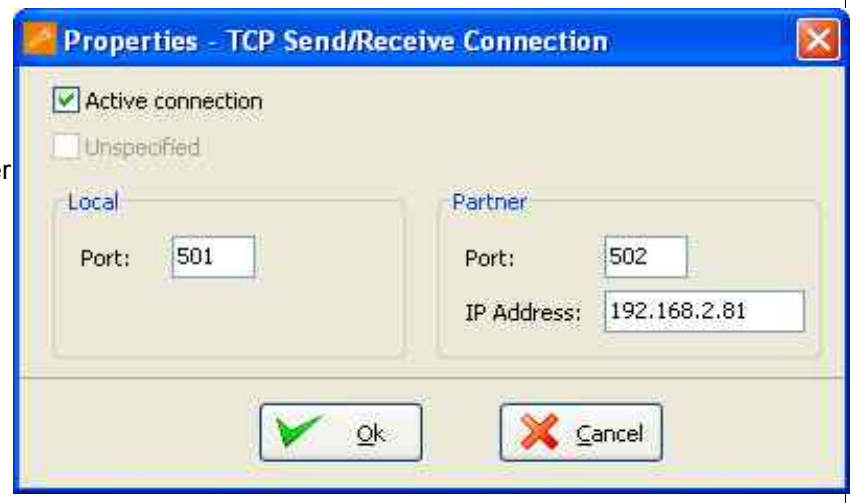
Configuration
For every Modbus-TCP-Server (slave) an own TCP-connection must be generated.
This is done in the INSEVIS-PLC configuration tool ConfigStage.

Choose „Ethernet“ and create a new connection of type TCP Send/Receive
The automatic assigned ID-number will be used in the S7-program.



Configure the TCP connection with the IP-address of the communication partner

The partners port-address (= address of the server) is always Modbus default 502.
The local port address could be freely chosen. It should be not 0 and must not conflict to other existing connections.



S7-Program
FB1 works as Modbus-TCP client driver and handles the system calls for send und receive via TCP/IP and is **no to change**.
As parameter it uses connection ID-number, node-number (UID), Modbus-command (function code 1, 2, 3, 4, 6, 15 or 16) and payload data pointer.

```
CALL  "ModbusTCP_Client" , "tcp"      // FB1 + DB1 as Instance-DB
  R         :=FALSE                   // Reset-input, to set once while startup
  ConnectID:=1                        //  connection ID-number from ConfigStage
  UID       :=                        //  obsolete node number from ModbusRTU
  Cmd       :=B#16#2                  // Modbus command (funktion code 1,2,3,4,6,15,16)
  Index     :=0                       // Register rsp. Bit-address (0...65535)
  Length    :=256          // number of register (1..125) or bits (1..2000) to transfer
  Data      :="Daten_TCP".Inputs      // ANY-Pointer payload data area
  Done      :="tcp1_done"             // TRUE if well done
  Error     :="tcp1_error"            // TRUE in case of trouble
  ErrSrc    :="tcp1_ErrorSrc"         // errorcode s. table
  ErrStatus:="tcp1_ErrCode"
```

The call of FB1 must repeated until DONE or ERROR are returned.
The length information of Pointer DATA will be used to copy data and must match to the used data packets. (Otherwise sending the buffer could be filled incomplete or other data are overwritten.)
All local data are kept in the instance data block

FC1 represents the Modbus customers application and **must be adapted** i. e. here will be defined when and how to communicate.
A state-machine realizes the communication steps. After an initialization in this example all Modbus-commands (function codes) are called once and then cyclically. This is an example and makes not really sense.

Caution::
FC1 uses as state-variable a 0-initialized memory. Do not reconfigure this variable as remanent!

Alternative topologies communicating to several servers

Several Modbus-communications to ONE device must run consecutively. But several TCP connections can run simultaneous.
To handle several Modbus-TCP servers it is possible extend the state machine and communicate one after another. Adapting a copy of FC1 to separated global variables and calling FB1 with an own instance data block and matching connection ID allows simultaneous communication with better performance and more independence.
If more then 16 connections are necessary, a multiplexing can be realized by dynamically reconfiguring connections with SFB125 „TCONFIG“This is not matter of this demo.

Troubleshooting
No connection:
A malicious problem occurs when the connection breaks hardly (e. g. loss of power at PLC or TCP device). The counterpart keeps the connection and denies re-enabling the same connection. This situation disappears after some minutes due to TCP connection timeout. Active disconnection with SFB 124 will help in some cases.
Tracing:
Using a manageable switch (e. g. Netgear GS105E) „port mirroring“ copies the data traffic between 2 physical RJ45-ports to a third port. Now a PC with the open source program „wire shark“ can log the whole ethernet traffic. Setting a Filter regarding IP-addresses and Port 502 is recommended.

Errorcodes
Return values of FB1 are divided into error source (ErrSrc) and a status code (ErrStatus).
Error source accords with the last state of the state machine in FB1, as the error occurred. Related to the error source the status code contains information about the cause of error:

ErrSrc	ErrStatus	description
0,1	Status return of SFB 124 TDISCON:	
	8001_{hex}	parameter ID incorrect
	8002_{hex}	Verbindung mit ID ist nicht konfiguriert oder inkorrekter Verbindungstyp.
4,5	Status return of SFB 122 TSEND:	
	8001_{hex}	parameter ID incorrect
	8002_{hex}	Connection with specified ID is not configured or of incorrect connection type
	8003_{hex}	parameter DATA incorrect, only I, O, M or DB areas allowed
	8004_{hex}	parameter DATA incorrect e.g. DB not available
	8005_{hex}	parameter LEN is 0 or bigger than specified in parameter DATA
	8006_{hex}	no connection to partner established
	8007_{hex}	Job failed due to connection problem (e. g. cable unplugged,communication denied by partner)
3	Syntax check parameter	
	8001_{hex}	reserved for UID > 127, check removed, because some applications use “FF”
	8002_{hex}	invalid CMD
	8003_{hex}	invalid len (register > 250, bits/coils > 2000)
	8004_{hex}	invalid area of payload data (≠ I, O, M, DB)
	else	errorcode of SFB20 (block move)
6	Status return of SFB 123 TRECV:	
	8001_{hex}	parameter ID incorrect.
	8002_{hex}	Connection with specified ID is not configured or of incorrect connection type.
	8003_{hex}, 8004_{hex}	parameter DATA incorrect
	8006_{hex}	no connection to partner established
	8007_{hex}	Job failed due to connection problem (e. g. cable unplugged,communication denied by partner)
7	Syntax check receive data	
	9000_{hex}	Invalid response or request denied
	$8xxxx_{hex}$	Status return of SFC20
*	$CAFE_{hex}$	Timeout

cross reference:
The integrated Modbus-TCP-Server can run simultaneous with these client application. This allows a „Relay“-funktion; i.e. collect daten via Modbus-TCP client, manipulate them and publish as Modbus-TCP Server. This is treated in a separate demo projekt.

FEEDBACK
Do you want to inform us about necessary increments or errors or do you want to provide us with your sample programs to offer it for free to all customers? ***Please inform us at info@insevis.de***
Gladly we would provide your program -if you wish with the authors name- to all other customers of INSEVIS.