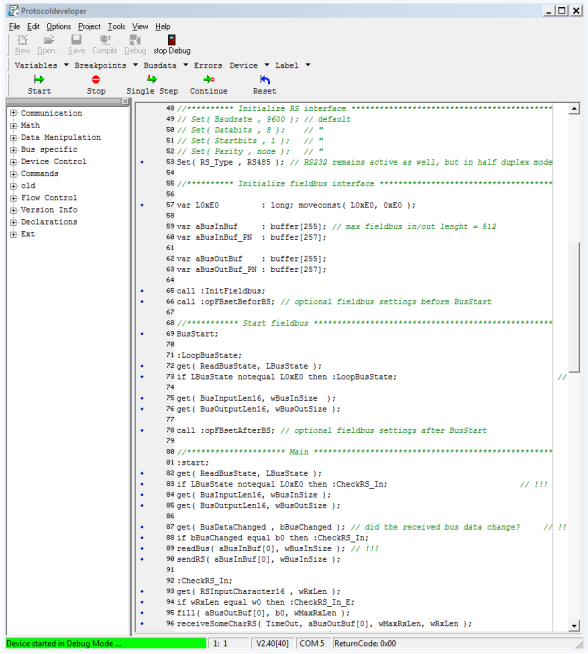
Deutschmann

your ticket to all buses

Bedienerhandbuch
PROTOCOL DEVELOPER



Deutschmann Automation GmbH & Co. KG
www.deutschmann.de | wiki.deutschmann.de

Handbuch Art.-Nr.: V3183

1 Einleitung

Unsere Kunden wollen flexible Lösungen: zu Recht. Das ist Anlaß genug für uns über eine solche Lösung auch auf dem Gateway-Markt nachzudenken.
Wir haben also alle uns bekannten Anforderungen an ein Gateway betrachtet, und aus dieser Obermenge wollten wir eine einfache Lösung aller Probleme schaffen. Relativ schnell wurde uns klar, dass man die Vielzahl der möglichen Anwendungen nicht mehr mit einer reinen Einstellung des Gateways, wie es bereits jetzt mit WINGATE® geschieht ausreicht. Allerdings ist eine Implementierung, die wir für den Kunden vornehmen oftmals zu teuer - bleibt der Weg dem Kunden die Programmierung selbst zu ermöglichen. Würde ein Kunde jetzt anfangen wollen sein C-Programm zu schreiben, hätte er die Probleme Buszugriffe zu programmieren, er bräuchte Kenntnisse der einzelnen Feldbuscontroller etc. Also bieten wir eine Zwischenvariante. Der Kunde braucht nur noch die Daten des Feldbusses weiter zu verarbeiten, und muß sich nicht mehr um Besonderheiten der Busse kümmern.
Für diese Programmierung benötigt er auch keine Kenntnisse von Programmiersprachen, sondern er erzeugt mit Hilfe eines Windows-Tools ein Script.

Allgemeine Grundkenntnisse der Programmierung werden allerdings vorausgesetzt. Beispiele, die in dieser Einleitung gegeben werden, sind Auszüge aus Scripten, die in der abgedruckten Form nicht unbedingt lauffähig sind, da etwaige Vorbedingungen nicht erfüllt sind. Diese Beispiele sind als prinzipielle Anweisungen zu verstehen.

2 Was ist ein Script

Ein Script ist eine Anreihung von Befehlen, die in exakt dieser Reihenfolge ausgeführt werden. Dadurch dass auch Mechanismen gegeben sind, die den Programmfluß im Script kontrollieren, kann man auch komplexere Abläufe aus diesen einfachen Befehlen zusammenbauen.

Das Script ist speicherorientiert. Das bedeutet, dass alle Variablen sich immer auf einen Speicherbereich beziehen. Allerdings brauchen Sie sich beim Entwickeln eines Scripts nicht um die Verwaltung des Speichers zu kümmern; das übernimmt der Protocol Developer für Sie.

2.1 Entscheidung für eine eigene Scriptsprache

Bedingt durch die Hardware kann auf dem Gateway keine bestehende Scriptsprache wie JavaScript, TCL, Perl, Python laufen. Da das Betriebsystem auch kein Microsoft® Betriebsystem ist, ist auch Visual Basic und seine Varianten ausgeschlossen. Noch ein weiterer Punkt spricht gegen diese Sprachen: Keine der oben genannten Sprachen ist auf den 'embedded' Bereich ausgelegt.
All diese Punkte führen zu einer möglichen Lösung:
Eine Sprache,

- die exakt auf den Bereich Gateway / Feldbusse ausgelegt ist
- die Rücksicht auf alle Eigenschaften des Gateways nimmt
- die einfach ist
- die wenig Speicher benötigt
- die effizient vom Gateway auszuführen ist

Diese Punkte zusammengenommen definieren unsere Sprache und begründen zugleich auch die Einschränkungen der Sprache.

2.2 Speichereffizienz der Programme

Ein Scriptbefehl kann z. B. eine komplexe Checksumme wie eine CRC-16 Berechnung über Daten ausführen. Für die Codierung dieses Befehls sind als Speicherbedarf (für den Befehl selbst) lediglich 9 Byte nötig. Dies ist nur möglich, indem diese komplexen Befehle in einer Bibliothek enthalten sind.
Ein weiterer Vorteil dieser Bibliothek ist, dass die zu Grunde liegenden Funktionen bereits seit Jahren im praktischen Einsatz sind und deswegen als fehlerarm bezeichnet werden können. Da diese Befehle auch im für den Controller nativen Code vorhanden sind, ist auch das Laufzeitverhalten des Scripts an dieser Stelle günstig.

3 Hardwarebeschreibung

3.1 UNIGATE® SC in Debug-Ausführung

Prinzipiell ist die Debug-Hardware nicht von einem Standard Gateway unterscheidbar. Zusätzlich zur normalen Hardware ist eine besondere Hardware-Variante verfügbar, die jedoch nur zur Entwicklung eines Scripts benötigt wird. Diese erweiterte Hardware ist technisch bedingt nicht für alle Busse verfügbar; es kann allerdings auch eine Entwicklung auf einer anderen als der Zielhardware erfolgen.
Dieses Debug-Gateway hat gegenüber dem Standard Gateway eine zusätzliche Schnittstelle RS232, die nur in der Ausführung mit dem 9-pol-D-Sub Verbinder nach außen verfügbar ist. Diese DEBUG-Schnittstelle selbst wird immer mit 9600 Baud, no Parity, 8 Datenbits und 1 Stopbit betrieben. Sonst sind keine weiteren Unterschiede vorhanden, weder in der Software noch in der Hardware.

3.2 Debugmodus

Der Debugmodus ist für die Entwicklung eines Scriptes wichtig. In diesem Modus wird das Script nur dann und nur soweit ausgeführt, wie der Benutzer es durch seine Angaben vorgibt. Außerdem kann das laufende Script durch den Benutzer angehalten werden und schrittweise weiter ausgeführt werden, oder auch an einer anderen Position ausgeführt werden. Inhalte von Variablen können beobachtet werden, um so die Abarbeitung von Scriptbefehlen begutachten zu können. Daten, die vom Bus oder der RS-Schnittstelle kommen, können angezeigt werden. Das wichtigste Hilfsmittel dürften der Einzelschrittmodus und die Möglichkeit zum Setzen von Haltepunkten (Breakpoints) sowie die Möglichkeit den aktuellen Speicher (Variablen) auszulesen sein.

3.2.1 Einstellung des Debugmodus

Beim Start des Gateways muß der Schalter Interface (bei UNIGATE SC) auf RS232 gestellt sein. Nach dem Start gibt das Gerät auf der Debug Schnittstelle eine binäre „0“ aus (0x00). Wird diese innerhalb von 0,5 s mit einem O (0x4F) beantwortet, ist das Gerät im Debug Mode.
Die Schalter S4 und S5, sowie der RS-Schalter können auf eine beliebige Stellung gebracht werden, sofern dies notwendig ist.
Für den Anwender selbst sind alle Debug-Befehle in einer komfortablen Oberfläche integriert, was die Entwicklung eines Scripts erleichtert.

