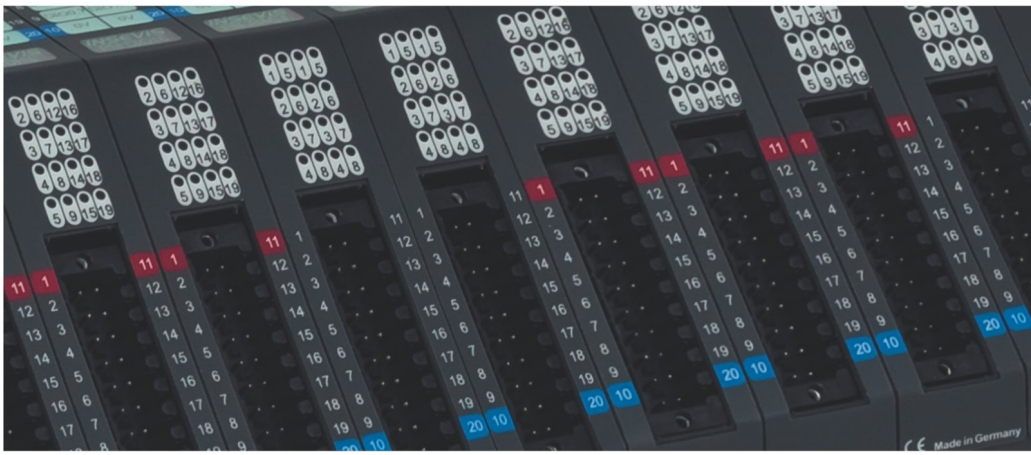
INSEVIS



Beispieldokumentation
Sample documentation

Modbus RTU Client und Server

Modbus RTU client and server

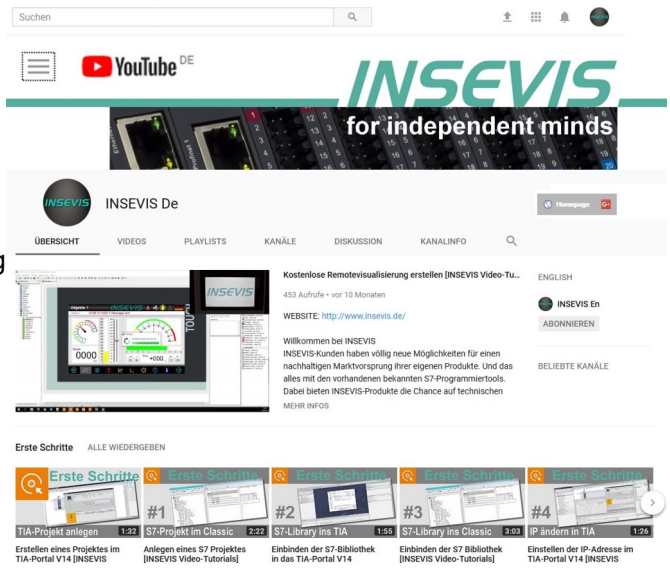
Hinweis zum besseren Verständnis durch Zusatzinformationen

Im deutschen INSEVIS-YouTube-Kanal INSEVIS DE stehen mehrere Playlists mit **Hantierungsvideos** für einzelne Details zur Verfügung.

Ebenfalls stehen **Handbücher** für die einzelnen Produktgruppen im Downloadbereich der Webseite insevis.de zur Verfügung

Bitte nutzen Sie diese Informationsquellen in Ergänzung zur vorliegenden Dokumentation. So können Sie sich noch leichter mit den INSEVIS-Funktionen vertraut machen.

Möchten Sie Erweiterungswünsche oder Fehler zu diesen Beispielen melden oder wollen Sie anderen eigene Bespielprogramme kostenlos zur Verfügung stellen? Gern werden Ihre Programme -auf Wunsch mit Benennung des Autors- allen INSEVIS- Kunden zur Verfügung gestellt.



Hinweis zu den verschiedenen Versionen der Beispielprogramme

Im Lieferumfang der Beispielprogramme können sich auch ältere Ausgabestände bzw. Versionen befinden. Diese wurden nicht aktualisiert und auf die neueste Siemens-Programmiersoftware angepasst, um einen Zugriff mit älteren Programmier-systemen weiterhin zu ermöglichen. Generell werden INSEVIS-Beispielprogramme immer mit dem aktuell neuesten Siemens-Programmiertools erstellt.

