KUNBUS
industrial communication

Anwenderhandbuch
KUNBUS-Scripter

PR100087R03_DE • 12.06.2019

Inhaltsverzeichnis

1 Übersicht

1.1 Übersicht

Mit dem KUNBUS Scripter ist es möglich, Daten über eine serielle Schnittstelle in einem beliebigen Format mit beliebigen Geräten auszutauschen.

Der KUNBUS-Scripter ist in drei Komponenten unterteilt:

KUNBUS Script Wizard

Der Script Wizard ermöglicht es Ihnen, ein Script ohne Programmierkenntnisse zu erstellen. Wir stellen Ihnen vordefinierte Templates für Query-Response-Telegramme zur Verfügung. Zudem haben Sie die Möglichkeit, eigene Templates für Befehle zu erstellen.

KUNBUS Script Editor

Mit dem Script Editor können Sie Ihr Script wie in einer Textsoftware editieren. Der Editor unterstützt Sie durch:

- Autovervollständigung (auto-complete)
- Syntaxhervorhebung (highlighting)

Beide Komponenten verfügen über eine Ablaufverfolgung (Trace). Mit dieser können Sie verfolgen, welche Werte die Variablen im Script annehmen und wo ein Fehler auftritt.

Wenn Sie ein Script im Script Wizard oder im Script Editor erstellt haben, können Sie das Script auf das Modul übertragen.

Interpreter auf dem Modul

Der Interpreter auf dem Modul arbeitet die übertragenen Scripte ab.

1.2 Funktionsweise

Ablauf der Kommunikation

Wenn Sie ein Script erstellt haben, wird dieses als PBS-Datei gespeichert. Die Abkürzung PBS steht für **P**rotocol **B**uilder **S**cript.

Beim Kompilieren wird aus dem Script eine Binärdatei erzeugt. Diese Binärdatei wird über die Common Debug Interface (CDI) zum Modul übertragen und im Flash abgelegt. Im Modul wird die Binärdatei abgearbeitet.

Über die CDI Schnittstelle wird der Trace zurück an den PC gesendet. Dadurch können Sie im Script Wizard und im Script Editor die aktuellen Werte der Variablen ansehen.

Scriptstruktur und -ausführung

Das Script besteht aus mehreren Sektionen. Jede dieser Sektionen hat eine bestimmte Aufgabe.

Das Script startet in der Sektion CONFIG über die Sektion INIT nach RUN, bzw. mehrere Sektionen RUN fortgesetzt. Das Script wird zyklisch ausgeführt.

- SECTION_CONFIG
  Befehle in dieser Sektion sind zur einmaligen Konfiguration des Moduls. Hier können Sie folgende Einstellungen an der seriellen Schnittstelle festlegen:
  - Bitrate
  - Bytereihenfolge für das Protokoll
  - Deklaration der Variablen
- SECTION_INIT
  Befehle in dieser Sektion sind zum Initialisieren des Moduls. Sie können hier Variablen und Modbus-Register auf Startwerte setzen.
- SECTION_RUN
  Das Script kann mehrere Paare aus SECTION_RUN und SECTION_ERROR enthalten die nacheinander ausgeführt werden. Nach der letzten SECTION_RUN springt das Protokoll wieder zur ersten SECTION_RUN.
- SECTION_ERROR
  Wenn in SECTION_RUN ein Fehler auftritt, wird die folgende SECTION_ERROR ausgeführt. Danach wird mit der Ablauf des Scripts in der folgenden SECTION_RUN fortgesetzt.
  Wenn in SECTION_ERROR erneut ein Fehler auftritt wird die SECTION_FATAL als nächstes ausgeführt.
- SECTION_FATAL
  In dieser Sektion treten Fehler auf, bei denen es sich um schwerwiegende Fehler handelt. Diese Fehler konnten in der SECTION_ERROR nicht behoben werden.
  Nach dem Code von SECTION_FATAL wird das Script mit SECTION_INIT neu gestartet.
  Mit dem Befehl stop() können Sie die Scriptausführung manuell anhalten. Das Script startet dann nach dem nächsten Modulreset erneut.

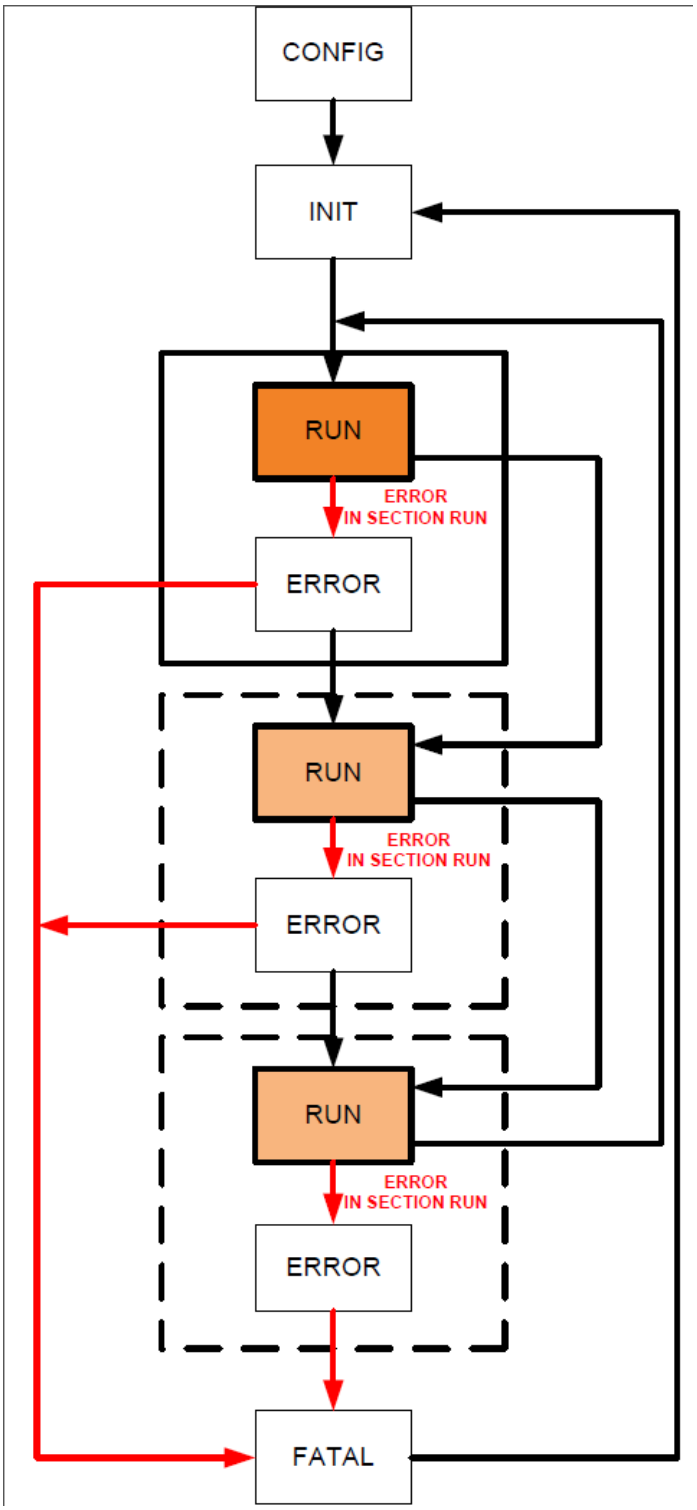


Abb. 1: Scriptstruktur

Der aktuellen Implementierung liegt ein zweistufiges Fehler- bzw. Ausnahmemodell zugrunde. Dieses Modell steuert den Ablauf in die einzelnen Sektionen durch implizite Sprünge. Befindet sich das Script gerade in der Sektion RUN, erfolgt beim Auftreten eines Fehlers ein Sprung zur folgenden ERROR Sektion (erste Fehlerstufe). Nachdem die Befehle in dieser Sektion abgearbeitet sind springt das Script an den Anfang der nächsten Sektion RUN. Ist die aktuelle Sektion RUN die letzte im Ablauf, erfolgt der Sprung zur ersten Sektion RUN.

Die Sektion INIT enthält in der Regel keine kritischen Befehle, die mit der Zielanwendung interagieren. Solle in dieser Sektion dennoch ein Fehler auftreten, springt das Script in die Sektion FATAL (zweite Fehlerstufe). Auch ein Fehler innerhalb der Sektion ERROR führt zum Sprung nach FATAL. Nach der Sektion FATAL wird das Script neu gestartet, d. h. ab der Sektion INIT ausgeführt. Um einen Neustart zu verhindern, z.B. weil im Fehlerfall eine Fortsetzen des Scripts nicht sinnvoll erscheint, kann ein stop() Kommando eingefügt werden.

1.3 Startmenü

Auf unserer Homepage (www.kunbus.de) erhalten Sie die Datei „Setup_KUNBUS_Scripter.exe“.

- Starten Sie die Datei mit einem Doppelklick

⇨ Das Startmenü öffnet sich

Im Startmenü des KUNBUS-Scripter können Sie auswählen, wie Sie Ihr Protokoll erstellen möchten.

– Die Funktion „Script Wizard“ erfordert nur geringe Programmierkenntnisse. Sie können auf eine Scriptvorlage zurückgreifen oder aus vorhandenen Bausteinen selbst ein Script zusammenstellen.

– Mit der Funktion „Script Editor“ schreiben Sie das Script in einem Texteditor. Hier stellen wir Ihnen einige Features zur Verfügung, die Ihnen die Arbeit erleichtern sollen.

- Wählen Sie die gewünschte Funktion aus und bestätigen Sie die Eingabe mit „OK“

Abb. 2: Startmenü

2 KUNBUS Script Editor

2.1 Übersicht

Übersicht Script Editor

Die Arbeitsoberfläche des Script-Editors ist in drei Fenster unterteilt.

- Im Scripteditor (5) schreiben Sie Ihr Script
- Im Anzeigefenster für Meldungen (9) erhalten Sie Informationen zum Status Ihres Scriptes
- Auf dem Trace-Monitor (7) können Sie den Ablauf Ihres Scriptes verfolgen.

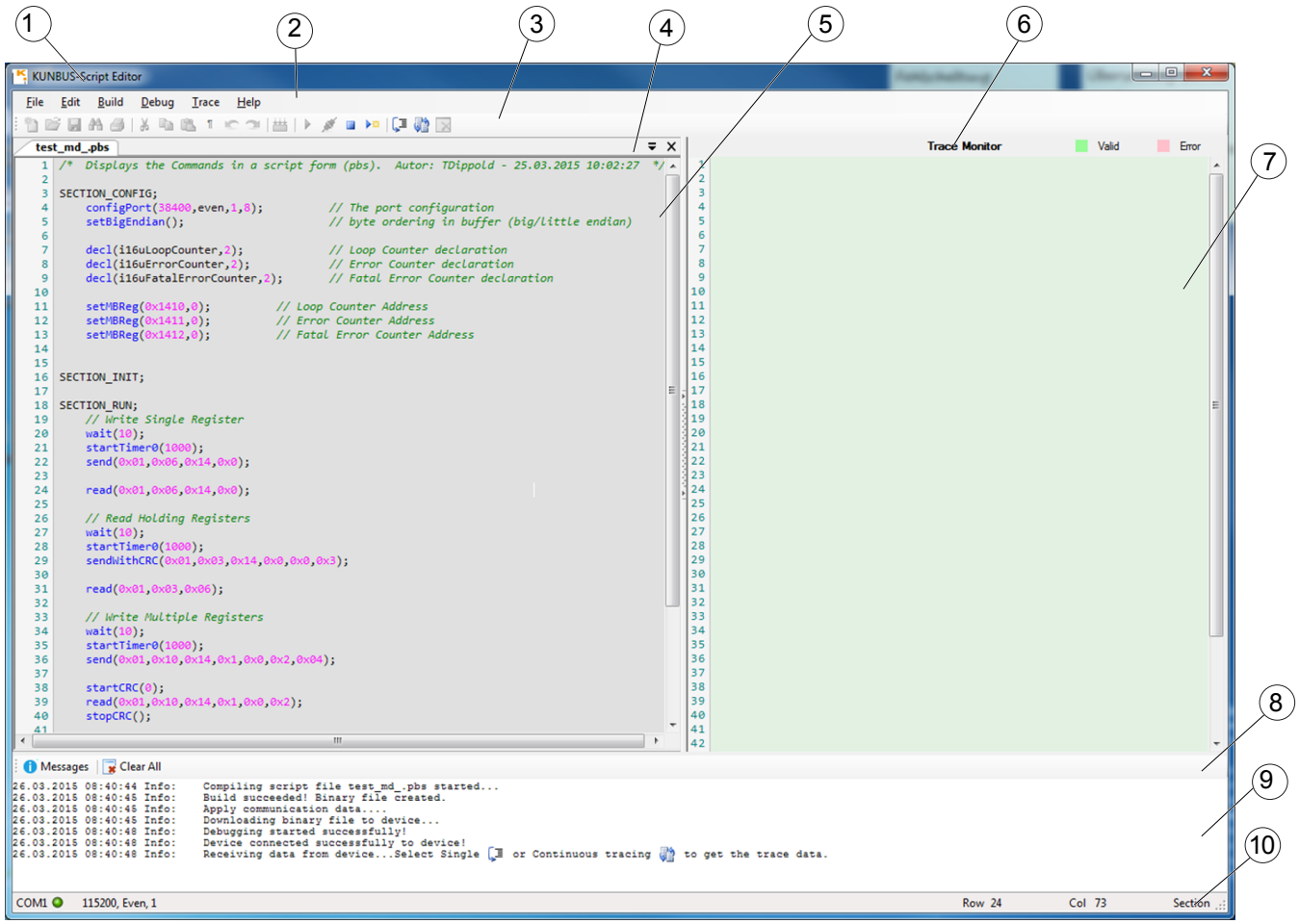


Abb. 3: Protocol Builder

1	Fensterleiste	2	Menüleiste
3	Symbolleiste	4	Editor Fensterleiste
5	Scripteditor	6	Trace Statusleiste
7	Trace Monitor	8	Toolbar des Nachrichtenfensters
9	Anzeigefenster für Meldungen	10	Statusleiste

2.2 Menüleiste

Dieses Kapitel beschreibt die einzelnen Funktionen in der Menüleiste.