3.2.1.1 Einschaltmeldung des Gateways

Wenn sich das Gerät im Konfigmode befindet (d. h. bei UNIGATE IC Config-Jumper gesetzt bzw. bei CL + SC S4 und S5 auf Stellung ”FF”), wird es beim Einschalten auf der RS-232 Schnittstelle (Standardschnittstelle) eine Einschaltmeldung ausgeben.

RS-PB-SC D(232/485) V5.0[6] (c)dA Switch=0xFFFF Script=Leer SN=12345678
Konfigmode...
RS-PB-SC bedeutet, dass es sich bei dem Gerät um ein PROFIBUS Script Gateway handelt.
D bedeutet, der Anschluß ist ein 9-pol DSUB.
(232/485) bedeutet, dass das Gerät über die RS-Schnittstellen RS232 und RS485 verfügt. Alternativ könnte hier bei SC die Bezeichnung (232/422) stehen. Bei CL = (232/422/485), bei IC entfällt die Agabe der Schnittstellen.
V5.0: Software Revision des Gerätes; Firmware ist V5.0
[6]: Script Revision 6
(c)dA Switch=0xFFFF: Copyright Angabe und Schalterstellung aller 4 Drehschalter. Diese Meldung variiert bei den unterschiedlichen Bussen geringfügig. Bei IC entfällt die Schalterstellung.
Script=Leer: im Gerät ist ein Leerscript enthalten. Hier wird die Bezeichnung Ihres Scripts stehen. Die Bezeichnungen müssen im Script am Anfang stehen und sind bis zu 32 Byte lang.

Zusätzlich kann hier auch noch ein Autor und eine Version des Scripts eingefügt werden, die ebenfalls hier angezeigt werden können.

3.3 UNIGATE® IC

Die UNIGATE® IC Serie beinhaltet alle analogen und digitalen Komponenten, die für eine Feldbusimplementierung notwendig sind. Prozessor, Flash-Speicher, RAM, Feldbus ASIC und alle analogen Komponenten sowie der Optokoppler und Spannungsversorgung sind auf der kleinen Fläche vereint.
Auch das IC Gateway wickelt die vollständige Feldbuskommunikation ab. Die serielle Schnittstelle ist nur insofern unterschiedlich zur Schnittstelle des UNIGATE® SC, dass beim UNIGATE® IC keine Treiber angeschlossen sind und die Pegel TTL-Pegel sind.

Bitte beachten Sie, dass die Gateways UNIGATE® CL, IC und SC dieselben Script Funktionen haben, es sei denn es wird ausdrücklich auf Unterschiede hingewiesen. Wird in diesem Handbuch der Begriff „Gateway“ verwendet, meint er sowohl das UNIGATE® CL, UNIGATE® IC als auch das UNIGATE® SC.

Weitere Hinweise zum UNIGATE® IC finden Sie im Handbuch UNIGATE® IC und auf unserer Internetseite http://www.deutschmann.de.

4 Was kann man mit einem Script Gerät machen

Unsere Script Geräte sind in der Lage eine Menge von Befehlen abzuarbeiten. Ein Befehl ist dabei immer eine kleine fest umrissenen Aufgabe. Alle Befehle lassen sich in Klassen oder Gruppen einsortieren. Eine Gruppe von Befehlen beschäftigt sich mit der Kommunikation im allgemeinen, die Befehle dieser Gruppe befähigen das Gateway Daten sowohl auf der seriellen Seite als auch auf der Busseite zu senden und zu empfangen.

Die Befehlsgruppen sind:

Declarations	Variablen Deklaration
Flow Control	Unterfunktionsaufrufe, Sprünge, Verzweigungen
Math	Mathematische Funktionen Datenkonvertierungen
Communication	Senden und empfangen von Daten
Device Control	Parameter setzen und lesen. Exemplarisch sei hier die Baudrate für die serielle Schnittstelle genannt.
Bus specific	Hier sind Befehl angesiedelt, die busspezifische Werte setzen. Wir haben bewußt so wenig wie möglich diese Befehle eingesetzt, damit die Scripte selbst kompatibel bleiben.
Version Info	Texte, die das Gateway in seiner Einschaltmeldung ausgibt. Diese Befehle haben keinen direkten Einfluß auf das Script selbst und sind auch zur Laufzeit nicht von Bedeutung.

Data Manipulation

Bitte beachten Sie, dass an dieser Stelle keine detaillierte Beschreibung der Befehle gegeben wird; die Befehle sind in der Online Hilfe beschrieben.

Die Menge der Aufgaben, die damit bearbeitet werden können, ist schier unendlich.

Es sind Scripte denkbar,
- die immer wieder automatisch Daten von einem Teilnehmer an der seriellen Schnittstelle ermitteln, diese aufbereiten und die aufbereiteten Daten im Bus darstellen
- die nur dann Aktionen ausführen, wenn sich die Busdaten ändern
- die zeitgesteuerte Aktionen ausführen
- die Kommunikationszustände mitteilen
- die Daten zwischen 2 seriellen Teilnehmern (RS485) austauschen und den Zustand im Bus darstellen

Anhand dieser kurzen Aufzählung wird deutlich, dass die Scripte eine flexible Möglichkeit sind, Ihre Probleme zu lösen. Auf beiden Seiten, also sowohl auf der RS Seite als auch auf der Bus-Seite können Daten verarbeitet, konvertiert, angeordnet werden und das Script bietet so prinzipiell die Möglichkeit fast allen Anforderungen gerecht zu werden.
Probleme sind eigentlich nur an wenigen Punkten zu erwarten:
- Ist ihre Anforderung extrem zeitkritisch? Dadurch, dass das Script interpretiert wird, ist das Laufzeitverhalten nicht ganz so günstig wie eine direkte Implementierung.
- Je nach Protokoll das abgewickelt werden muß, kann jedoch auch mit dem Script eine Reaktionszeit von wenigen Millisekunden erreicht werden, was für die allermeisten Anwendungen vollkommen ausreicht.

4.1 Unabhängigkeit von Bussen

Prinzipiell sind die Scripte nicht vom Bus abhängig, auf dem sie arbeiten sollen, d. h. ein Script, das auf einem PROFIBUS Gateway entwickelt wurde, wird ohne Änderung auch auf einem Interbus Gateway laufen, da sich diese Busse von der Arbeitsweise sehr stark ähneln. Um dieses Script auch auf einem Ethernet Gateway abzuarbeiten, muß man evtl. noch weitere Einstellungen im Script treffen, damit das Script sinnvoll ausgeführt werden kann.
Es gibt keine festen Regeln, welche Scripte wie richtig arbeiten. Beim Schreiben eines Scripts sollten Sie beachten, auf welcher Zielhardware Sie das Script ausführen wollen, um die nötigen Einstellungen für die jeweiligen Busse zu treffen.

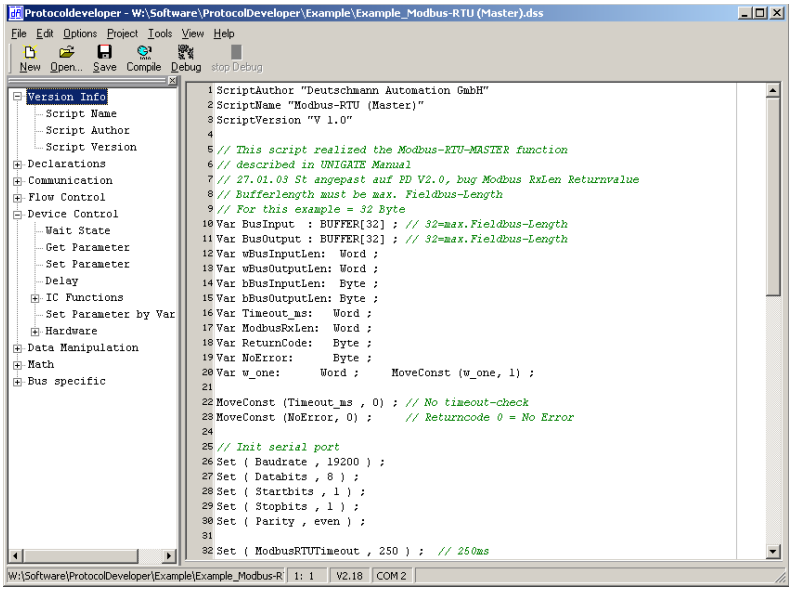
4.2 Weitere Einstellungen am Gateway

Die meisten Geräte benötigen keine weiteren Einstellungen außer denen, die im Script selbst getroffen sind. Allerdings gibt es auch Ausnahmen hierzu. Diese Einstellungen werden mit der Software WINGATE getroffen. Wenn Sie bereits unsere Serie UNIGATE® kennen, wird Ihnen die Vorgehensweise hierbei bereits bekannt sein. Beispielhaft sei hier die Einstellung der IP-Adresse und der Net-Mask eines Ethernet-Gateways genannt. Diese Werte müssen fix bekannt sein und sind auch zur Laufzeit nicht vorhanden. Ein weiterer Grund für die Konfiguration dieser Werte in WINGATE ist folgender: Nach einem Update des Scripts bleiben diese Werte unangetastet, d. h. die einmal getroffenen Einstellungen sind auch nach einer Änderung des Scripts weiterhin vorhanden.
Nur so ist es auch möglich, daß das gleiche Script auf verschiedenen Ethernet-Gateways arbeitet, die alle eine unterschiedliche IP-Adresse haben.