BEISPIELBESCHREIBUNG

Inhalt

Dieses Beispiel erklärt die Grundlagen einer Modbus-RTU-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') und Ausgänge ('Coils' – man kannte damals nur Magnetspulen als Aktoren), sowie Analoge 16 Bit-Eingänge („Input Register“) und Ausgänge („Holding Register“) 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

Konfiguration

Client:

Schnittstelle RS485:

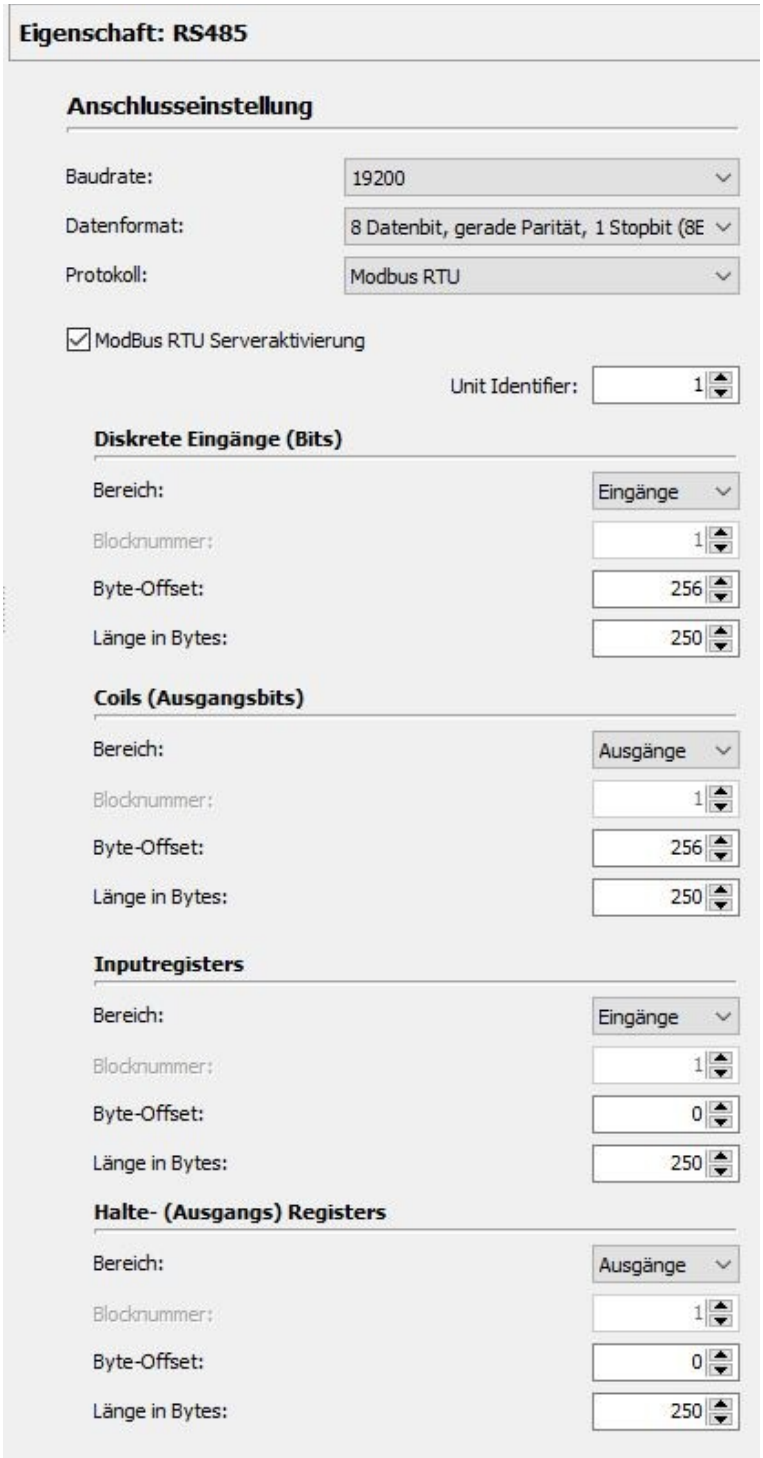
- Baudrate auf 19200 (maximal)
- Datenformat 8E1 (empfohlen)
- Protokoll Modbus RTU
- Serveraktivierung AUS

Server:

Schnittstelle RS485:

- Baudrate auf 19200 (maximal)
- Datenformat 8E1 (empfohlen)
- Protokoll Modbus RTU
- Serveraktivierung EIN
- UID 1
- Konfiguration der Inputs, Coils, Input Register und Holding Register

  ACHTUNG: Bei Serveraktivierung sind KEINE weiteren S7-/TIA-Programmierungen vorzunehmen Nur die Clientfunktion wird mit S7-/TIA programmiert.