1 File

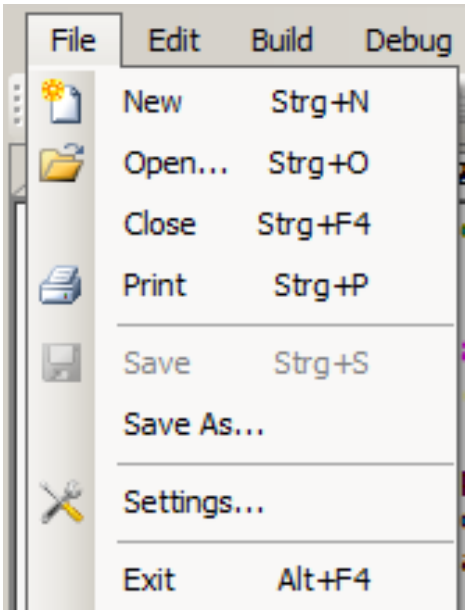


Abb. 4: Menü File

1.1 New:

Ein leeres Script öffnen.

1.2 Open:

Ein vorhandenes Script öffnen.

1.3 Close:

Das geöffnete Script schliessen.

1.5 Print:

Script ausdrucken.

1.6 Save:

Script als PBS-Datei speichern.

1.7 Save As:

Script als PBS-Datei speichern. Sie können dabei den Dateinamen und den Speicherort festlegen.

1.8 Settings

1.8.1 Connection Settings

Kommunikationseinstellungen zwischen PC und Modul festlegen.

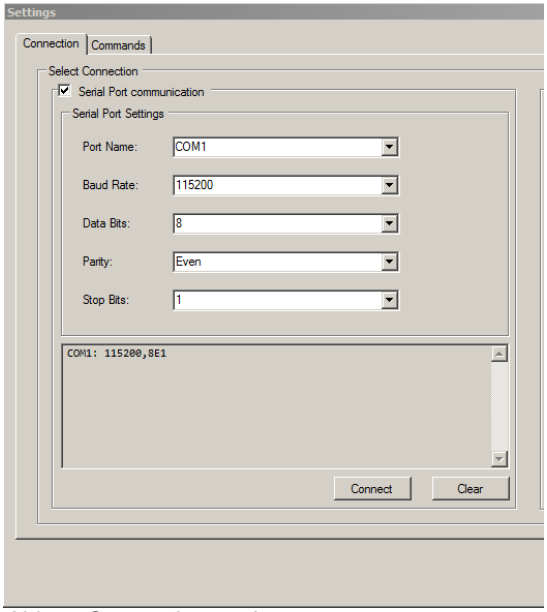


Abb. 5: Connection settings

1.8.2 Connection Commands

Übersicht der zulässigen Syntax.

Info!: Dieses Menü ist rein informativ. Sie können hier keine eigenen Werte vergeben**.**

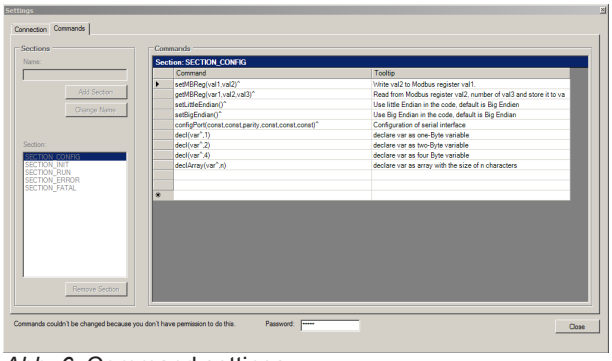
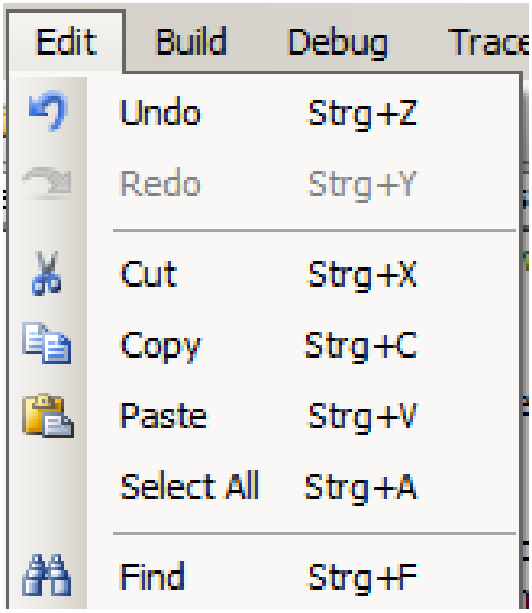


Abb. 6: Command settings

1.9 Exit:

Programm beenden.

2. Edit



2.1 Undo:

Letzte Eingabe rückgängig machen.

2.2 Redo:

Letzte Eingabe wiederholen.

2.3 Cut:

Einen markierten Bereich ausschneiden.

2.4 Copy:

Einen markierten Bereich kopieren.

2.5 Paste:

Einen zuvor kopierten oder ausgeschnittenen Bereich an der markierten Stelle einfügen.

2.6 Select All:

Den gesamten Editor-Bereich markieren.

2.7 Find:

Einen Begriff im Editor suchen.

3. Build

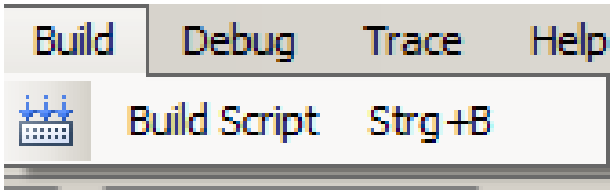


Abb. 7: Menü Build

3.1 Build Script:

Script auf Fehler prüfen und eine Binärdatei erzeugen.

4. Debug

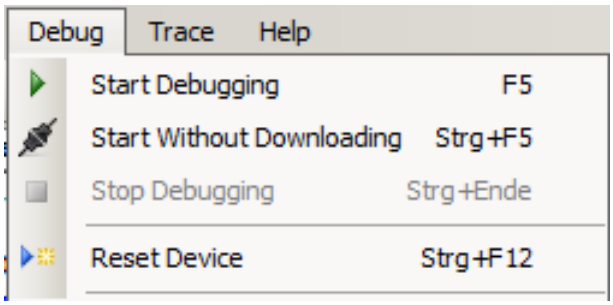


Abb. 8: Menü Debug

4.1 Start Debugging:

Script auf Fehler prüfen und eine Binärdatei erzeugen.Wenn die Datei fehlerfrei ist, wird sie auf das Modul geladen und ausgeführt. Ab jetzt besteht eine Debug-Verbindung zum Modul. Der Trace kann angezeigt werden.

4.2 Start Without Downloading:

Prüfen, ob das Script im Editor und das Script auf dem Modul identisch sind. Ist das der Fall, wird eine Trace-Verbindung hergestellt. Ab jetzt kann der Trace angezeigt werden. Das Script wird dabei nicht unterbrochen oder neu gestartet.

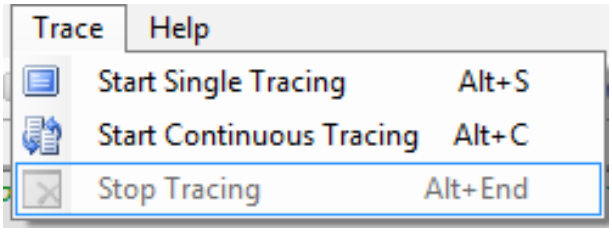
4.3 Stop Debugging:

Die Kommunikation mit dem Modul unterbrechen.

4.4 Reset Device:

Modul neu starten.

5. Trace



5.1 Start Single Tracing:

Ablauf des Scipts im Tracemonitor anzeigen. Im Single Tracing Modus wird ein einzelner Scriptablauf abgefragt. Diese Funktion ist hilfreich, wenn Sie eine bestimmte Stelle im Scriptablauf genau betrachten wollen.

5.2 Start Continuous Tracing:

Ablauf des Scipts im Tracemonitor anzeigen. Im Continuous Tracing Modus wird der Trace zyklisch abgefragt.

5.3 Stop Tracing:

Beendet die Trace-Verbindung.

6. Help

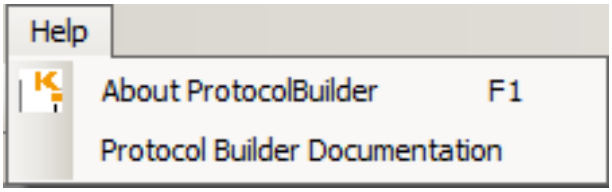


Abb. 9: Menü Help

6.1 About KUNBUS Scripter:

Zeigt Systeminformationen zum KUNBUS-Scripter.

6.2 KUNBUS Scripter Documentation:

Zeigt Hilfethemen zum Scripter an.

Hier finden Sie u. a. auch die Syntax.

2.3 Symbolleiste

Über die Symbolleiste haben Sie schnellen Zugriff auf folgende Aktionen:

Neu

Ein leeres Script öffnen.

Öffnen

Ein vorhandenes Script öffnen.

Speichern

Script als PBS-Datei speichern.

Suchen

Einen Begriff im Editor suchen.

Drucken

Script ausdrucken.

Ausschneiden

Einen markierten Bereich ausschneiden.

Kopieren

Einen markierten Bereich kopieren.

Einfügen

Einen zuvor kopierten oder ausgeschnittenen Bereich an der markierten Stelle einfügen.

Steuerzeichen

Blendet Steuerzeichen ein oder aus.

Eingabe rückgängig machen

Letzte Eingabe rückgängig machen.

Eingabe wiederholen	 Letzte Eingabe wiederholen.
Script kompilieren	 Script auf Fehler prüfen und eine Binärdatei erzeugen.
Start Debug	 Script auf Fehler prüfen und eine Binärdatei erzeugen.Wenn die Datei fehlerfrei ist, wird sie auf das Modul geladen und ausgeführt. Ab jetzt besteht eine Debug-Verbindung zum Modul. Der Trace kann angezeigt werden.
Start Whitout Downloading	 Prüfen, ob das Script im Editor und das Script auf dem Modul identisch sind. Ist das der Fall, wird eine Trace-Verbindung hergestellt. Ab jetzt kann der Trace angezeigt werden. Das Script wird dabei nicht unterbrochen oder neu gestartet.
Stop Debug	 Beendet die Debug-Verbindung.
Reset Module	 Modul neu starten.
Start Single Tracing	 Ablauf des Scipts im Tracemonitor anzeigen. Im Single Tracing Modus wird ein einzelner Scriptablauf abgefragt. Diese Funktion ist hilfreich, wenn Sie eine bestimmte Stelle im Scriptablauf genau betrachten wollen.
Start Continuous Tracing	 Ablauf des Scipts im Tracemonitor anzeigen. Im Continuous Tracing Modus wird der Trace zyklisch abgefragt.
Stop Tracing	 Beendet die Trace-Verbindung.

2.4 Script Editor

Im Texteditor können Sie Ihr Script schreiben.

Beachten Sie dazu die vorgegebene Funktionsweise [▶ 4] und die verwendeten Übersicht der Scriptbefehle [▶ 43]

Navigation und Orientierungshilfen

Navigation

Sie haben die Möglichkeit mehrere Scripts im Editor zu öffnen. Mit jedem Script erscheint in der Navigationsleiste, am oberen Fensterrand des Editors ein neuer Reiter. Zwischen diesen Ansichten schalten Sie um, indem Sie:

- Auf den gewünschten Reiter klicken oder
- auf den nach unten gerichteten Pfeil auf der rechten Seite klicken und das gewünschte Script anklicken.