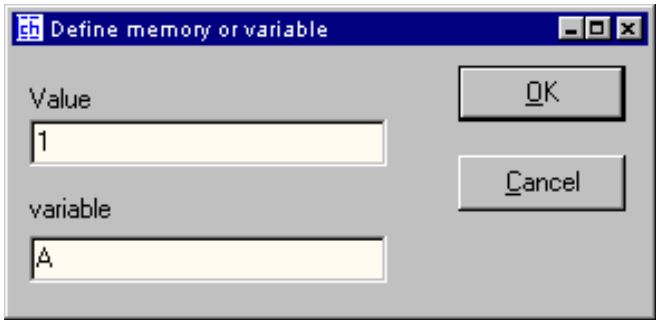
4.3 Die Benutzung des Protocol Developers

Der Protocol Developer ist als Werkzeug zum einfachen Erstellen eines Scripts für unsere Script Gateways gedacht; seine Bedienung ist genau darauf ausgerichtet. Nach dem Start des Programms wird das zuletzt geladene Script erneut geladen, sofern es nicht der erste Start ist.
Windows typisch können Script Befehle per Maus oder Tastatur hinzugefügt werden. Soweit für den entsprechenden Befehl definiert und notwendig wird der Dialog zu dem entsprechenden Befehl angezeigt, und nach dem Eingeben der Werte wird automatisch der richtige Text in das Script eingefügt. Das Einfügen von neuen Befehlen durch den Protocol Developer erfolgt so, dass niemals ein existierender Befehl überschrieben wird. Generell wird ein neuer Befehl vor dem eingefügt, auf dem momentan der Cursor positioniert ist. Selbstverständlich können die Befehle auch einfach per Tastatur geschrieben werden, oder bereits geschriebene Befehle bearbeitet werden.

4.3.1 Das Hauptfenster



Im Hauptfenster findet das eigentliche Schreiben eines Scripts statt. Aus der Liste im linken Teil des Fensters können Sie mit der Maus ein Kommando "nehmen" und in das Script auf der rechten Seite einfügen. Steht für den jeweiligen Befehl ein Dialog zur Verfügung, wird er vor dem Einfügen des Textes aufgerufen. Sie können jetzt Variablennamen etc. eintragen, die dann im Script selbst zur Verfügung stehen. An dieser Stelle brauchen Sie sich nicht um die Schreibweise von Script Befehlen kümmern, der Protocol Developer unterstützt sie hier.
Beispiel für einen Dialog zu einem Befehl.



Nach Bestätigung des Dialogs mit OK erzeugt der Protocol Developer den Code für diesen Befehl, im Beispiel wäre das der Code "MoveConst (A,1)". Sie können den Code nun auch manuell weiterbearbeiten.

4.4 Menüstruktur

Menu File

Im File Menü finden sich alle Menüpunkte, die für die Dateien notwendig sind.

New

Mit File New wird ein neues Editorfile erzeugt. Das neue File enthält kein Script und ist vollkommen leer. Wird ein solches File kompiliert wird lediglich der Grundcode für ein Script erzeugt.

Open

Mit File open wird eine bestehende Datei geöffnet. Die Datei muß bereits existieren. Eine neue Datei ist per File New zu erzeugen.

Save

Mit File Save wird eine Datei auf einem Datenträger gespeichert. Hat eine Datei noch keinen Namen, da sie mit File New erzeugt wurde, wird automatisch der File Save as Dialog geöffnet.

Save as

Mit diesem Menüpunkt kann eine Datei unter einem anderen Namen als dem bisher vergebenen Namen gespeichert werden.

Save compiled file

Die Datei liegt als Quellcode (Sourcecode) vor. Mit Save compiled File wird der Quellcode erneut kompiliert, und wenn das File erfolgreich kompiliert wurde, als binäres File gespeichert. Dieses File kann mit anderen Tools (WINGATE oder SPT- ScriptProgramTool) an ein Script Gateway übertragen werden.

Print

Mit Print kann der gesamte Quellcode eines Programmes gedruckt werden.
Es kann z. Zt keine Einschränkung vorgenommen werden, welche Seiten gedruckt werden.

Exit

Mit Exit wird der Protocol Developer verlassen. Ist das aktuelle File nicht gespeichert, wird zum speichern des Files aufgefordert.

Menu Options
Settings

Mit Settings können die zu Grunde liegenden Einstellungen des Protocol Developers eingestellt werden. Hierzu zählt z. B. die Einstellung der seriellen Schnittstelle.
Außerdem sind hier auch alle Dateien angelegt, die Dialoge, Befehle etc. enthalten. Sie sollten diese Liste jedoch niemals unaufgefordert bearbeiten.

Menu Project
Compile

Das aktuelle Script, das sich im Protocol Developer befindet wird übersetzt. Im Falle eines Fehlers wird der Compiler anzeigen, wo er den Fehler erkannt hat und von welcher Art der Fehler ist. Sie können dann im Editor diese Stelle gezielt bearbeiten.

Debug

Hiermit wird der Debugger gestartet. Allerdings muß hierzu das Script syntaktisch in Ordnung sein und es muß ein Gateway im Debugmode an der Schnittstelle angeschlossen sein. Siehe hierzu auch Kapitel Debugging.

Menu View
Editor

Der Editor wird angezeigt.

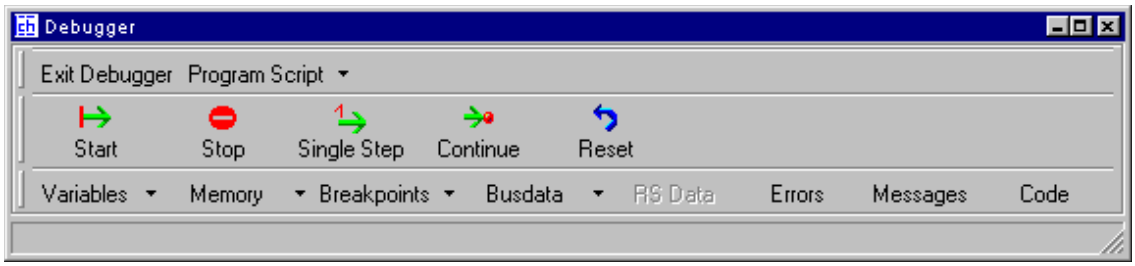
Internals

Mit Internals kann man sich die internen Werte des Compilers ansehen.

Help

Wenn sich die Hilfedatei in dem Verzeichnis befindet, in dem auch der Protocol Developer enthalten ist, wird die Startseite der Hilfe angezeigt.

4.5 Der Debugger



Das Hauptfenster des Debuggers ermöglicht die Steuerung eines Gateways, das sich im Debugmode befindet. Das Fenster bietet die Schnittstelle zur Bedienung und Steuerung des Debuggers bzw. des Debug-Gateways.