S7-Programm
Das Betriebssystem stellt Sende- und Empfangsfunktionen zur Modbus-RTU-Kommunikation bereit. Diese beinhalten die Rahmensynchronisation nach Modbus-Spezifikaion einschließlich Checksummenberechnung. Als Interface zur Applikation dient FB2:

```
CALL  "ModbusRTUClient" , "RTU_Instance"    // FB2 mit DB2 als Instanz-DB
  R         :=FALSE                         // Reset-Eingang, einmalig nach Hochlauf setzen
  UID       :=B#16#1                        // Geräteadresse
  Cmd       :=B#16#2                        // Modbus Kommando (Funktionscode 1,2,3,4,6,15,16)
  Index     :=0                             // Register bzw. Bit-Adresse (0...65535)
  Length    :=2000                 // Anzahl zu übertragender Register (1..125) bzw. Bits (1..2000)
  Data      :="Daten_RTU".Inputs            // ANY-Pointer auf Nutzdatenbereich
  Done      :="RTU_done"                    // TRUE wenn erfolgreich
  Error     :="RTU_error"                   // TRUE bei Fehler
  ErrSrc    :="RTU_ErrorSrc"                // Fehlercode s. Tabelle
  ErrStatus:="RTU_ErrCode"
```

Daraus werden in FB2 die Telegramme erstellt, gesendet, empfangen und die Nettodaten kopiert. Für Modbus-Header und Checksumme besteht ein eigener Puffer im Instanzdatenbaustein. Es erfolgt ebenfalls eine Timeout-Überwachung. Erfolgt keine Antwort innerhalb 1 Sekunde, wird der Empfang abgebrochen und das Errorflag gesetzt. Werden CRC-Fehler erkannt oder wird ein RTU-Request vom Server zurückgewiesen, wird ebenfalls das Errorflag gesetzt, der zurückgelesene Funktionscode wird überpfüft, weitere Protokollfehler werden nicht beachtet.

FC2 realisiert als Test-Beispiel eine Modus-RTU-Client Anwendung. Beispielhaft werden zyklisch alle implementierten Funktionen aufgerufen. Dieser Ablauf muss an die Anforderung der Anwendung angepasst werden. Am Ende des FC2 erfolgt eine Fehlerauswertung. Diese muss ebenfalls an die Gegebenheiten der Anwendung angepasst werden.

Fehlercodes
Die Rückgabewerte des FB2 sind in eine Fehlerquelle (ErrSrc) und einen StatusCode (ErrStatus) aufgeteilt. ErrSrc entspricht dem state der statemachine in FB2, in dem der Fehler aufgetreten ist. Davon abhängig sind die jeweiligen Fehlercodes:

ErrSrc	ErrStatus	Bedeutung
0	8001_{hex}	RS485 nicht in Betriebsart ModbusRTU
	sonst Rückgabewerte des SFB 61 bei Initialisierung	
1	8001_{hex}	UID > 127
	8002_{hex}	Ungültiges Modbus-CMD (function code)
	8003_{hex}	Ungültige Längenangabe (Register > 250, Bits/Coils > 2000)
	8004_{hex}	ungültiger Bereich Nutzdaten (≠ E, A, M, DB)
2	8000_{hex}	SFB 60 belegt (Sendepuffer overflow)
	Rückgabewert SFB20 oder SFB 60 (Senden)	
3	Rückgabewert SFB 61 bei Empfang	
	$CAFE_{hex}$	Timeout
4	9001_{hex}	Bit 7 im Empfangswert des Funktionscodes gesetzt = Server weist Anfrage zurück (ungültige Parameter)
	9000_{hex}	sonstiger Fehler im Empfangswert des Funktionscodes
	Rückgabewert des SFC 20 beim Kopieren der Nutzdaten	