Syntaxhervorhebung

Funktionen, Kommentare, Werte und Variablen sind zur leichteren Orientierung farbig dargestellt

- Funktionen: blau
- Werte: pink
- Variablen: schwarz
- Kommentare: grün

test_md_.pbs

```
/*  Displays the Commands in a script form (pbs).  Autor: TDippold - 25.03.2015 10:02:27  */

SECTION_CONFIG;
    configPort(38400,even,1,8);         // The port configuration
    setBigEndian();                     // byte ordering in buffer (big/little endian)

    decl(i16uLoopCounter,2);            // Loop Counter declaration
    decl(i16uErrorCounter,2);           // Error Counter declaration
    decl(i16uFatalErrorCounter,2);      // Fatal Error Counter declaration

    setMBReg(0x1410,0);          // Loop Counter Address
    setMBReg(0x1411,0);          // Error Counter Address
    setMBReg(0x1412,0);          // Fatal Error Counter Address


SECTION_INIT;

SECTION_RUN;
    // Write Single Register
    wait(10);
    startTimer0(1000);
    send(0x01,0x06,0x14,0x0);

    read(0x01,0x06,0x14,0x0);

    // Read Holding Registers
    wait(10);
    startTimer0(1000);
    sendWithCRC(0x01,0x03,0x14,0x0,0x0,0x3);

    read(0x01,0x03,0x06);

    // Write Multiple Registers
    wait(10);
    startTimer0(1000);
    send(0x01,0x10,0x14,0x1,0x0,0x2,0x04);

    startCRC(0);
    read(0x01,0x10,0x14,0x1,0x0,0x2);
    stopCRC();

    add(i16uLoopCounter,1);
    setMBReg(0x1410,i16uLoopCounter);

SECTION_ERROR;
    add(i16uErrorCounter,1);
    setMBReg(0x1411,i16uErrorCounter);

SECTION_FATAL;
    add(i16uFatalErrorCounter,1);
    setMBReg(0x1412,i16uFatalErrorCounter);
```

Abb. 10: Syntaxhervorhebung

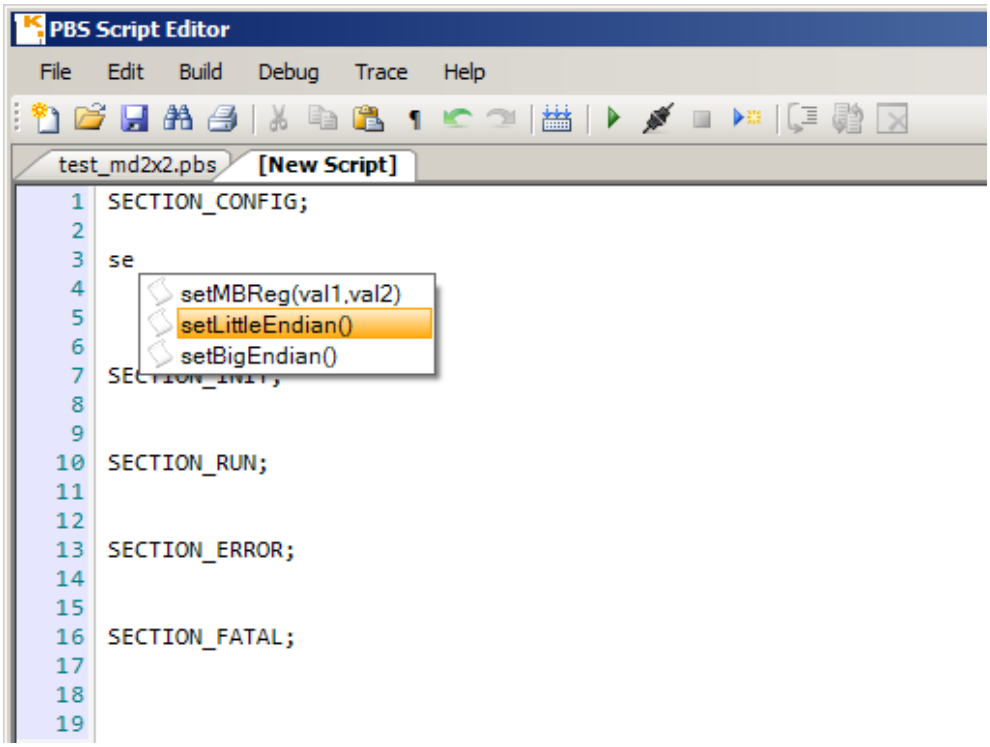
Arbeitserleichternde Funktionen

Automatische Textergänzung

Der KUNBUS-Scripter unterstützt Sie durch automatische Textergänzung der zulässigen Syntax.

- Schreiben Sie im Editor die ersten Buchstaben, der gewünschten Syntax in den Editor.
  - ⇨ Bereits nach dem ersten Buchstaben öffnet sich ein Auswahlfenster.
- Klicken Sie die gewünschte Syntax an.

⇨ Die ausgewählte Syntax wird vom Editor übernommen.



Einrücken von Textblöcken

- Markieren Sie den gewünschten Textblock
- Wählen Sie im Kontextmenü „AutoIndent selected text“

⇨ Der Ausgewählte Text wird nun eingerückt dargestellt.

2.5 Script starten

Voraussetzung: Sie haben ein Script im Editor erstellt oder ein Vorhandenes geöffnet.

- Wählen Sie in der Symbolleiste die Funktion „Verbinden“
  - ⇨ Es wird eine Verbindung zum Modul hergestellt
- Wählen Sie in der Symbolleister die Funktion „Compile File“
  - ⇨ Mit dieser Funktion kompilieren Sie das Script.
- Wählen Sie in der Symbolleister die Funktion „Start Debug “
  - ⇨ Mit dieser Funktion starten Sie das Script.

⇨ Beenden Sie die Verbindung mit der Auswahl

2.6 Trace Monitor

Auf dem Trace-Monitor, haben Sie die Möglichkeit, den Ablauf Ihres Scripts zu verfolgen. Das Trace-Monitor Fenster öffnet sich auf der rechten Seite, neben dem Editor-Fenster. Das Fenster öffnet sich automatisch, sobald Sie eine Trace-Funktion wählen.

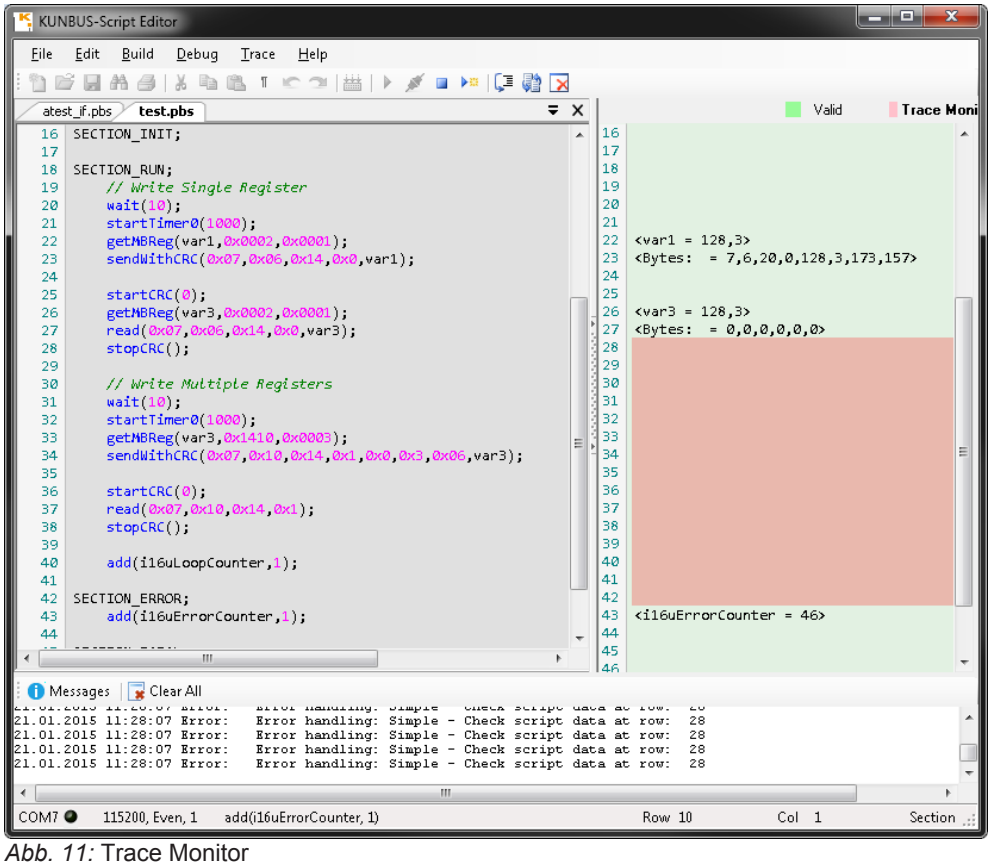


Abb. 11: Trace Monitor

Darstellung

- In der Leiste auf der linken Fensterseite können Sie verfolgen, in welcher Zeile des Scripts eine Meldung erscheint.
- Zeilen des Scripts, die ausgeführt wurden sind grün hinterlegt.
- Zeilen des Scripts, die nicht ausgeführt wurden sind rot hinterlegt.

Tracing-Funktionen

In der Navigationsleiste, im Menüpunkt „Trace“ finden Sie alle Tracing-Funktionen:

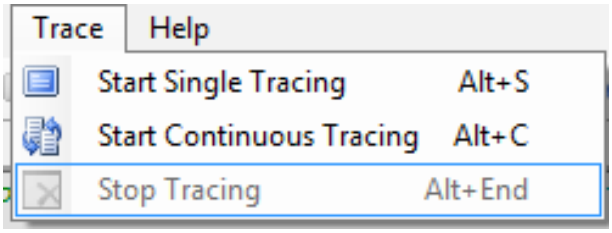
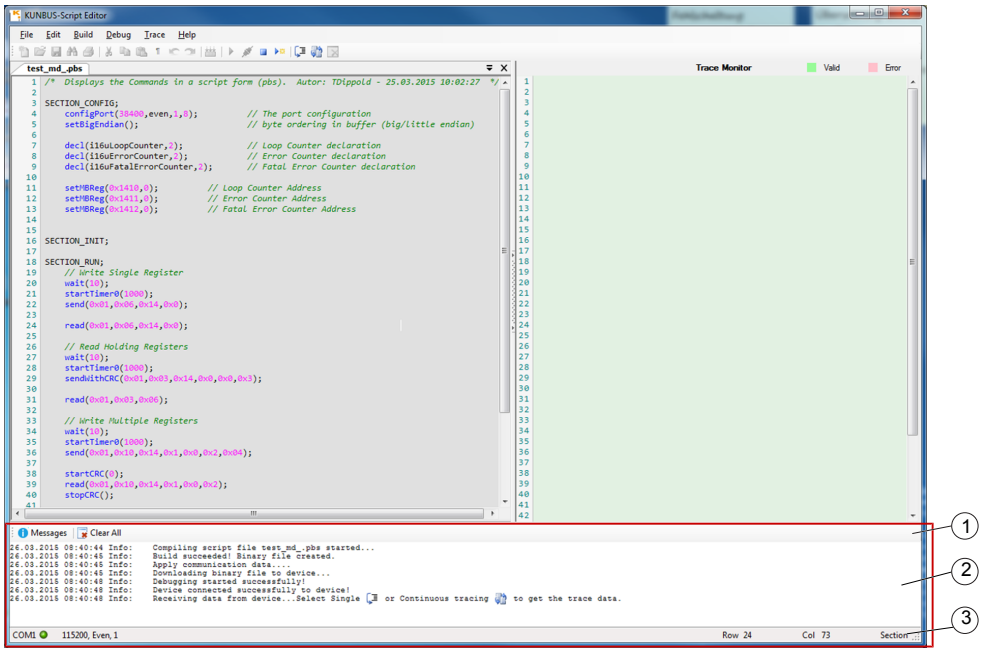


Abb. 12: Menü Trace

Start Single Tracing	Mit dieser Funktion haben Sie die Möglichkeit, den Ablauf Ihres Scripts schrittweise zu verfolgen.
Start Contiuous Tracing	Mit dieser Funktion haben Sie die Möglichkeit, den Ablauf Ihres Scripts zu verfolgen. In diesem Modus wird das Script zyklisch ausgeführt.
Stop Tracing	Diese Funktion beendet das Tracing.

2.7 Nachrichtenfenster

Das Nachrichtenfenster befindet sich unter dem Editorfenster.