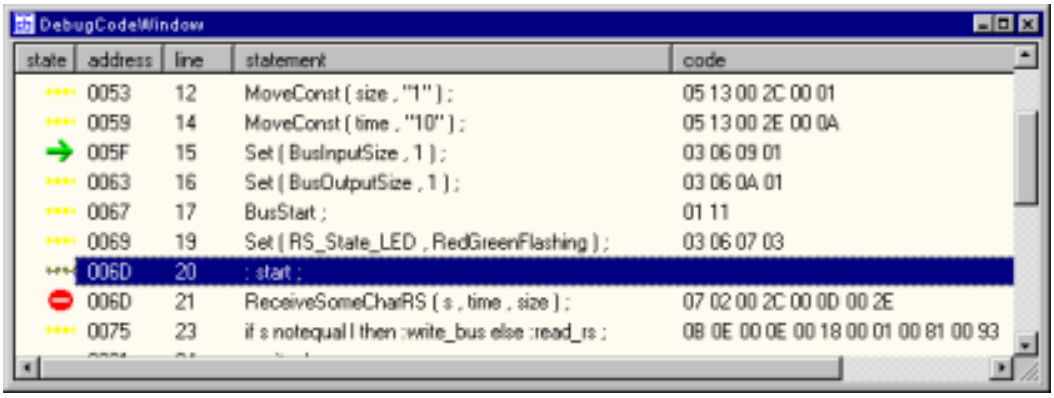
Exit Debugger

Das Debugger-Fenster wird geschlossen.

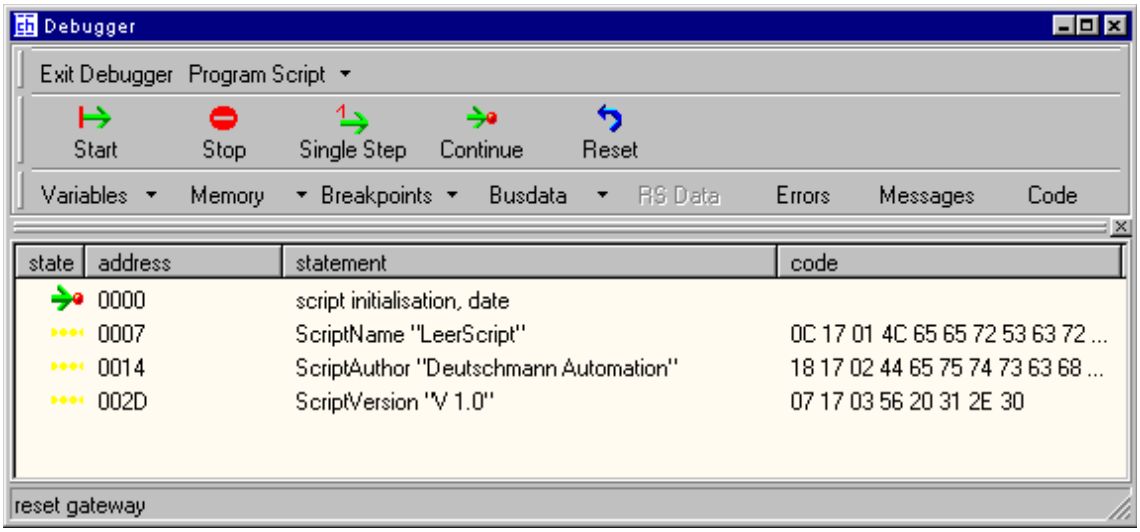
Program Script

Das momentan sich im Debugger befindliche Script wird fest in das Gateway (UNIGATE® IC, UNIGATE® SC) programmiert, so dass es auch nach einem Neustart des Gateways noch vorhanden ist. Der Debugger kann nicht feststellen, ob bereits ein Script programmiert ist und wird jedes Mal, wenn er einen Neustart des Debug-Gateways feststellt das zuletzt bearbeitete Script erneut an das Gerät senden. Mit dieser Vorgehensweise ist sichergestellt, dass sich im Debug-Gateway immer das Script aus dem Code-Fenster befindet.

Das Code-Fenster



Im Code-Fenster wird das Script selbst in einer vom Compiler überarbeiteten Variante angezeigt. Im Code-Fenster können Sie Haltepunkte setzen, Scripte starten etc. Hier verfolgen Sie den Ablauf Ihres Scripts und beobachten das Verhalten des Scripts und Ihrer Anwendung um eventuelle logische Fehler im Script zu finden und zu beheben.
Normalerweise ist das Fenster im Hauptfenster angedockt, es kann aber auch aus diesem gelöst werden.



Starten eines Scripts

Nachdem der Debugger gestartet ist, hat das Script normalerweise die Ausführungsadresse 0000, es steht ganz am Anfang. Ein grüner Pfeil in der Spalte state zeigt Ihnen die aktuelle Position an. Mit "Start" können sie nun das Script selbst ausführen. Sobald das Gateway läuft und die Abarbeitung des Scripts gestartet hat werden die Schalter für "Single Step" und "Continue" ausgeblendet.

Anhalten eines Scripts

Um das Script anzuhalten müssen sie warten, bis das Gateway auf einen Haltepunkt stößt und damit die weitere Abarbeitung von Scriptkommandos stoppt, oder sie können auch mit der "Stop" Taste das Script anhalten. Das Gateway wird nicht angehalten, wenn zur Zeit ein Befehl ausgeführt wird, der längere Zeit in Anspruch nimmt. Der aktuelle Befehl wird komplett ausgeführt. Das scheint dann so, als ob der Debugger nicht mehr läuft, oder das Gateway nicht mehr reagiert.
Beispiel: der Befehl "delay(10000);" wartet 10 Sekunden. Wenn Sie diesen Befehl per "Stop" unterbrechen, wird auf jeden Fall der Befehl noch zu Ende ausgeführt, d. h. die Kontrolle geht erst nach Ablauf der 10 Sekunden an den Debugger über. Beachten Sie, dass es auch Befehle gibt, die keine feste Laufzeit haben. Diese Befehle sind nicht durch Stop unterbrechbar.

Reset eines Gateways

Mit Reset wird das Gateway in den Urzustand gebracht, in dem es sich auch nach dem Einschalten befindet. Der gesamte Speicher des Gerätes ist mit 0 vorbelegt, die Ausführungsposition für Script-Befehle ist Adresse 0000.

Setzen eines Haltepunktes

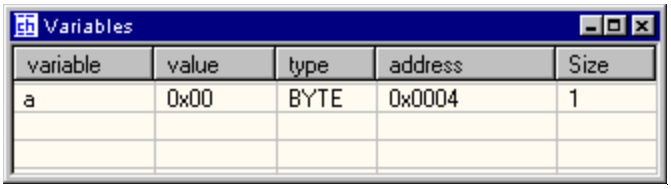
Gehen Sie im Code-Fenster auf die Zeile, vor der die Ausführung des Scripts anhalten soll. Sie können nun über das kontextsensitive Menü (rechte Maustaste) einen Haltepunkt ein- oder ausschalten. Beachten Sie, dass nach einem Reset alle Haltepunkte gelöscht sind.

Verändern der Ausführungsposition

Klicken Sie im Code-Fenster auf die Zeile, an der Sie die Ausführung des Codes fortsetzen wollen. Über das kontextsensitive Menü können Sie nun den "ProgramCounter" auf diese Position setzen. Sie sollten diese Vorgehensweise nur dann machen, wenn Sie alle Nebeneffekte, die hierdurch auftreten können im Blick behalten. Dieser Befehl ist dann nützlich, wenn Sie z. B. ohne Bus entwickeln und den Befehl "wait(Bus_Active)" übergehen wollen.