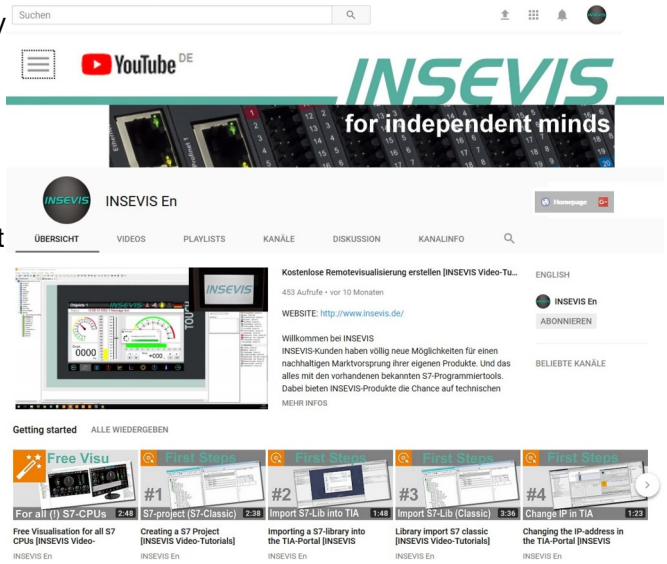
Modbus RTU Observer:
Falls kein PC mit einer RS485-Schnittstelle verfügbar ist, kann im Problemfall eine weitere SPS als Protokoll-beobachter eingesetzt werden. Alle empfangenen Bytes werden mit einem Zeitstempel in einen Ringpuffer geschrieben. (Das Zeitraster ist mit 10 ms zu grob, um einzelne Telegramme zu trennen, aber die Anfragen des Client können zeitlich beliebig verzögert werden. Die Hardwarekonfiguration über ConfigStage muss als plain ASCII erfolgen.

Vorgehensweise

- Laden der Konfiguration für Modbus-RTU-Server in Server SPS → RUN
- Laden der S7 Software in Client SPS
- Laden der Konfiguration für Modbus-RTU-Client in Client SPS → RUN
- ggf. Laden von S7 Software und Konfiguration für Modbus-RTU-Observer in Observer SPS → RUN
- Beobachten/Steuern über Variablentabellen mit folgenden Möglichkeiten:
    1. Auslesen aller oder bestimmter Coils (Funktion 01hex)
    2. Auslesen aller oder bestimmter Inputs (Funktion 02hex)
    3. Auslesen aller oder bestimmter Holding Register (Funktion 03hex)
    4. Auslesen aller oder bestimmter Input Register (Funktion 04hex)
    5. Schreiben einzelner Coils (Funktion 05hex)
    6. Schreiben einzelner Holding Register (Funktion 06hex)
    7. Schreiben mehrerer/aller Coils (Funktion 0Fhex)
    8. Schreiben mehrerer/aller Holding Register (Funktion 10hex)

Hint for better understanding by additional information

In the English YouTube-channel INSEVIS EN we supply different playlists with handling videos for single details. This will help you to get familiar with INSEVIS much faster.



Please download the referring manual from the download area of our English website insevis.com to get familiar with INSEVIS technology in detail.

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? Gladly we would provide your program -if you wish with the authors name- to all other customers of INSEVIS.

Hint to different versions of the sample programs

There could be older versions in delivery scope of the sample programs too. These were not updated and converted to the newest programming tool versions to allow access by older programming tools too. INSEVIS sample programs will be created in the present newest Siemens-programming tool always.

SAMPLE DESCRIPTION

Abstract

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

Modbus History

Modbus – former as ModbusRTU – was primary defined to connect decentral periphery via a serial link.
As data types 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 each 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 an 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).