1	Toolbar
2	Anzeigefenster für Meldungen
3	Statusleiste

Anzeigefenster für Nachrichten

In der Nachrichtenbox erhalten Sie folgende Informationen:

- Status des Scripts
- Fehlerbenachrichtigungen
- Weiterführende Anweisungen

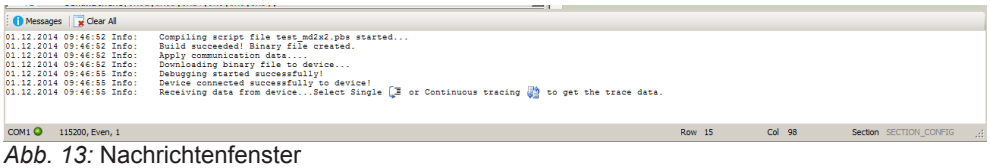


Abb. 13: Nachrichtenfenster

Tipp!: Durch einen Klick in eine Nachrichtenzeile springt der Cursor auf die entsprechende Zeile im Script.

Toolbar

In der Fensterleiste können Sie alle Nachrichten löschen. Klicken Sie dazu auf „Clear All“.

Statusleiste

In der Fußleiste erhalten Sie Informationen zu den Kommunikationseinstellungen zum Modul.

Die farbige Statusanzeige signalisiert ob eine Verbindung zum Modul besteht:

Rot	Keine Verbindung zum Modul
Grün	Verbindung zum Modul

Im abgebildeten Beispiel können Sie in der Fußleiste erkennen, dass eine Verbindung zum Modul besteht. Die Kommunikationseinstellungen sind: 115200 bit/s, Even Parity, 1 Stoppbit.

3 KUNBUS Script Wizard

3.1 Übersicht

Mit dem Script Wizard können Sie ein Script zusammenstellen. Sie benötigen dazu keine Programmierkenntnisse.

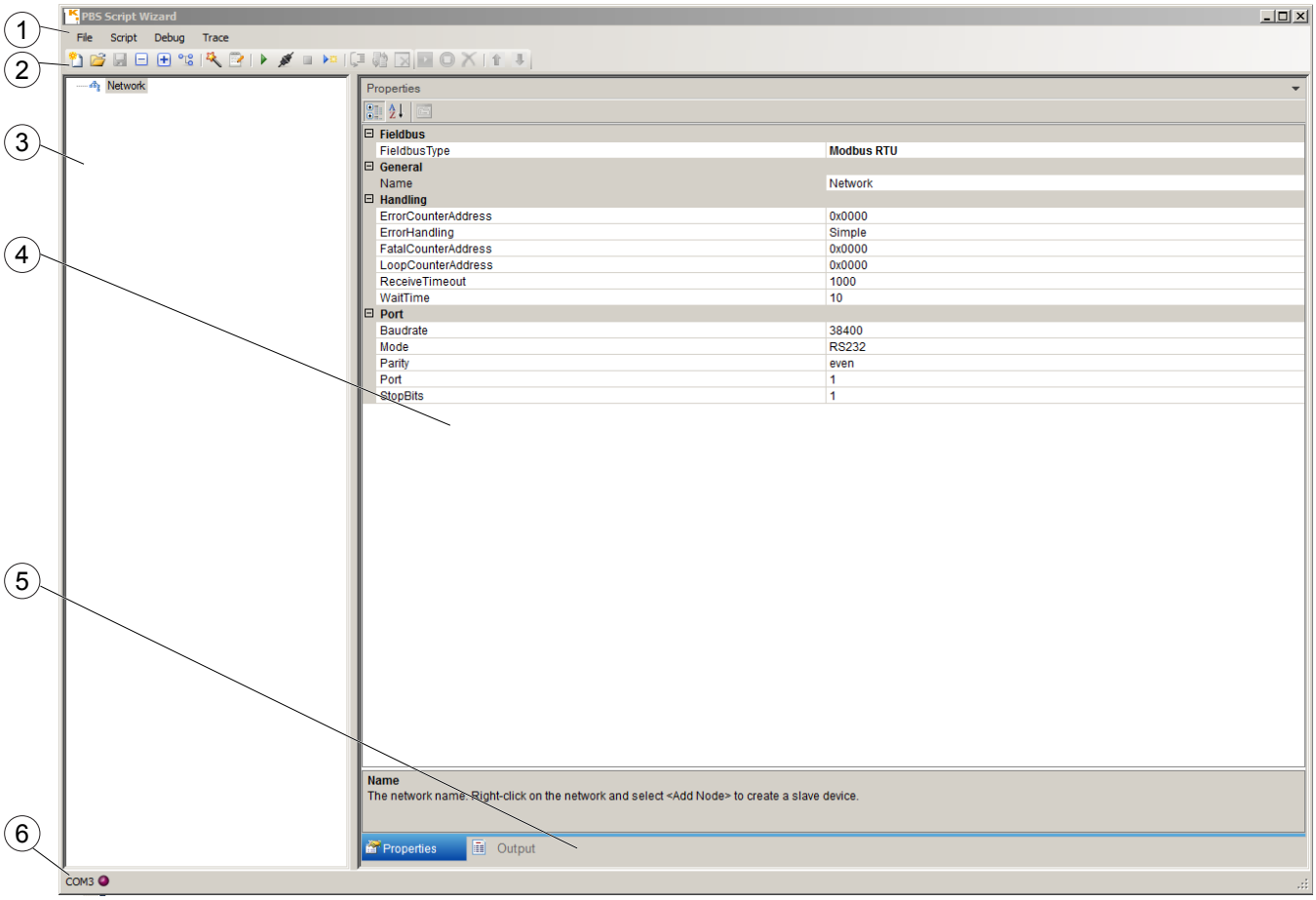


Abb. 14: Startansicht

1	Menüleiste
2	Symbolleiste
3	Strukturbaum (Treeview)
4	Arbeitsbereich
5	Registerleiste
6	Statusleiste

3.2 Menüleiste

Dieses Kapitel beschreibt die einzelnen Funktionen in der Menüleiste.

1 Menü File

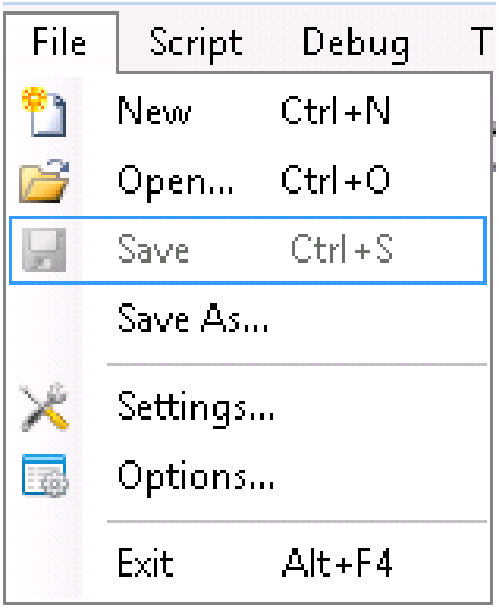


Abb. 15: Menü File

1.1 New:

Ein leeres Script öffnen.

1.2 Open:

Ein vorhandenes Script öffnen.

1.3 Save:

Wizard Projekt als XML-Datei speichern.

1.4 Save As:

Wizard Projekt als XML-Datei und das Sript als PBS-Datei speichern. Sie können dabei Speicherort und Dateiname bestimmen.

1.5 Settings:

1.5.1 Connection Settings

Kommunikationseinstellungen zwischen PC und Modul festlegen.

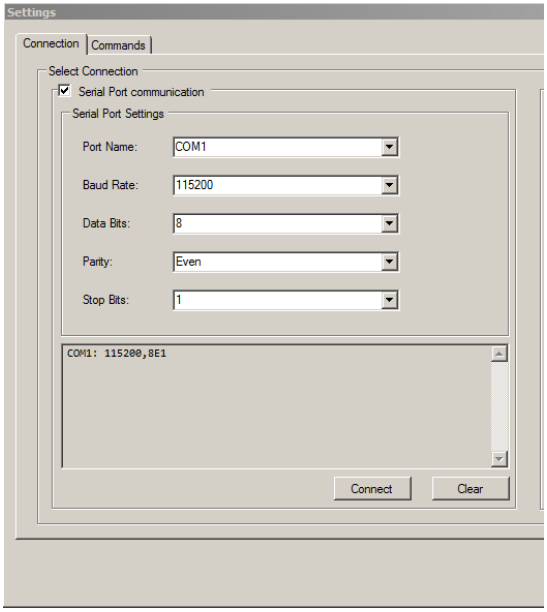


Abb. 16: Connection settings

1.5.2 Command Settings

Übersicht der zulässigen Syntax.

Info!: Dieses Menü ist rein informativ. Sie können hier keine eigenen Werte vergeben**.**

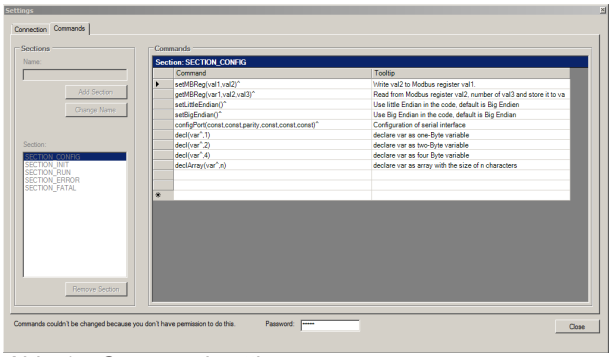
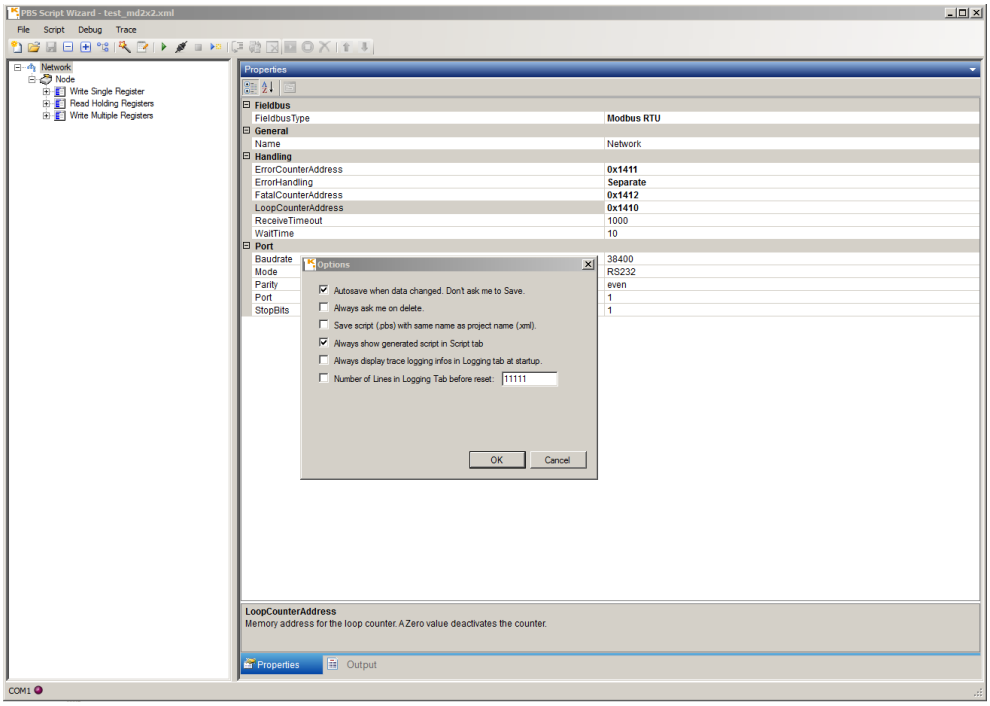


Abb. 17: Command settings

1.6 Options:

Voreinstellungen festlegen:

- Automatisches Speichern nach Datenänderung
- Rückfrage vor dem Löschen
- Automatische Namensvergabe beim Speichern
- Öffnen des Script, nachdem es generiert wurde, in einem separaten Tab.
- Anzeige des Trace-Logging beim Start
- Maximale Anzahl der Zeilen, nach denen ein Logging gelöscht werden soll.



1.7 Exit:

Programm beenden.

2. Menü Script

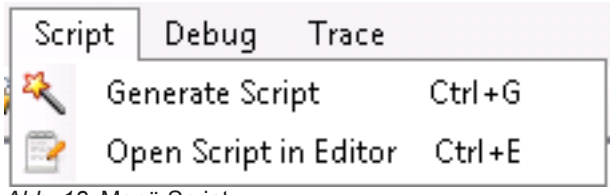


Abb. 18: Menü Script

2.1 Generate Script

Script generieren.

2.2 Open Script in Editor

Das generierte Script im Script Editor öffnen.