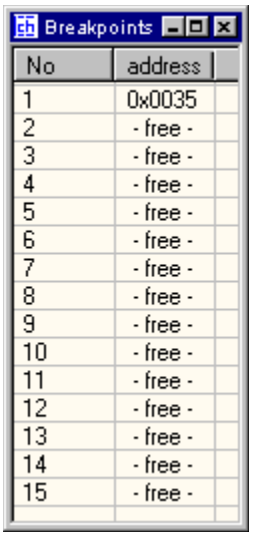
Variablenanzeige

Aus dem Debuggerfenster kann die Anzeige des Fensters selbst gesteuert werden. Es können Variablen angezeigt und die Darstellungsart dieser Variable kann eingestellt werden. Eine Variable kann wieder aus der Anzeige entfernt werden. Es können auch alle Variablen gelöscht oder alle Variablen eines Scripts angezeigt werden.
Außerdem besteht die Möglichkeit alle Werte der Variablen manuell zu aktualisieren. Normalerweise werden alle Variablen nach einem Stop des Gateways automatisch aktualisiert.
Um Variablen hinzuzufügen können Sie die Taste im Debugger benutzen oder das kontextsensitive Menü des Variablenfensters. Es können nur die Variablen hinzugefügt werden, die auch im aktuellen Script enthalten sind. Sind Variablen aus früheren Debug-Sitzungen enthalten, wird nur der Name ohne Wert angezeigt. Diese Variablen sollten entfernt werden.
Mit einem Doppelklick bearbeiten Sie die Variable. Sie können jetzt die Darstellungsart für die Variable ändern.



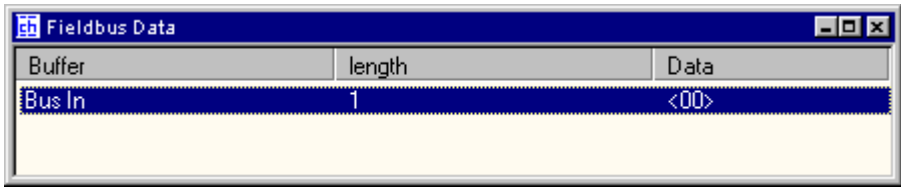
Breakpoints

Das Script Gateway kann für die Entwicklung eines Scripts bis zu 15 Breakpoints verwalten. Ein Breakpoint wird auf eine Zeile im Code gesetzt, indem die Markierung auf die entsprechende Zeile bewegt wird (oder mit der Maus auf diese Zeile geklickt wird) und entweder über das Menü, der rechten Maustaste oder mit der Taste F5 umgeschaltet wird. Ein Breakpoint kann nur dann gesetzt werden, wenn das Gateway sich im Stop Zustand befindet. Führt das Gateway gerade ein Script aus ist das Setzen eines Breakpoints ohne Funktion.
Eine Liste der aktiven Breakpoints kann über das Debugger Menü aufgerufen werden.
Ist ein Breakpoint im Script gesetzt und kommt die Ausführung des Scripts auf einen Breakpoint, so wird VOR dem Ausführen des entsprechenden Befehls angehalten. Mit Single Step kann diese eine Zeile ausgeführt werden, mit Go kann das Script ab der Halteposition weiter ausgeführt werden.



Bus Daten Fenster

Hier können die aktuellen Busdaten beobachtet werden. Diese Daten werden nach jedem Stop Befehl aktualisiert. Sie können auch manuell aktualisiert werden. Die Daten stehen allerdings immer erst NACH der Ausführung des ersten Scriptbefehls zur Verfügung. Für eine sinnvolle Nutzung dieses Fensters ist es zwingend erforderlich, daß der Bus in der Art in Betrieb ist wie er auch später eingesetzt werden soll.

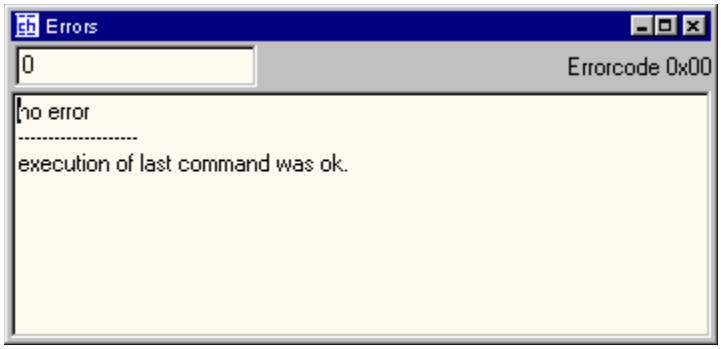


Error Fenster

Jeder Script Befehl wird vom Gateway einzeln ausgeführt. Bei der Ausführung selbst kann eine Konstellation auftreten, dass der Befehl zwar ausgeführt wurde, aber trotzdem eine Fehlerbedingung erreicht wurde.
Beispiel: Empfang von seriellen Daten mit Timeout, dann löst Timeout einen Fehler aus, der aber nicht zum Abbruch des Scripts führt. Diese Bedingung muß vom Script weiter ausgewertet werden.
Mit jedem Stop des Gateways im Debug-Modus wird der letzte Fehlerzustand des Geräts an den PC übertragen. Das Gateway führt einen Stop aus:

- nach einem Single Step.
- nach einem Neustart.
- mit dem Erreichen eines Breaks.

Im Fenster wird der Fehlercode und eine kurze Beschreibung zu diesem Fehler angezeigt.
In der Online-Hilfe sind die Fehlercodes ebenfalls enthalten.



4.6 Programmierung von Scripten

4.6.1 Schreibweise von Script Befehlen

Es wird nicht zwischen Groß- und Kleinschreibung unterschieden.

4.6.1.1 Zahlen

Es gibt verschiedene Möglichkeiten Zahlen zu schreiben. Es gibt die dezimale, binäre und hexadezimale Notation. Außerdem kann eine explizite Typumwandlung vorgenommen werden, wenn eine Konstante in ein spezielles Format gewandelt werden muß.

Beispiel:

```
var b: word;                    // B belegt 2 Byte im Speicher
MoveConst ( B, 257 ) ;          // B hat den Wert 0101
MoveConst ( B, 0x110 ) ;        // B hat den Wert 0110
MoveConst ( B, "AA" ) ;         // B hat den Wert 4040
MoveConst ( B, #0x01#13 ) ;     // B hat den Wert 010D
MoveConst ( B, "A"#13 ) ;       // B hat den Wert 400D
MoveConst ( B, 0b1111 ) ;       // B hat den Wert 000F
```

4.6.1.2 Texte

Texte werden generell in doppelten Anführungszeichen angegeben. Es ist auch möglich, Sonderzeichen in den Text einzubinden. Es werden zur Deklarationszeit Längenprüfungen durchgeführt, das bedeutet, dass es nicht möglich ist, einen 30-stelligen Text an eine Variable mit nur 10 Zeichen Länge zuzuweisen. Binäre Zeichen können durch die Angabe des dezimalen Codes an eine String-Konstante angehängt werden.
Anders als z. B. in Pascal sind jedoch keine Längen in den Texten selbst gespeichert, da die Länge durch das Script selbst berechnet werden muß.

4.6.1.3 Kommentare

Es können an jeder beliebigen Stelle im Script Kommentare eingefügt werden. Kommentare werden immer mit einem doppelten Schrägstrich eingeleitet und gehen grundsätzlich bis zum Ende einer Zeile. Es besteht keine andere Möglichkeit, dass sich Kommentare über mehrere Zeilen erstrecken, als dass jede Zeile neu als Kommentar eingeleitet wird.

4.6.1.4 Label

Label können grundsätzlich alle Bezeichner sein, die gültig aufgebaut sind. Es gibt keine Längenbeschränkung, jedoch werden Labels nur mit 255 Stellen signifikant behandelt, das bedeutet, dass Label mit mehr als 255 Zeichen, die sich in den ersten 255 Zeichen gleichen als identisch betrachtet werden. Das erste Zeichen eines Labels ist der Doppelpunkt gefolgt von beliebigen alphanumerischen Zeichen und dem Unterstrich.

4.7 Hinweise zur Scriptentwicklung

Stellen sie einige Vorüberlegungen an:
Ist Ihr Script nur für einen Bus oder soll es für mehrere Busse arbeiten?
Sind die Busse vom Prinzip ähnlich oder gegensätzlich?

Wenn Ihr Script nur für einen Bus gedacht ist, können Sie alle Eigenschaften dieses Busses ausnutzen. Wenn Ihr Script mehrere Busse unterstützen soll, müssen Sie einiges beachten:
Welcher Bus, der unterstützt werden soll, hat die kleinste Datenbreite? Nehmen Sie diese Breite als Basis und bauen Sie die Busdaten so auf, dass alle Busse hiermit arbeiten können. Sie ersparen sich hiermit die Unterscheidung der Scripte in Bus-spezifische Teile.