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

Example of a Modbus data packet to read input register:

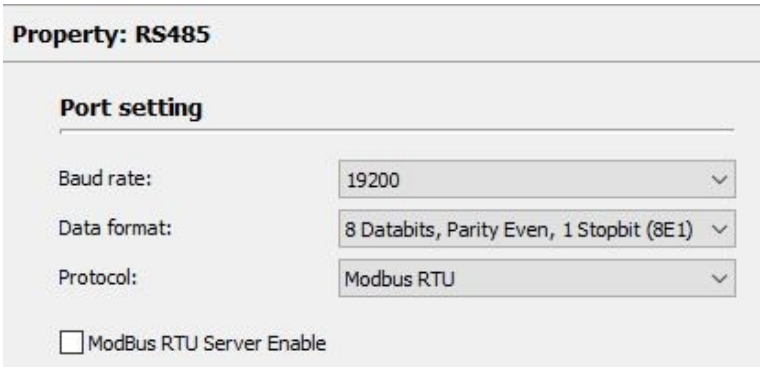
1 Byte: UID
1 Byte: command (function code): 04
2 Bytes: Register-Index
2 Bytes: number of registers to read

Configuration

Client:

RS485 interface:

- Baud rate 19200 (maximum)
- Data format 8E1 (recommended)
- Protocol Modbus RTU
- Server Enable OFF
- UID 1

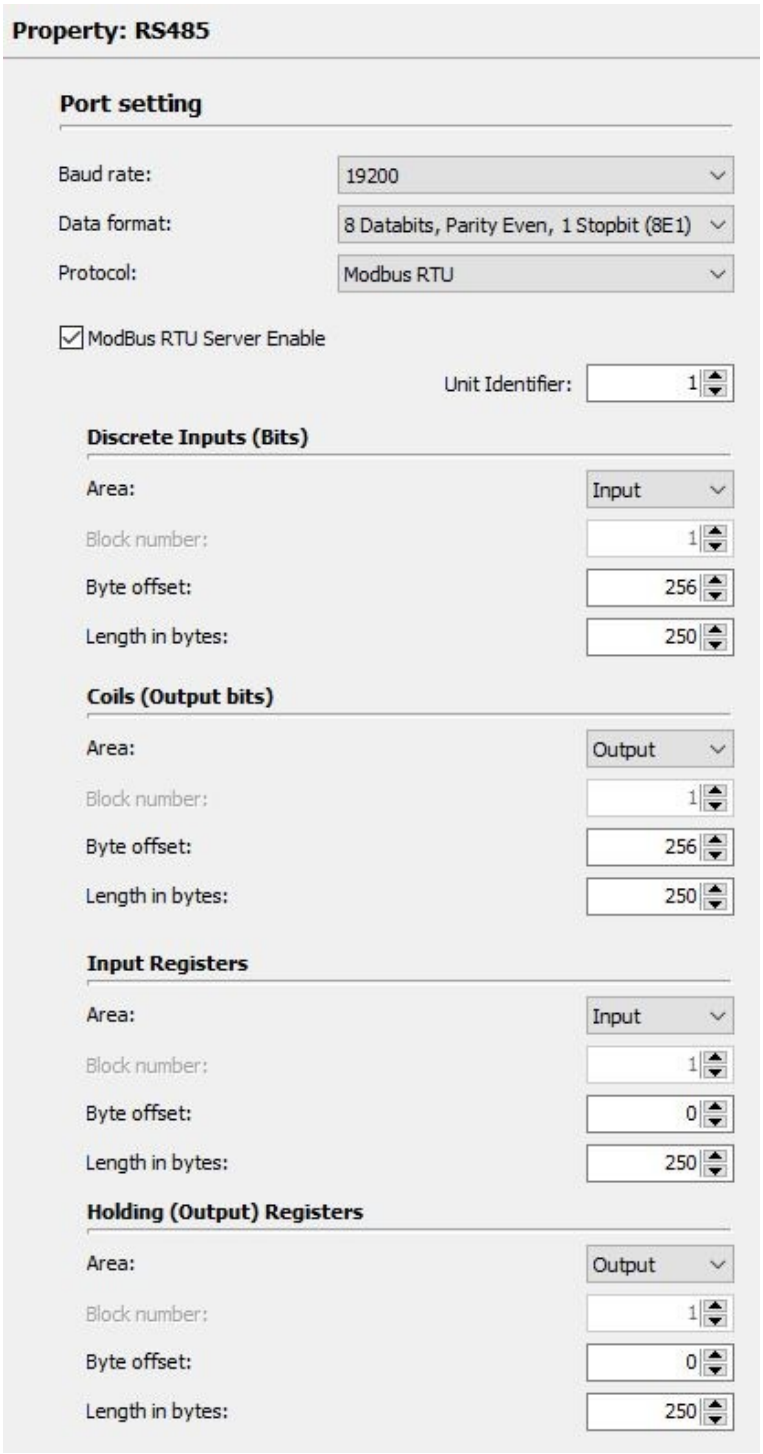


Server:

RS485 interface:

- Baud rate 19200 (maximum)
- Data format 8E1 (recommended)
- Protocol Modbus RTU
- Server Enable ON
- UID 1
- configuration of Inputs, Coils, Input registers and Holding registers

ATTENTION:
When activating the server NO further S7/TIA programming is required. Only the client function is programmed with S7/TIA.