3 Menü Debug

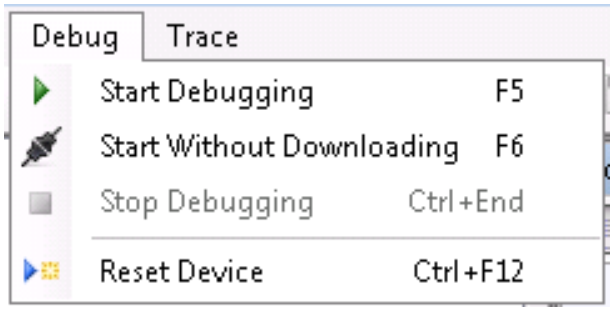


Abb. 19: Menü Debug

3.1 Start Debugging

Diese Funktion führt folgende Aktionen aus:

- Script erzeugen
- Script compilieren
- Script zum Modul übertragen
- Script starten
- Traceverbindung öffnen

3.2 Start Without Downloading

Prüfen, ob das Script im Editor und das Script auf dem Modul identisch sind. Ist das der Fall, wird eine Trace-Verbindung hergestellt. Ab jetzt kann der Trace angezeigt werden. Das Script wird dabei nicht unterbrochen oder neu gestartet.

3.3 Stop Debugging

Beendet die Trace-Verbindung.

3.4 Reset Module

Modul neu starten.

4. Menü Trace

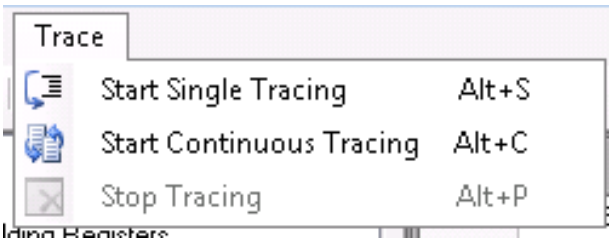


Abb. 20: Menü Trace

4.1 Start Single Trace

Einen Trace vom Modul laden. Die Statusanzeige im Strukturbaum zeigt an, welche Teile des Scripts erfolgreich ausgeführt wurden. Bei einer grünen Statusmeldung war die Ausführung erfolgreich. Eine rote Statusmeldung signalisiert einen Fehler.

4.2 Start Continuous Trace

Trace zyklisch abfragen. Das Ergebnis wird im Strukturbaum angezeigt.

4.3 Stop Trace

Beendet die Trace-Verbindung.

3.3 Symbolleiste

Über die Symbolleiste haben Sie schnellen Zugriff auf folgende Aktionen:

Neu

Ein leeres Script öffnen.

Öffnen

Ein vorhandenes Script öffnen.

Speichern

Script als PBS-Datei speichern.

Einklappen

Einen ausgewählten Befehl im Strukturbaum einklappen.

Ausklappen

Einen ausgewählten Befehl im Strukturbaum ausklappen.

Alle Ein-/Ausklappen

Alle Befehle im Strukturbaum ein-/ausklappen.

Script generieren

Ein Script generieren.

Wechsel in den Editor

Script im Script-Editor öffnen.

Start Debug

Diese Funktion führt folgende Aktionen aus:

- Script erzeugen
- Script compilieren
- Script zum Modul übertragen
- Script starten
- Traceverbindung öffnen

Start Without Downloading 

Prüfen, ob das Script im Editor und das Script auf dem Modul identisch sind. Ist das der Fall, wird eine Trace-Verbindung hergestellt. Ab jetzt kann der Trace angezeigt werden. Das Script wird dabei nicht unterbrochen oder neu gestartet.

Stop Debug 

Beendet die Trace-Verbindung

Reset Module 

Modul neu starten.

Start Single Tracing 

Einen Trace vom Modul laden. Die Statusanzeige im Strukturbaum zeigt an, welche Teile des Scripts erfolgreich ausgeführt wurden. Bei einer grünen Statusmeldung war die Ausführung erfolgreich. Eine rote Statusmeldung signalisiert einen Fehler.

Start Continuous Tracing 

Trace zyklisch vom Modul laden. Das Ergebnis wird im Strukturbaum angezeigt.

Stop Tracing 

Beendet die Trace-Verbindung.

Start Logging 

Logging starten. Das Logging ist der Trace in im Loggingfenster in kontinuierlich angezeigt.

Stop Logging 

Logging stoppen.

Clear Logging 

Logging löschen.

Nach oben springen 

In die erste Zeile des Logging springen.

Nach unten springen 

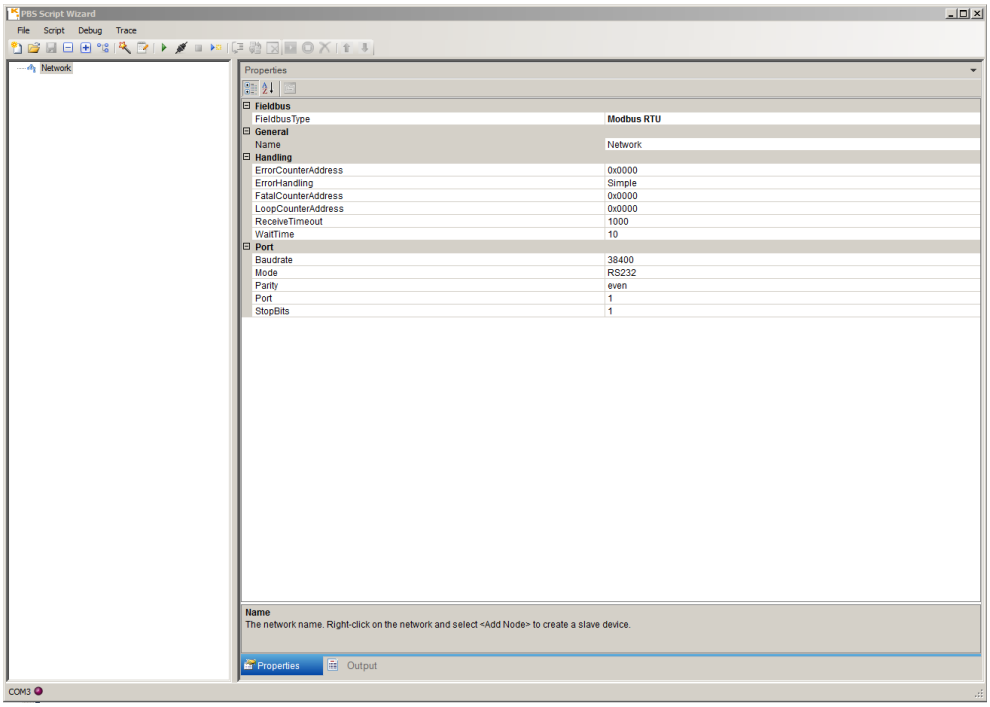
In die letzte Zeile des Logging springen.

3.4 Script erstellen mit dem KUNBUS-Wizard

Öffnen Sie den KUNBUS-Wizard

Properties

Im Arbeitsfenster sehen Sie das Register Eigenschaften („Properties“). Hier müssen sie die Einstellungen für die Kommunikation mit dem Modul festlegen:



Feld	Einstellung	Wertebereich
FieldbusType	Tragen Sie den Feldbus ein, den Sie verwenden.	
Name	Vergeben Sie einen Namen für das Netzwerk	
ErrorCounterAddress	Tragen Sie hier das Zielregister für den Fehlerzähler ein.	SDI -Eingangs-datenbereich 0x1401-0x1480 Um die Funktion zu de-aktivieren, tragen Sie in dieses Feld 0x000 ein
ErrorHandling	Tragen Sie den Modus der Fehleranzeige ein	Jedes Command hat seine eigene Fehlerbe-hand
FatalCounterAddress	Tragen Sie hier das Zielregister für den Zäh-ler für Fatale Fehler ein.	SDI -Eingangs-datenbereich 0x1401-0x1480 Um die Funktion zu deaktivieren, tragen Sie in dieses Feld 0x000 ein

Feld	Einstellung	Wertebereich
LoopCounterAddress	Tragen Sie hier das Zielregister für den Zähler für den Script-durchlauf ein.	SDI- Eingangs-datenbereich 0x1401-0x1480 Um die Funktion zu deaktivieren, tragen Sie in dieses Feld 0x000 ein
ReceiveTimeout	Tragen Sie hier die Wartezeit bis zur Antwort des Moduls ein	Wartezeit in ms
Wait Time	Tragen Sie die Warte-zeit zwischen den Scriptabläufen ein.	Wartezeit in ms
Baudrate	Tragen Sie die Baudrate ein	2400 4800 9600 19200 38400 57600 115200
Parity	Tragen Sie die Parity ein.	Even, odd, none
StopBits	Tragen Sie die Anzahl der Stop Bits ein.	0-2

Ein Gerät hinzufügen

- Klicken Sie in der Baumansicht mit der rechten Maustaste auf die Zeile „Network“
- Wählen Sie „Add node“.

⇨ Sie haben ein neues Gerät hinzugefügt.

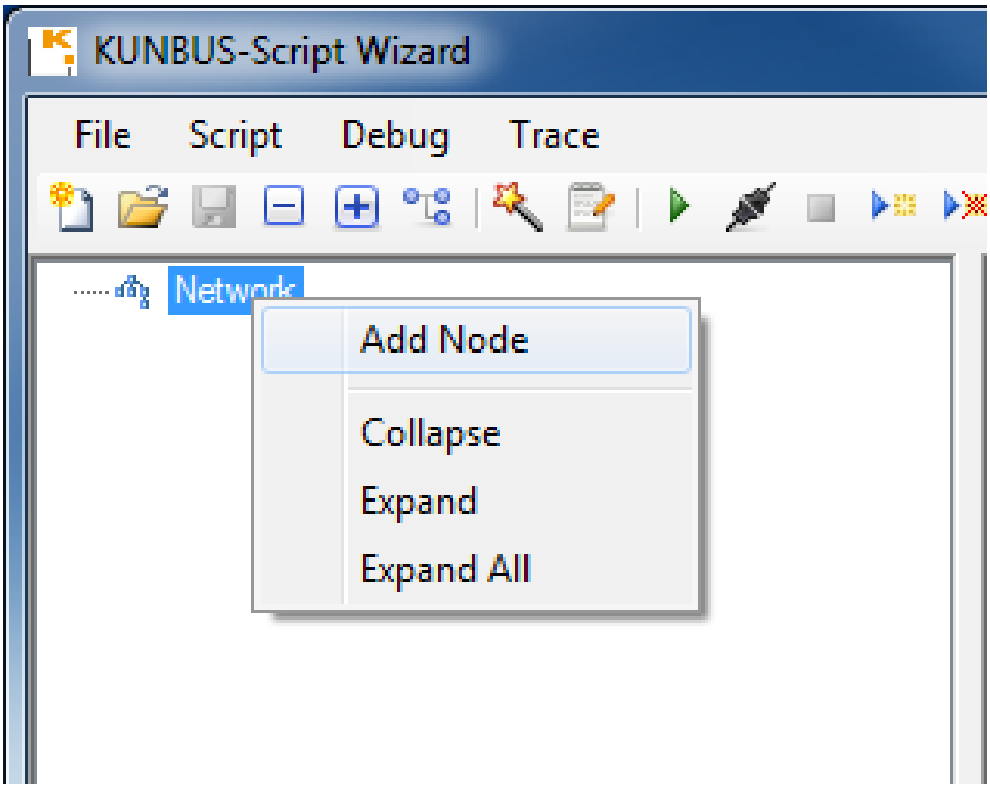


Abb. 21: Knoten hinzufügen

Im Fenster „Properties“ können Sie eine Adresse und einen Namen für das Gerät festlegen. Mit dieser Adresse können Commands dem Gerät zugeordnet werden.

Command

Ein Command besteht jeweils aus einem Request und einer Response. Der Request wird vom Script an den Node geschickt. Dann wird auf eine Response gewartet. Da ein Command meistens mehrfach mit unterschiedlichen Parametern verwendet wird gibt es im Command Editor vordefinierte Commands die man auswählen kann.

Ein neues Command hinzufügen

- ✓ Sie haben bereits ein neues Gerät (Node) im Strukturbaum hinzugefügt
  - ◦ Klicken Sie in der Baumansicht mit der rechten Maustaste auf das Gerät, dem Sie ein Command zuweisen möchten.
  - ◦ Wählen Sie „Add Command“.
- ⇨ Der Command Editor öffnet sich. Im nächsten Abschnitt erfahren Sie, wie Sie ein Command erstellen.

Command Editor

Im Command Editor können Sie Befehle für Ihr Script auswählen.