Sind die Busse vom Prinzip ähnlich oder gegensätzlich?
Als ähnliche Busse würde man hier Busse bezeichnen, die z. B. Single-Master Busse sind bei denen der Master die Daten zyklisch überträgt. Hierzu zählt man z. B. PROFIBUS, Interbus, eingeschränkt auch DeviceNet (im poll-Betrieb).
Als unähnliche Busse können hier z. B. Ethernet und CANopen® genannt werden.

Beschreiben Sie die Aufgabe des Scripts:
Nur mit einer klar umrissenen Aufgabe können Sie ein Script erfolgreich umsetzen.

Erstellen Sie sich jetzt eine Struktur in irgendeiner Ihnen geläufigen Form, egal ob eine grafische Struktur oder eine verbale Beschreibung. Solche Abläufe sind sinnvoll, um logische Fehler in Scripten zu finden.

Beginnen Sie erst jetzt mit der Codierung der einzelnen Aufgaben. Trennen Sie dabei wenn möglich einige Aufgaben in kleinere Einheiten auf um sie übersichtlich zu halten und sie auch entsprechend testen zu können.

Testen Sie das Script immer stückweise, merken oder notieren Sie sich, welche Eigenschaften Unterfunktionen haben.

4.8 Besondere Regeln für Scripte

Im Script müssen die Befehle "ScriptName", "ScriptAuthor" und "ScriptVersion" als erste Befehle eingetragen sein, damit sie in der Einschaltmeldung des Geräts erscheinen. Das Script-Datum ist das Übersetzungsdatum des Scripts und wird automatisch eingetragen. Sind die Befehle an beliebiger anderer Stelle, werden Sie bei der Initialisierung des Gateways nicht erkannt, führen aber auch nicht zu einer Fehlfunktion des Scripts selbst.
Die Reihenfolge der Befehle selbst spielt keine Rolle; es kann auch nur ein Befehl aus dieser Gruppe benutzt werden.

4.9 Debugging

Mit Debugging wird in der Programmierung allgemein die Fehlersuche in einem Projekt bezeichnet. Ein wenig eingeschränkt durch die Hardware, auf der das Script ausgeführt wird, ist das Debugging auf dem Gateway zu sehen. Es bietet dennoch alle nötigen Eigenschaften um Fehler logischer Struktur zu erkennen. Allerdings sollten Sie sich darüber im klaren sein, dass der Protocol Developer nicht automatisch Fehler finden kann, er bietet nur das Werkzeug dazu. Die genaue Analyse dessen, was das Script tut müssen weiterhin sie machen.
Der Protocol Developer kann Sie an einigen Punkten unterstützen.

4.9.1 Vorgehensweise

Sie müssen als erstes ein Script entwickeln, oder sofern Sie sich mit den Funktionen des Protocol Developers vertraut machen wollen, ein bestehendes Script in die Entwicklungsumgebung laden. Es dürfen keine syntaktischen Fehler mehr im Script vorhanden sein.
Ein entwicklungsfähiges Gateway muß angeschlossen sein und sich im Debugmodus befinden (siehe hierzu auch das Kapitel Hardware).
Das Script muß gespeichert sein.
Starten Sie den Debugger über das Menü oder den Button. Der Protocol Developer wird jetzt versuchen das Gateway zu erkennen, das Script zu übersetzen und an das Gateway zu übertragen. Erst wenn diese Schritte erfolgreich waren wird der Debugger selbst gestartet.

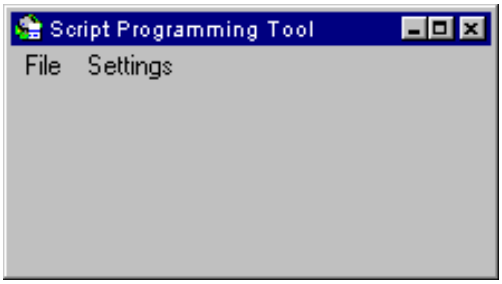
4.9.2 Debugbefehle

Es stehen eine Menge von Debug-Befehlen zur Verfügung, die über die Diagnose-Schnittstelle des Gateways ausgeführt werden können. Die Befehle sind so ausgelegt, dass sie von einem beliebigen Terminal ausgeführt werden können. Normalerweise ist die Bedienung "von Hand" nicht nötig, da diese Befehle vom Debugger des Protocol Developer angewandt werden. Der Debugger führt außerdem noch weitere Aktionen durch, die eine Fehlersuche in einem Programm erheblich erleichtern, wie z. B. das direkte Beobachten von Variablen in verschiedenen Anzeigeformaten, das leichte Verändern der momentanen Ausführungsposition.
Zusätzlich sichert die Umgebung noch zu, dass man immer die aktuell gültige Zeile Code sieht (Kommentare sind hier nicht sichtbar und Befehle, die sich im Originalcode über mehrere Zeilen erstreckten werden hier auf eine Zeile reduziert. Sprungziele werden angezeigt). Der vom Compiler erzeugte Code ist zu sehen, so dass auch Speichergrenzen etc. aufgedeckt werden können.

5 Beschreibung Script Program Tool

Dieses Tool ist entwickelt worden, um ein Script, das in der übersetzten Form vorliegt in ein scriptfähiges Gateway zu programmieren. Ein Script in der übersetzten Form kann mit dem Protocol Developer erzeugt werden.

Prinzipiell stehen 2 Ausführungsvarianten zur Verfügung.



5.1 Manueller Modus

Das Programm wird aufgerufen
Aus dem Menü heraus wird eine Datei (ein übersetztes Script) geladen und an das Gerät gesendet (Per Dialog File...).
Das Programm wird beendet (File Exit).
Einstellungen können getroffen werden (COM 1.. COM 4).

5.2 Automatischer Modus

Im automatischen Modus wird dem Programm direkt der Dateiname des übersetzten Scripts mitgeliefert, das programmiert werden soll.
Es wird versucht, das Programm zu starten. Beim ersten Start wird zusätzlich interaktiv die serielle Schnittstelle ermittelt, an der das Scriptgateway angeschlossen ist.
Mit dem Programmstart selbst wird automatisch versucht diese Datei zu laden.
Danach wird das Programm versuchen, das Script an das Gateway zu senden. Wenn diese Operation durchgeführt werden kann wird eine Fortschrittsanzeige angezeigt, ansonsten ein Fehler.
Das Programm wartet, bis das Gateway nach dem Schreiben des Scripts neu gestartet ist und beendet sich dann selbständig. Zum Programmieren ist also nicht mehr nötig als ein Klick.

5.2.1 Einrichten des automatischen Modus

Zum Einrichten des automatischen Modus muß im Explorer ein "Link" erzeugt werden (rechte Maustaste, New -> Link). Dann muß der Link selbst markiert werden (Klick mit der Maus).
Die Eigenschaften des Links müssen angepaßt werden (rechte Maustaste, Edit Properties).
In der Kommandozeile ist der Dateiname als Parameter einzurichten.
Beispiel: (SPT.EXE "FileName.gws.gwc")
Dialog bestätigen, fertig.

6 Anhang

Auflistung aller Debug Befehle in alphabetischer Reihenfolge

Break

Mit Break wird ein Breakpoint gelöscht oder gesetzt. Setzen eines Breakpoints auf eine Adresse: B030020: der Breakpoint 03 wird auf die Adresse 0020 gesetzt, gleichgültig, ob der Breakpoint vorher existierte oder nicht.
B030000: der Breakpoint 03 wird gelöscht. Es hat keine Auswirkung einen nicht existierenden Breakpoint zu löschen.

ChangeProgramCounter

Es kann bei einem im Stop Zustand befindlichen Gateway der Program Counter, also die aktuelle Position des Scripts manuell geändert werden. Die Angabe der neuen Ausführungsposition erfolgt in 4 Byte ASCII Hex:
P0020: der ProgramCounter wird auf die Adresse 0020 gesetzt (typischerweise der Anfang des Scripts). Mit dem nächsten Go oder Single Step wird das Script ab dieser Position ausgeführt.

Download

Der Download eines Programms ins RAM wird mit D eingeleitet. Es folgt ein Wort Daten (binär) das die Länge der folgenden Bytes angibt, die Daten selbst im binären Format und anschließend die Checksumme (1Wort binär). Der Download wird vom Gateway mit O bestätigt, wenn er erfolgreich war oder mit E, wenn ein Fehler aufgetreten ist.
Die Checksumme berechnet sich als Summe über alle Datenbyte.