S7 program

The operating system supports Modbus-RTU communication by low level send- and receive functions. These handles frame syncronisation and checksum calculation regarding Modbus specification.
The S7 code FB2 works as application interface:

```
CALL  "ModbusRTUClient" , "RTU_Instance"    // FB2 + DB2 as Instance-DB
  R         :=FALSE                         // Reset-input, to set once while startup
  UID       :=B#16#1                        // node address
  Cmd       :=B#16#2                        // Modbus command (function code 1,2,3,4,6,15,16)
  Index     :=0                             // Register rsp. Bit-address (0...65535)
  Length    :=2000                 // number of register (1..125) or bits (1..2000) to transfer
  Data      :="Daten_TCP".Inputs            // ANY-Pointer payload data area
  Done      :="RTU_done"                    // TRUE if well done
  Error     :="RTU_error"                   // TRUE in case of trouble
  ErrSrc    :="RTU_ErrorSrc"                // error code s. table
  ErrStatus:="RTU_ErrCode"
```

FB2 builds the telegramm, sends and receives and copies data into specified user area. Buffer for Modbus header and checksum is in FB2's instance datablock. A timeout of 1 second cancels receive and signals by errorflag. In case of detected CRC-errors or denied RTU-requests the errorflag is set too. The received function code is compared to the sended code, further invalid protocoll data are not handled.

FC2 is an example of Modbus RTU client application. For test reasons all implemented functions are called cyclically. This sequence **MUST BE customized** to the application's needs.
At the end of FC2 a dummy **error handler** is disposed, this must be **customized** to the application's needs too.

Error codes

Return values of FB2 are divided into error source (ErrSrc) and a status code (ErrStatus).
Error source accords with the last state of the state machine in FB2, as the error occurred. Related to the error source the status code contains information about the cause of error:

ErrSrc	ErrStatus	description
0	8001_{hex}	RS485 not configured as ModbusRTU
	else Status return of SFB 61 while initialization	
1	8001_{hex}	UID > 127
	8002_{hex}	Invalid CMD (function code)
	8003_{hex}	Invalid LEN (register > 250, bits/coils > 2000)
	8004_{hex}	Invalid area of payload data (≠ I, O, M, DB)
2	8000_{hex}	SFB 60 busy (send buffer overflow)
	else Status return SFB20 or SFB 60	
3	Status return SFB 61 while receive	
	$CAFE_{hex}$	Timeout
4	9001_{hex}	Bit 7 of returned function code set = Server denies request (invalide parameter)
	9000_{hex}	Returned function code invalid
	Status return of SFC 20 copying user data	

Modbus RTU Observer:

For troubleshooting and in case no PC with a RS485-UART interface is available, another PLC can be used to observe running communication. All received bytes are stored with a timestamp into a ringbuffer. The time resoltution of 10ms is too coarse to detect single frames but the requests by the client are apart enough
Notice that the hardware must be be configured by ConfigStage as plain ASCII

Procedure

- transfer configuration for Modbus RTU server into server PLC → RUN
- transfer S7 software into client PLC
- transfer configuration for Modbus RTU client into client PLC → RUN
- if needed transfer configuration for Modbus RTU observer into observer PLC → RUN
- monitor/control via watch tables with the following options:
    1. read all or certain Coils (function 01hex)
    2. read all or certain Inputs (function 02hex)
    3. read all or certain Holding registers (function 03hex)
    4. read all or certain Input registers (function 04hex)
    5. write single Coils (function 05hex)
    6. write single Holding register (function 06hex)
    7. write multiple/all Coils (function 0Fhex)
    8. write multiple/all Holding registers (function 10hex)

INSEVIS

INSEVIS Vertriebs GmbH

Am Weichselgarten 7
D - 91058 Erlangen

Fon: +49(0)9131-691-440
Fax: +49(0)9131-691-444
Web: www.insevis.de
E-Mail: info@insevis.de