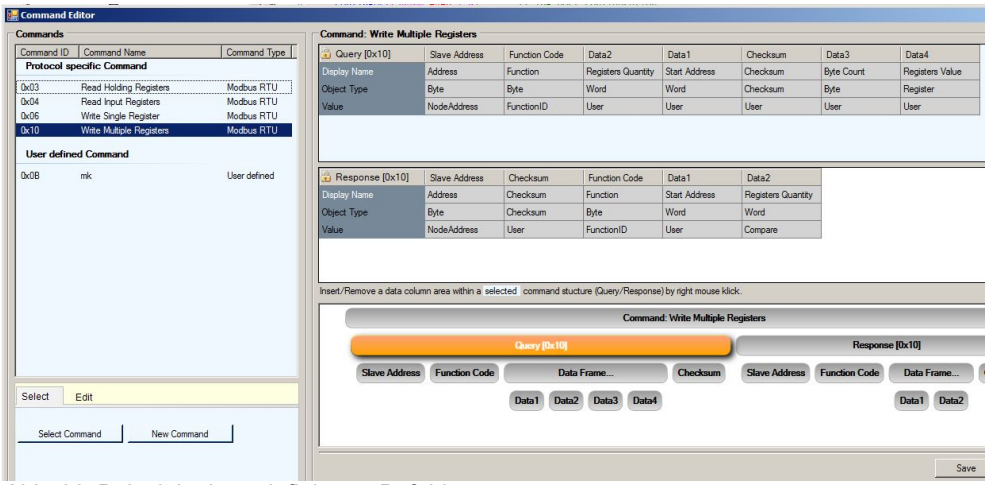


Abb. 22: Beispiel mit vordefinierten Befehlen

Im Fenster „Commands“ finden Sie den Abschnitt „Protocol specific Command“. Die hier aufgelisteten Befehle sind bereits vordefiniert. Sie können je nach verwendetem Feldbus variieren. In diesem Beispiel verwenden wir Modbus RTU als Protokoll.

Info!: Den verwendeten Feldbus können Sie in den Properties, im Feld „FieldbusType“ hinterlegen.

Vordefinierte Befehle

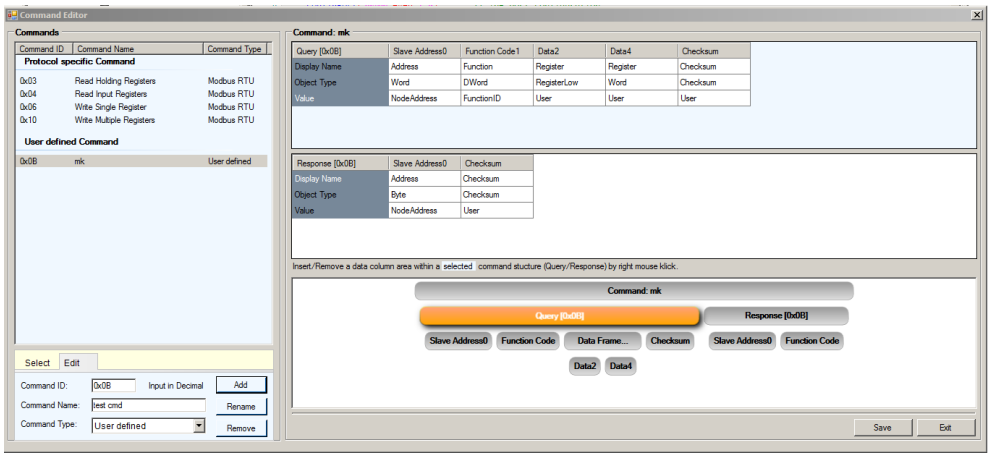
- Wählen Sie den gewünschten Befehl aus dem Abschnitt „Protocol specific Command“

⇨ Im Fenster „Command“ wird das vordefinierte Command angezeigt. Es ist in den Bereich Query und Response unterteilt. Der Bereich Query stellt ein Anfragetelegramm dar, das an das Modul gesendet wird. Der Bereich Response ist das Antworttelegramm, das die Werte vom Modul übermittelt.

Die Commands übernehmen folgende Funktionen:

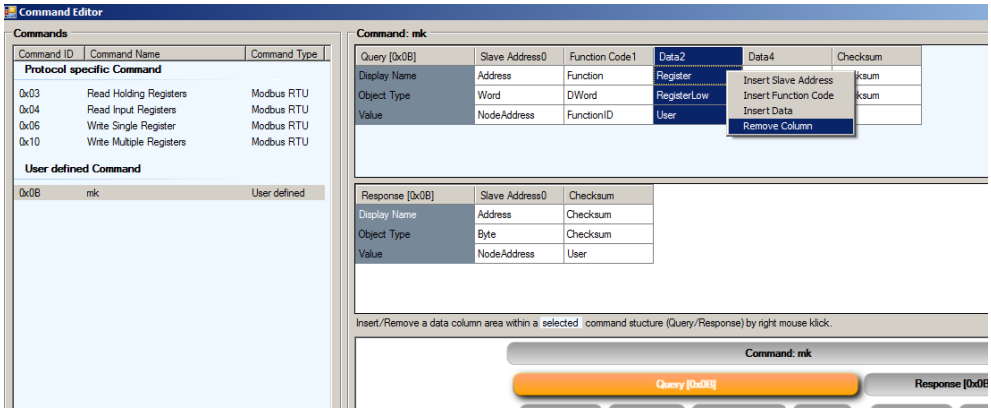
Funktion	Beschreibung
Read Holding Registers	Liest Messwerte
Read Input Registers	Liest Eingangsregister
Write Single Registers	Schreibt auf ein einzelnes Register
Write Multiple Registers	Schreibt auf mehrere Register
Benutzerdefiniert	Benutzerdefiniert

Benutzerdefinierten Befehl erstellen

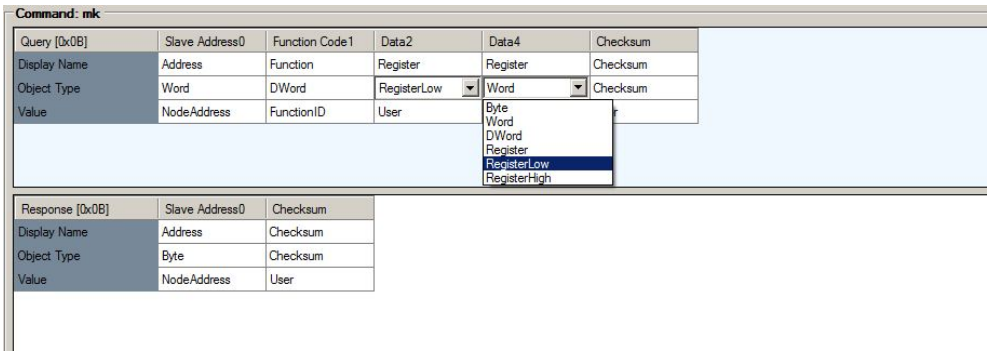


- Wählen Sie die Registerkarte „Edit“ im Fenster „Commands“
- Geben Sie eine neue Command ID ein.
- Geben Sie einen Command Name ein.
- Wählen Sie „User defined“ um ein beliebiges Command zu definieren. Wenn Sie hier "Modbus RTU" wählen, muss das Command mit Slave Adress und Function Code beginnen und mit eine Checksum enden. Die Felder dazwischen können beliebig definiert werden.
- Klicken Sie auf „Add“ um Ihr Script im Fenster „Command“ zu definieren.
  Tipp!: In der Registerkarte Edit können Sie den erstellten Befehl wieder löschen oder umbenennen.
- Fügen Sie im Feld „Command“ die gewünschten Elemente hinzu. Klicken Sie mit der rechten Maustaste in den Bereich „Query“ oder „Response“.

  ⇨ Ein Auswahlfeld mit den verfügbaren Elementen öffnet sich.
- Wählen Sie die gewünschten Elemente aus.



- Definieren Sie die Werte des Ihrer Elemente. Wir unterstützen Sie auch hier durch vordefinierte Auswahlmöglichkeiten.



- Speichern Sie Ihren Befehl mit „Safe“.
- Wählen Sie in der linken Spalte 'Commands' ein Command aus. Mit der Schaltfläche „Select Command“ können Sie das Command in Ihr Script im Hauptfenster einfügen.
- Klicken Sie auf „Exit“ um den Command Editor zu verlassen.

Funktion und gültige Werte der Commands

Command	Wert	Funktion
Display Name	Frei definierbar	Name, mit dem das Feld im Hauptfenster dargestellt wird.
Object Type	Hier können Sie den Typ eines Feldes auswählen.	
	Byte	Das Feld umfasst ein Byte mit einem festen Wert.
	Word	Das Feld umfasst 2 Byte mit einem festen Wert.
	DWord	Das Feld umfasst 4 Byte mit einem festen Wert.
	Register	Das Feld umfasst 2 Byte die aus einem Speicherregister entnommen werden bzw. dort hin geschrieben werden, je nachdem ob das Feld in der Query oder der Response verwendet wird.
	RegisterHigh	Das Feld umfasst 1 Byte wobei nur das Highbyte des Registers verwendet wird.
	RegisterLow	Das Feld umfasst 1 Byte wobei nur das Lowbyte des Registers verwendet wird.
	Checksum	Eine Prüfsumme des Telegramms wird eingefügt bzw. erwartet.

Command	Wert	Funktion
Value	Diese Zeile definiert wo die Werte herkommen bzw. hin geschrieben werden. Hier ist die Auswahlmöglichkeit stark abhängig vom Typ und ob das Feld in der Query oder Response verwendet wird. Möglich sind:	
	User	Sie müssen im Hauptfenster den zu übertragenden Zahlenwert angeben.
	Assign	Der Wert wird einem Register zugewiesen
	NodeAddress	Die Adresse des Geräts wird eingetragen.
	FunctionID	Die Id des Commands wird eingetragen.

Command definieren

Sie sehen Strukturbaum jetzt das erstellte Command mit allen dazugehörigen Feldern. Bei einigen Feldern sind bereits Werte eingegeben. Diese Werte werden automatisch vom Gerät oder vom Command übernommen.

✓ Sie haben im Command Editor ein Command zusammengestellt oder ausgewählt.

◦ Klicken Sie auf ein Feld.

⇨ Im Fenster „Properties“ können Sie jetzt die Werte für dieses Feld definieren.

◦ Prüfen Sie alle Felder und ergänzen Sie die fehlenden Werte.

⇨ Sie haben Ihr Command nun definiert. Im nächsten Schritt können Sie Ihr Script generieren.

Feld	Bedeutung
Address	Adresse des Geräts (Node)
Function	Nummer der ModbusRTU-Funktion
Register Address	Adresse ab der die Register gelesen/geschrieben werden
Register Value	Definition der Quelle bzw. des Ziels aus der bzw. in das die Werte gelesen/geschrieben werden
Checksum	Liefert eine Prüfsumme
Quantity	Anzahl der Register die gelesen bzw. geschrieben werdensollen
Byte Count	Anzahl der Bytes, die gelesen/geschrieben

Tab. 1: Eingabewerte der Command-Felder

Beispiel

✓ Von einem ModbusRTU Server mit der Geräteadresse 7 sollen die Input-Register 4-7 gelesen werden und im Modul in die Speicherregister 0x1410-0x1413 geschrieben werden.

✓ Die Register sollen alle 20 ms gelesen werden.

✓ Im Register 0x1405 soll gezählt werden wie oft die Register gelesen wurden.

✓ Im Register 0x1406 soll aufgetretene Fehler gezählt werden.

◦ Öffnen Sie mit „File/New“ ein neues Projekt.

◦ Weisen Sie folgende Eigenschaften im Fenster „Properties“ zu:
- ErrorCounterAddress: 0x1406
- LoopCounterAdress: 0x1405
-WaitTime: 20 (ms)

◦ Klicken Sie mit der rechten Maustaste auf das Element „Network“.

◦ Wählen Sie im Kontextmenü „Add Node“

⇨ Sie haben jetzt ein neues Gerät hinzugefügt.

◦ Tragen Sie im Feld „Property“ als Adresse „7“ ein.

◦ Klicken Sie mit der rechten Maustaste auf das neu erzeugte Gerät (Node) im Strukturbaum.

◦ Wählen Sie im Kontextmenü „Add Command“.

⇨ Das Fenster „Command Editor öffnet sich.

◦ Klicken Sie auf das vordefinierte Command „Read Input Registers“

◦ Klicken Sie auf „Select Command“

⇨ Das Command erscheint jetzt im Strukturbaum

◦ Prüfen Sie alle Datenfelder auf fehlende Werte. Die Werte Address und Function können Sie nicht Zuweisen. Diese Werte werden automatisch vom Gerät bzw. der gewählten übernommen.

✓ Weisen Sie die Werte der Query im Fenster „Properties“ zu:

◦ Address: 4

◦ Quantity:4

✓ Weisen Sie die Werte der Response im Fenster „Properties“ zu:

◦ Byte Count: 8. Bei Modbus werden 4 Register mit je zwei Byte gelesen. Sie müssen in diesem Feld die Anzahl der Bytes eintragen.

◦ Location: 0x1410

◦ Input Registers und Checksum werden automatisch geliefert, sobald das Script auf dem Modul läuft.

◦ Klicken Sie auf die Schaltfläche „Start Debugging“

⇨ Sie haben ein Script definiert. Es läuft jetzt auf Ihrem Modul.