Go

Ein Script kann von Anfang des Scripts ausgeführt werden, indem ein Go Befehl ab der Adresse 0 ausgeführt wird. Es kann auch eine beliebige andere Adresse angegeben werden, ab der gestartet wird. Falls eine Adresse angegeben wird, an der kein Scriptbefehl anfängt wird wahrscheinlich ein Fehler ausgegeben werden. Die Scriptadresse wird in 4-stelliger Schreibweise angegeben.
Beispiele: G0000 oder auch z. B. G0043.

MemoryDump

Das user memory des Gateways kann ausgelesen werden. Jede deklarierte Variable hat eine gültige Adresse im user memory. Gültige Adressen liegen vor, wenn der Bereich zwischen 0 und 8000 (hex) ist. Alle anderen Adressen sind keine gültigen Speicheradressen, die zurückgelieferten Werte sind wahrscheinlich ungültig.
Die Anforderung eines Memory Dumps erfolgt durch den Befehl Mxxaaaa, wobei xx die Anzahl der anzufordernden Datenbytes und aaaa die Startadresse ist, jeweils in HEX-ASCII Notation. Die Rückgabe besteht aus 2*xx+2Byte. Das erste Byte gibt die Länge der zurückgelieferten Daten an, danach folgen die Daten im Hex-ASCII Format. Es dürfen max. 128 Byte pro Aufruf gelesen werden; eine größere Zahl kann zu verfälschten Daten während der Ausführung führen.

Es gibt 2 Sonderfälle:

- Auslesen des Feldbuseingangspuffers: die Startadresse ist FFF0
  Die Länge spiegelt hier die tatsächlich verwendete Anzahl der Daten im Bus wider und nicht die Anzahl der gelesenen Bytes.
- Auslesen des RS Eingangspuffers: die Startadresse ist FFE0
  Auch hier ist die Länge die Anzahl der Daten im RS-Eingangspuffer.

Programming

Das aktuelle Script nach einem Download liegt nur im RAM vor und kann nur solange ausgeführt werden, solange das Gateway mit Spannung versorgt wird. Um das aktuelle Script in den nichtflüchtigen Speicher zu übernehmen, muß das aktuelle Script programmiert werden. Dies geschieht, indem das Zeichen E an das Gateway gesendet wird. Nach erfolgreichem Übernehmen des Scripts antwortet das Gateway mit "O", im Falle eines Fehlers mit "E".

Reset

Mit R wird der Reset des Gateways ausgelöst. Nach einem Reset wird allerdings das aktuell im RAM stehende Script verworfen, und beim Neustart das zuletzt im EEROM enthaltene Script aktiviert. Aus diesem Grund sollte entweder vor einem Restart das aktuelle Script ins EEROM übertragen werden (Programming) oder nach einem Restart das Script erneut ins RAM geladen werden. Da der Debugger nicht entscheiden kann, ob das aktuelle Script auch im EEROM enthalten ist, wird immer nach einem Reset auch ein neuer Download durchgeführt.
Nach einem Reset führt das Gateway ein Restart aus und sendet seine Start-Message. Darauf folgend wird es seinen State ausgeben.

SingleStep

Das Gateway kann angewiesen werden eine einzelne Scriptanweisung oder eine Folge von Anweisungen zu verarbeiten. Die Anzahl der Befehle muß mit angegeben werden. Der Befehl lautet "Sxx", wobei x die Anzahl der Befehle ist. Ausnahme: Wird "00" als x angegeben, werden 256 Befehle ausgeführt. Nach der Ausführung des Befehls (der Befehle) gibt das Gateway seinen Status aus.

StartMessage

Wird ein Gateway im Debugmodus neu gestartet (Kaltstart durch Spannung oder Warmstart durch Reset) gibt es die Zeichenfolge CR LF (0x0D 0x0A) aus. Ein angeschlossenes Programm kann nach Empfang dieser Sequenz erkennen, dass das Gerät neu gestartet wurde.

State

Das Gateway wird nach verschiedenen Bedingungen seine State Meldung ausgeben.
1. nach einer Reset Message.
2. bei Erreichen eines Breakpoints
3. nach einer Single Step Ausführung
4. nach dem Ausführen eines Stop Script Befehls
5. nach einem Stop durch den Debugger

Der Aufbau der Meldung selbst ist immer gleich:
Ibbppppeeaass:
I: Konstantes Schlüsselzeichen
bb: Breakpoint
pppp: ProgramCounter, die aktuell erreichte Position des Scripts
ee: ErrorCode, Fehlerzustand zum Zeitpunkt des Breaks 00, kein Fehler.
aa: Die Adresse, an der der Fehler aufgetreten ist
ss: Der Stackpointer

Stop

Durch das Senden des Zeichens X an das Gateway wird das aktuelle Script beim nächsten Befehl angehalten. Das Gateway sendet seine Statusmeldung aus, die Breaknummer ist FF.

Upload

Mit dem Senden des Befehls U an das Gateway gibt das Gerät seinen Speicher aus. Dieser Inhalt ist z. Zt auf 32 Byte begrenzt.
Format des Uploads ist:
1 Wort binär (Datenbytes), Daten im binären Format, 1 Wort Checksumme(binär);
Die Checksumme wird als 16 Bit Summe über alle Datenbyte gebildet.

WriteMemory

Als Äquivalent zum Auslesen des user memory kann der Speicher auch beschrieben werden. Es können max. 128 Byte im Speicher pro Aufruf überschrieben werden. Das Format für den Aufruf ist Waaaaxxd1d2..dn, wobei
aaaa die Startadresse,
xx die Länge der Daten und
d1..dn die Daten selbst sind.
Die Daten müssen im Hex-ASCII Format vorliegen. Die Ausführung des Kommandos erfolgt unmittelbar und liefert keine Bestätigung nach der Ausführung.

Die nachfolgenden Beschreibungen, Befehle und die zugehörigen Erklärungen sind nur in englisch vorhanden. Da alle Befehle selbst in englisch gehalten sind sollte ein Programmierer, der damit zurechtkommt auch kein Problem mit den englischen Beschreibungen der Befehle haben. Es ist in diesem Bereich üblich, der Aktualität wegen keine Übersetzungen zu pflegen.

Die Beschreibungen zu den Befehlen können in diesem Dokument nicht immer aktuell gehalten werden. Es ist daher möglich, daß neuere Befehle oder Korrekturen hier nicht enthalten sind. Diese sind lediglich in der Online Hilfe enthalten.

Auch die Beispiele sind hier nur als kurze Auszüge zu sehen. Weitere Beispiele sowie vollständige Scripte sind auf unserer Homepage erhältlich.

7 Quick start

Getting started with Protocol Developer

Welcome to Deutschmann Automation Protocol Developer. We have designed this product for you to create a protocol for your fieldbus interface, either a UNIGATE CL, UNIGATE IC or a UNIGATE SC, regardless of the fieldbus system.
To get the best benefit from this description you should read it carefully. It is short enough not to bore you while reading and detailed enough to answer most of your questions.

What you need

You need a PC running either Windows 95, Windows 98, Windows ME, Windows 2000, Windows NT or XP. The PC must have at least one serial port available.
You need an installation of the Protocol Developer software package.
And you need the hardware. We strongly recommend a starterkit, containing a script capable device (either a UNIGATE CL, UNIGATE IC or a UNIGATE SC), a cable and a power supply.

Some Add-Ons are available. The Add Ons are easy fieldbus master systems, not capable of running all features but the most important feature for the bus. To use an Add-On you do not need to open your PC and plug in a PC-Card; all Add-Ons are connected to the PC by a serial interface; so you need a second COM Port for the Add-On or a different PC.

What you should do

We recommend that you follow our step-by-step procedure to get a first primitive script running. This script does not make much sense in the meaning of your protocol, but it shows you the general handling of the software and the hardware. After those steps you should be able to load another example file and adapt this file to your requirements.

7.1 Step-by-step

7.1.1 Step 1

Installation

Install the software package Protocol Developer first. If you have ordered an Add-On follow the Add-On page for installation and start of this installation.