Script generieren

Um ein Script zu generieren haben Sie 2 Möglichkeiten:

– Wählen Sie in der Menüleiste „Script>Generate Script“

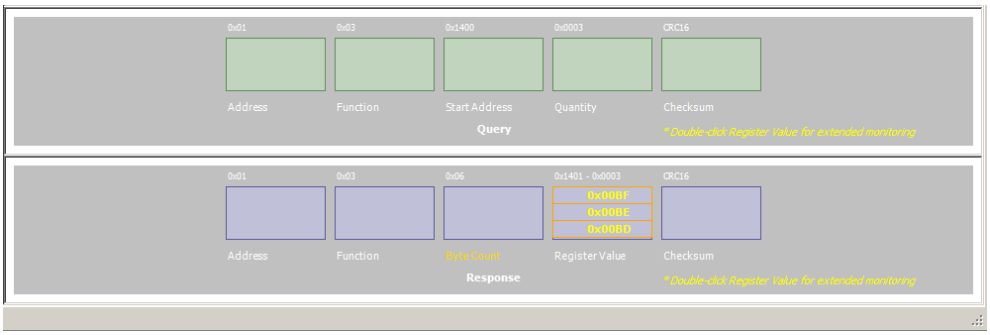
– Klicken Sie in der Symbolleiste auf 

Script überprüfen

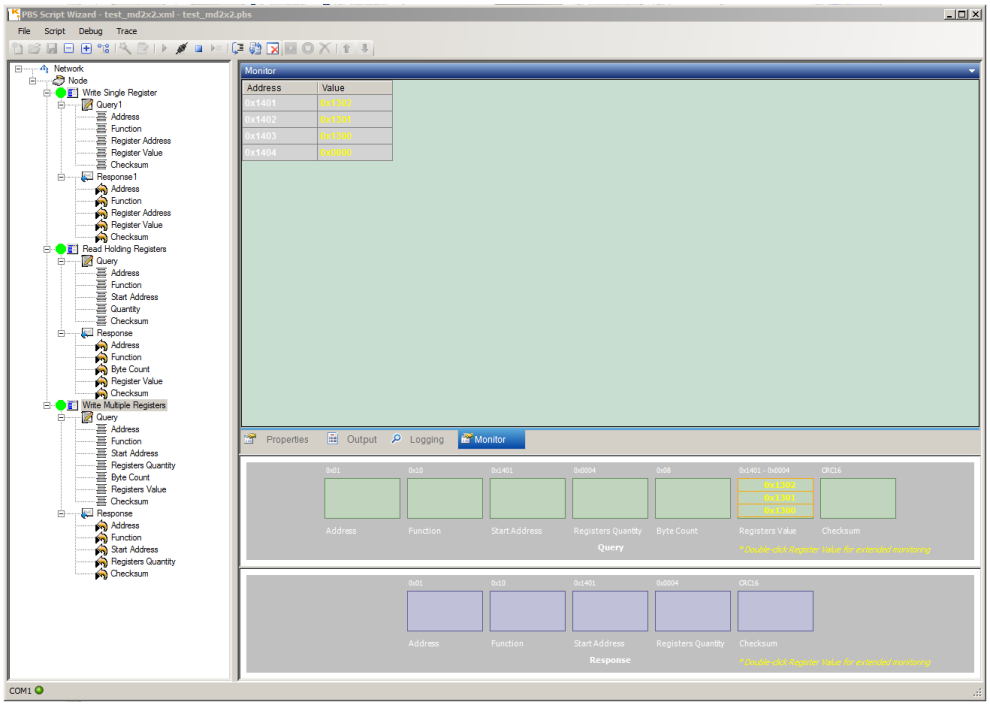
- Wählen Sie im Strukturbaum einen Befehl mit einem Doppelklick aus.

⇨ Folgendes Fenster öffnet sich:

Sie können hier den Scriptablauf in den Speicherregistern verfolgen.



Wählen Sie ein Register mit Doppelklick aus, um den Ablauf in den einzelnen Registern zu verfolgen.



Fehler im Script

Tritt im Scriptablauf ein Fehler auf, signalisiert der Wizard diesen:

– Der betroffene Befehl wird im Strukturbaum durch eine rote Statusanzeige markiert.

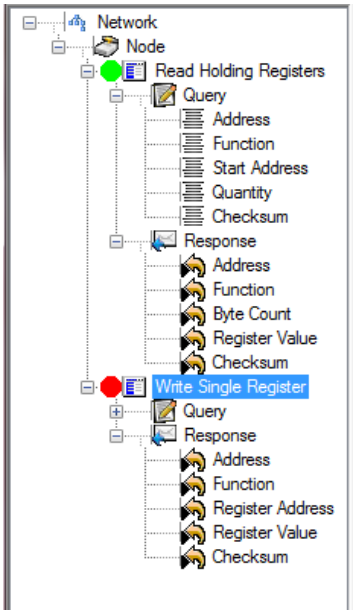


Abb. 23: Fehleranzeige im Strukturbaum

– Der Fehler wird im Trace-Monitor rot hinterlegt und im Nachrichtenfenster gemeldet.

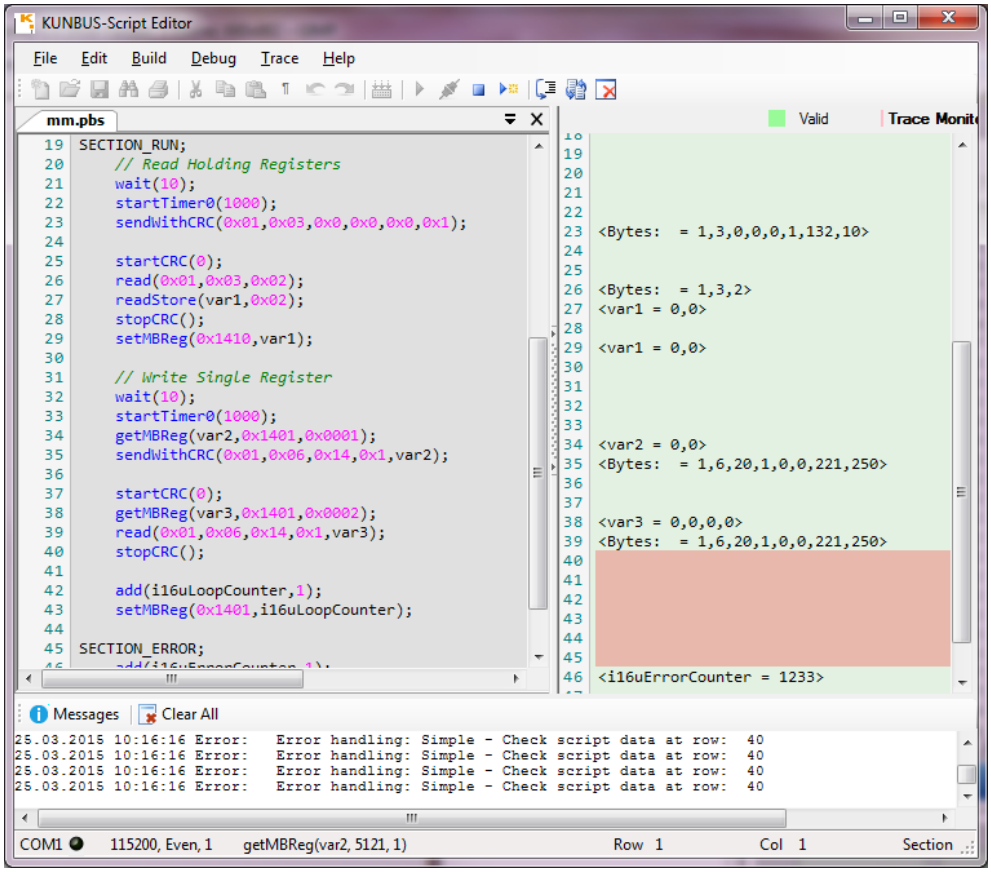


Abb. 24: Fehleranzeige im Trace-Monitor und im Nachrichtenfenster

Tipp!: Wechseln Sie mit  in den Script-Editor. Dort wird das generierte Script angezeigt. Es kann ebenfalls compiliert und auf das Modul gespielt werden. Wenn Sie im Script-Editor die Traceansicht öffnen, wird der Quellcode und bei den ausgeführten Zeilen die Werte der Variablen angezeigt. Code der nicht ausgeführt wird, ist rot markiert. Häufige Fehlerquellen sind Timeouts beim Warten auf die Response und die Verwendung von Registeradressen die es im Modul nicht gibt.

Strukturbaum

Der Strukturbaum ist eine Visualisierung Ihres Scripts. Sie erhalten hier Informationen zu Status, Funktion und Inhalt der einzelnen Befehle. Was angezeigt wird hängt von den Inhalten in Ihrem Script ab.

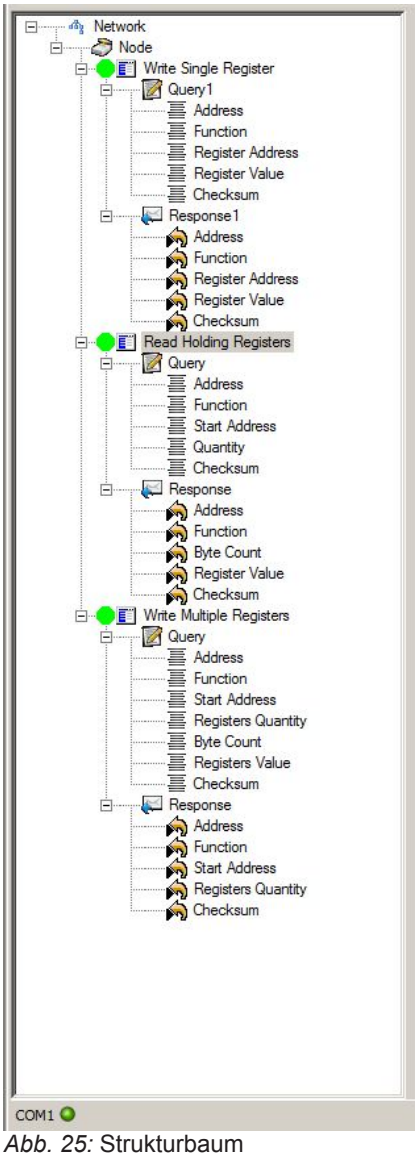


Abb. 25: Strukturbaum

Statusanzeige: Die farbige Statusanzeige vor dem Befehl wird aktiv/ aktualisiert nachdem ein Trace vom Modul geladen wurde. Sie zeigt an, ob der Befehl ausgeführt wird oder ob ein Fehler aufgetreten ist:

Grün	Kein Fehler
Rot	Fehler
Grau	Keine Verbindung zum Netzwerk bzw. keine Information über das Command verfügbar weil vorher ein Fehler auftrat

Inhalte des Befehls: Unterhalb der Funktion des Befehls ist aufgelistet, aus welchen Elementen der Befehl besteht.

Zunächst sehen Sie, ob Ihr Befehl eine Anfrage (Query) an das Modul sendet und eine Antwort (Response) vom Modul erhält.

Die Ebene unterhalb von Query und Response zeigt an, aus welchen Elementen die Anfrage oder die Antwort bestehen.

4 Anhang

4.1 Beispiel-Skript

```
SECTION_CONFIG;
configPort(38400,even,1,8); // BaudRate, Parity, Stop, Data
setBigEndian(); // set the byte-order to Big Endian
decl(var1, 1); // declare var1 as one-Byte variable
decl(var2, 2); // declare var2 as two-Byte variable
decl(var4, 4); // declare var4 as four Byte variable
declArray(var5, 10); // declare var5 as array with the size of 10 bytes
SECTION_INIT;
setMBReg(0x1401, 1); // set register 0x1401 to 1 while the script runs
set(var1, 0x03); //var1 has the value 3
set(var2, 0x0101);
set(var4, 0x12345678);
SECTION_RUN;
wait(100); // wait 100 ms
startTimer0(100); // expect response in 100 ms or error
add(var2, var1); //add var2 and var1 and save the result in var2
minus(var4, var1);
divide(var4, var1);
multiply(var4, var2);
setArray(var5, 3, 4, 250, var1, var1); // var5 = {3, 4, 250, 3, 3}
sendWithCRC(0x11, 0x1201, var2, var5);
// send data 0x11, 0x12, 0x01, high byte of var2, low byte of var2, 3,
4, 250, 3, 3, CRC, CRC
startCRC(0); // start CRC for subsequent bytes
readStore(var5, 0x05); // store 5 bytes in var5
stopCRC(); // read 2 more bytes and compare them to the calculated
CRC
SECTION_ERROR;
setMBReg(0x1401, 2); // set register 0x1401 to 2 if an error occured
SECTION_FATAL;
stop(); // stop
```

4.2 Übersicht der Scriptbefehle

add(*parameter1, parameter2*);

Dieser Befehl zählt den Wert von *parameter2 zu parameter1* und speichert ihn in *parameter1*. Dabei muss *parameter1* eine Variable sein.

configPort(*Baudrate, Parity, StopBit ,DataBit*);

Dieser Befehl konfiguriert die serielle Schnittstelle.

Der erste Parameter definiert die Baudrate. Zulässige Werte sind 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 und 115200.

Die Parität wird als Text definiert. Erlaubt sind ‚even', ‚odd' und ‚none'.

Das dritte Argument definiert die Anzahl der Stoppbits, Wertebereich 0 bis 2.

Das vierte Argument definiert die Anzahl der Datenbits, Wertebereich 7 oder 8.

decl(*name ,size*);

Dieser Befehl deklariert eine Variable mit Name *name* und Größe *size*. Möglich sind 1, 2 und 4 Byte große Variablen. Insgesamt können bis zu 20 Variablen und Arrays in einem Script benutzt werden.

declArray(*name, size*);

Dieser Befehl definiert ein Array Name (*name)* und Größe (*size).*

- Die Größe darf Werte zwischen 1 und 128 haben.
- Alle Arrays zusammen dürfen nicht länger als 256 Bytes sein.

divide(*parameter1, parameter2*);

Dieser Befehl dividiert den Wert von *parameter1 durch den Wert von parameter2.* Das Ergebnis wird in *parameter1* gespeichert. *parameter1* muss eine Variable sein. Falls parameter2 den Wert 0 hat, wird eine Exception ausgelöst.

getMBReg(*value, address*);
getMBReg(*value, address, number*);

Dieser Befehl speichert den Wert des Speicherregisters mit der Adresse *address* in die Variable *value*. Value kann eine Variable oder ein Array sein. Der Paramerter *number* gibt die Anzahl der zu lesenden Register an. Falls er nicht angegeben wird, wird ein Register gelesen.

getMBRegHigh(*value, address*);
getMBRegHigh(*value, address, number*);
getMBRegLow(*value, address*);
getMBRegLow(*value, address, number*);

Dieser Befehl speichert den Wert des Speicherregisters mit der Adresse *address* in die Variable *value*. Value kann eine Variable oder ein Array sein. Der Paramerter *number* gibt die Anzahl der zu lesenden Register an. Falls er nicht angegeben wird, wird nur ein Register gelesen. Aus den Registern wird nur das höherwertige oder das niederwertige Byte gelesen.

If-Ausdruck

If par1 cmp par2 then … endif

If par1 cmp par2 then … else … endif

Die Bedingung wird par1 cmp par2 wird geprüft:

- Wenn die Bedingung wahr ist werden die Kommandos nach then ausgeführt.
- Wenn die Bedingung falsch ist und ein else-Zweig vorhanden ist, werden die Commandos nach else ausgeführt.
- Dann wird das Script nach endif fortgesetzt.

Die Bedingung besteht aus 1 Variablen oder Konstanten mit einem Vergleichsoperator dazwischen. Möglich sind folgende Operatoren:

Operator	Beschreibung	Beispiel für wahr	Beispiel für falsch
==	Gleich	1 == 1	1 == 2
!=	Ungleich	1 != 2	3 != 3
<	Kleiner	1 < 2	2 < 1
<=	Kleiner oder gleich	1 <= 3	3 <= 1
>	Größer	3 > 2	2 > 3
>=	Größer oder gleich	3 >= 3	2 >= 3
bs	Bit gesetzt	0x02 bs 1	0x02 bs 2
bc	Bit nicht gesetzt	0x10 bc 3	0x10 bc 4

bs (bit set) und bc (bit cleared) prüfen ob in einem Wert ein Bit gesetzt ist oder nicht. Der erste Parameter ist der Wert in dem das Bit geprüft wird. Der zweite ist die Bitposition wobei mit 0 das niederwertigste Bit und 31 für das höchstwertigste Bit geprüft wird. Die Zahl 0x10 wäre als Binärzahl 10000b, d.h. nur Bit 4 ist gesetzt. Somit wäre (0x10 bs 4) wahr und (0x10 bc 4) falsch.

incCounter0(*value*); incCounter1(*value*);	Es gibt 2 Zähler: Counter0 und Counter1. Mit den Befehlen resetCounter0 bzw. resetCounter1 können Sie die Zähler auf 0 setzen. Bei jedem Aufruf der Funktionen incCounter0 bzw. incCounter1 wird Zähler erhöht.
minus(*parameter1, parameter2*);	Dieser Befehl subtrahiert den Wert von *parameter2 von parameter1 und speichert ihn in parameter1*. Dabei muss *parameter1* eine Variable sein. Info!: Der Wert von parameter1 muss größer sein, als der von parameter2 damit das Ergebnis nicht negativ ist. Im Falle eines negativen Ergebnisses wird ein Exception ausgelöst.
modulo(*parameter1, parameter2*);	Dieser Befehl dividiert den Wert von *parameter1 durch parameter2. Der Divisionsrest wird in parameter1* gespeichert. *parameter1 muss* eine Variable sein. Falls parameter2 den Wert 0 hat, wird eine Exception ausgelöst.
multiply(*parameter1, parameter2*);	Dieser Befehl multipliziert beide Parameter. Das Ergebnis wird in *parameter1* gespeichert. Dabei muss *parameter1* eine Variable sein.
read (parameter1 , paramter2, …);	Dieser Befehl vergleicht die Anzahl der empfangenen Bytes mit den Werten der Argumente. Bei einer Abweichung der Werte wird eine Exception ausgelöst. **Info!:**Falls vorher startTimer0() aufgerufen wurde und in der angegeben Zeit nicht genügend Bytes empfangen wurden wird ebenfalls eine Exception ausgelöst. **Info!**:Die Funktion empfängt zuerst die angegebene Anzahl Bytes. Danach vergleicht sie die Datenwerte. Beispiel: read(1,2,3); empfängt 3 Bytes und prüft dann, ob das erste Byte 1 ist. Falls ein anderes Verhalten gewünscht wird, muss read mehrfach aufgerufen werden: read(1); read(2); read(3); – Die Funktion darf zwischen 1 und 16 Parametern haben. – Alle Datentypen sind erlaubt.
readSkip(*length*);	Dieser Befehl empfängt *length* Bytes. Diese Bytes werden bei der Prüfsummenberechnung berücksichtig.

readStore(*parameter1, parameter2*);

Mit diesem Befehl hinterlegen Sie in *parameter2* die Anzahl der zu empfangenden Bytes. Diese werden in *parameter1* gespeichert.

– *parameter1 muss* eine Variable oder ein Array sein.

resetCounter0();
resetCounter1();

Siehe incCounter0

send(parameter1, parameter2, …);

Dieser Befehl sendet Daten über die serielle Schnittstelle.

– Die Funktion darf zwischen 1 und 16 Parametern haben.
– Alle Datentypen sind erlaubt.
– Bei 2 oder 4 Byte Konstanten und Variablen werden die Bytes entsprechend der eingestellten Reihenfolgen ausgegeben (setBigEndian() oder setLittleEndian()).
– Bei Arrays werden die Bytes in der Reihenfolge gesendet in der diese im Array stehen.

sendWithBCC(parameter1, parameter2, …);

Dieser Befehl sendet Daten über die serielle Schnittstelle.

– Die Funktion darf zwischen 1 und 16 Parametern haben.
– Alle Datentypen sind erlaubt.
– Bei 2 oder 4 Byte Konstanten und Variablen werden die Bytes entsprechend der eingestellten Reihenfolgen ausgegeben (setBigEndian() oder setLittleEndian()).
– Bei Arrays werden die Bytes in der Reihenfolge gesendet in der diese im Array stehen.

Zusätzlich wird nach den Daten noch eine 1-Byte-Prüfsumme gesendet, die aus den Daten berechnet wird. Die Art der Prüfsumme können Sie durch die Funktion startBCC() festlegen.

sendWithCRC(parameter1, parameter2, …);

Dieser Befehl sendet Daten über die serielle Schnittstelle.

– Die Funktion darf zwischen 1 und 16 Parametern haben.
– Alle Datentypen sind erlaubt.
– Bei 2 oder 4 Byte Konstanten und Variablen werden die Bytes entsprechend der eingestellten Reihenfolgen ausgegeben (setBigEndian() oder setLittleEndian()).
– Bei Arrays werden die Bytes in der Reihenfolge gesendet in der diese im Array stehen.

Zusätzlich wird nach den Daten noch eine 2-Byte-Prüfsumme gesendet, die aus den Daten mit dem CRC-16 Verfahren berechnet wird.

set(*name, value*);

Dieser Befehl weißt der Variable *name* den Wert *value* zu. Die Werte können Konstanten oder Variablen sein.

setArray(*name, parameter1, paramter2, …*);	In das Array *name* werden die Werte der restlichen Parameter geschrieben. Wenn nur ein Parameter vorhanden ist, darf er eine 1, 2 oder 4 Byte Variable oder Konstante sein. Beim Schreiben wird die durch setLittleEndian() bzw. setBigEndian() festgelegte Bytereihen verwendet. Bei mehr als einem Parameter dürfen nur 1 Byte Konstanten oder Variablen angegeben werden.
setBigEndian();	Dieser Befehl legt die Byte-Reihenfolge wird auf Big Endian fest.
setLittleEndian();	Dieser Befehl legt die Byte-Reihenfolge wird auf Little Endian fest.
setMBReg(*address, value*);	Dieser Befehl schreibt den Wert in *value* in das Speicherregister mit der Adresse *address*. Der *value* darf eine Konstante, Variable oder ein Array sein. Die Länge des Array muss ein Vielfaches von 2 sein. **Info!:**Die Speicherregister sind immer zwei Byte groß. Eine vier Byte großen Variable oder ein Array das mehr als zwei Byte enthält, wird in den folgenden Register weiter geschrieben. Die *address* wird dabei hochgezählt.
setMBRegHigh(*address, value*); setMBRegLow(*address, value*);	Dieser Befehl schreibt den Wert in *value* in das Speicherregister mit der Adresse *address*. Bei setMBRegHigh() wird nur das höherwertige Byte beschrieben, das niederwertige Byte bleibt unverändert. Bei setMBRegLow() wird entsprechend nur das niederwerte Byte beschrieben. *value* kann eine Konstante, Variable oder ein Array sein. Bei dieser Funktion wird jeweils ein Byte in die Speicherregister geschrieben. Eine Variable oder ein Array das mehr als zwei Byte enthält, wird in den folgenden Registern weiter geschrieben. Die *address* wird dabei hochgezählt.
startBCC(*mode*);	Dieser Befehl startet eine Prüfsummenberechnung der empfangenen Daten. Dabei spielt es keine Rolle mit welcher read-Funktion die Daten empfangen werden. Mit *mode* wird die Methode der Prüfsummenberechnung ausgewählt: *mode* = 0 -> ZERO_MINUS_SUM Methode: Berechnet die Summe aller empfangenen Bytes und subtrahiert das Ergebnis von 0. *mode* = 1-> BITWISE_NEGATION:

	Berechnet die Summe aller empfangenen Bytes und negiert das Ergebnis bitweise. *mode* = 2-> ADDITION: Berechnet die Summe aller empfangenen Bytes Mode = 3 -> XOR Verknüpft alle Bytes bitweise mit XOR.
startCRC(*mode*);	Mit diesem Befehl startet eine Prüfsummenberechnung auf den empfangenen Daten. Dabei spielt es keine Rolle mit welcher read-Funktion die Daten empfangen werden. Mit *mode* wird die Methode der Prüfsummenberechnung ausgewählt, im Moment ist jedoch nur 0 für eine CRC16 Berechnung zulässig.
startTimer0(*value*);	Dieser Befehl wird verwendet um die Wartezeit einer folgenden read-Funktion zu begrenzen. Es wird ein Timer gestartet der *value* Millisekunden lange läuft. Wenn er abläuft während eine read-Funktion noch auf Daten wartet dann wird ein Fehler ausgelöst und die nächste SECTION_ERROR ausgeführt.
stop();	Dieser Befehl stoppt die Ausführung des Skriptes. Starten Sie das Modul neu um das Skript erneut zu starten.
stopBCC();	**Voraussetzung:** Sie haben eine Prüfsummenberechnung gestartet. Mit diesem Befehl erhält stopBCC mit jeder eingehenden read-Funktion ein weiteres Byte. stopBCC vergleicht dieses Byte mit der berechneten Prüfsumme. Wenn die Prüfsummen nicht übereinstimmen, wird eine Exception ausgelöst.
stopCRC();	Voraussetzung: Sie haben eine Prüfsummenberechnung gestartet. Wenn nach dem Aufruf von StartCRC ein Byte mit einer read-Funktion empfangen wird, wird die Prüfsumme aktualisiert. stopCRC empfängt nun zwei weitere Byte und vergleicht sie mit der berechneten Prüfsumme. Wenn diese nicht übereinstimmen, wird ein Fehler ausgelöst.
wait(*time*);	Dieser Befehl unterbricht den Programmablauf für die angegebene Zeit *time* . Der Wert wird in Millisekunden angegeben.

waitSequence(*parameter1, parameter2, …*);	Dieser Befehl unterbricht den Programmablauf bis die angegebene Byte-Folge empfangen wird. Zulässige Parameter sind Konstanten und Variablen.