Known problems:

1. If you run the software under Windows NT or Windows 2000 under some circumstances we have heard of problems (concerning the Windows registry). If you have troubles try to run the software as a local administrator.
2. Some notebooks or laptops do not work properly at the serial communication port. You do not have another solution than using a different computer.

7.1.2 Step 2

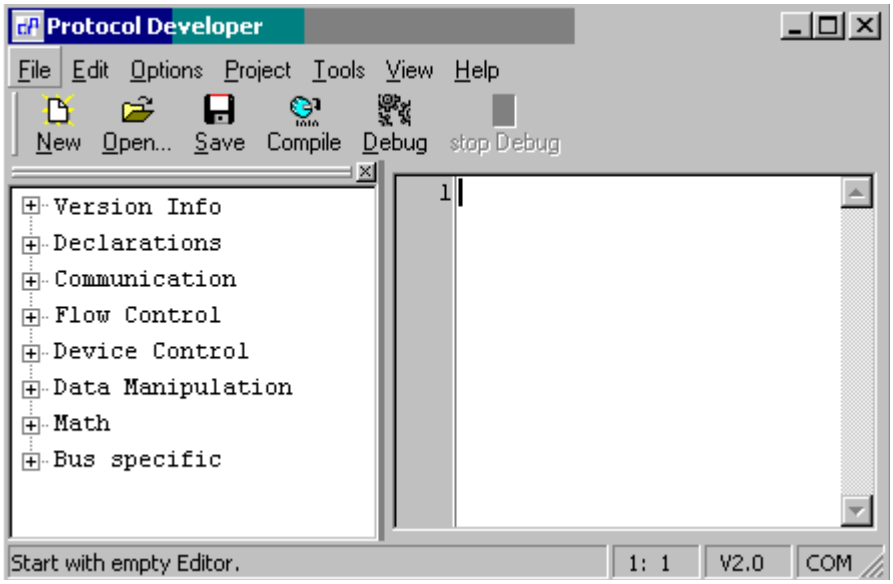
Connecting the device (Power and Debug)

At this point it is not necessary to connect a fieldbus cable or your device. We will do this step after we have checked the general functionality.
You do not need the adapter for configuration now.

7.1.3 Step 3

Starting the Software

Start the software Protocol Developer. After a short while it should come up with an empty file. It should look similar to the screen shot.



Now we write a first trivial script. This script does not have any real functionality, but it shows the general handling. You may "copy and paste" the script code from this here. All commands can also be found in the list on the left side.

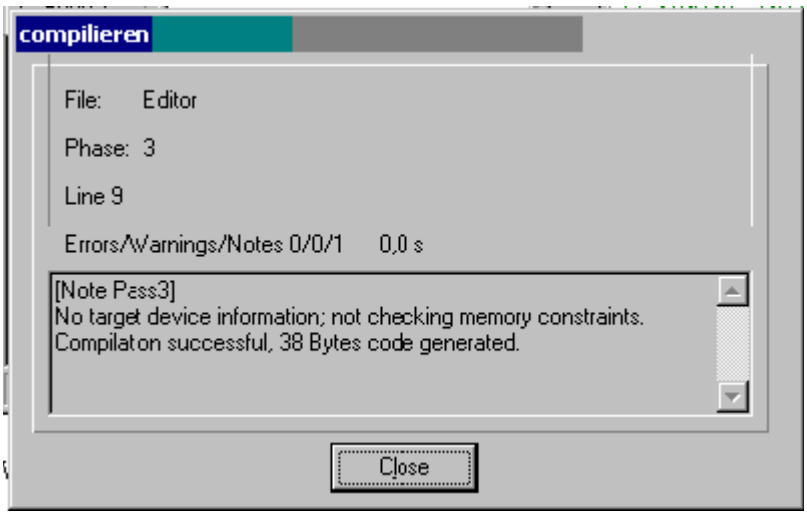
```
// Script initialization
ScriptName "First Simple Script"

// we declare a variable and define this variable with a fix value
var vw_a: word;
MoveConst (vw_a, 100);

// stop the script.
stop;
```

7.1.4 Step 4

Compile the script and start the device

After writing the script you have to compile the script. Use the compile button or the menu to do so.

If you do not have any errors in the script you will get a Compiler Window like this:



The script is now checked for syntax and a compressed code is generated for the code.

7.1.5 Step 5

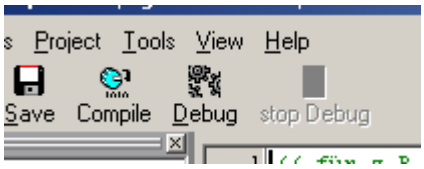
Start Debugger

Now you should start the UNIGATE® CL, UNIGATE® IC or UNIGATE® SC by applying the power to the device. Depending on the device it takes a few seconds for starting. In the lower left corner of the Protocol Developer you will see a message showing the device is in Debug Mode.

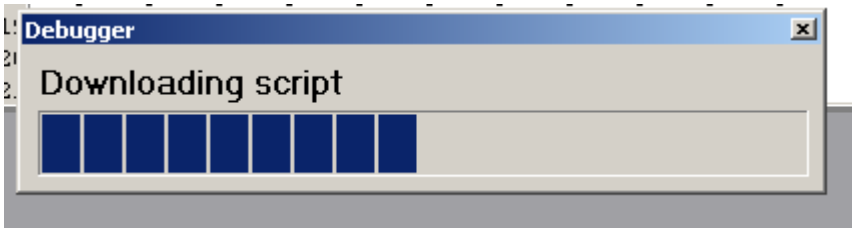
From now on it is possible to debug the script.
In Debug Mode the device is not executing the script; it waits until for user commands to start and stop the script.

Every time the device is switched off and on again you will see this message.

Please save this file now because it is only possible to debug a saved file.



Now it is time to start the Debugger.
For this purpose you should select "Debug" from the project menu or use the debug button.

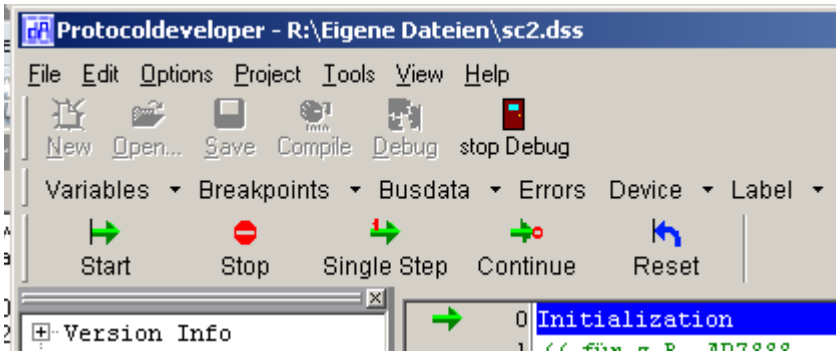


When you start a debug session the script is automatically compiled and downloaded to the script device. This transmission takes a short while, depending on the size of a script. For this time you see a progress bar.

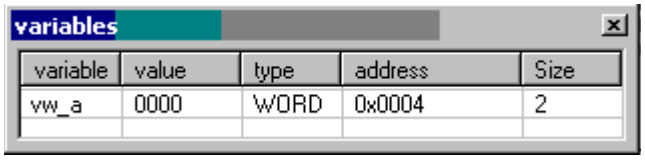
7.1.6 Step 6

Debugging

After the script is completely downloaded to the device the Debugger is stopped, the device's script is not executed. You will see the Debugger's main window, which has two more button bars for debugging.
You should note the green arrow, which marks the next line to be executed by the device.



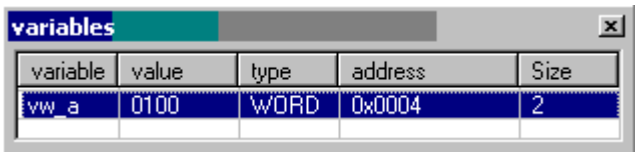
In our script we use a variable called vw_a (our variable-naming convention for variable word, so we can see its type everywhere the variable is used). Now click on the variables button and add these variables to the window (right mouse "Add" in window or drop down list in variables button); thereafter it looks like this:



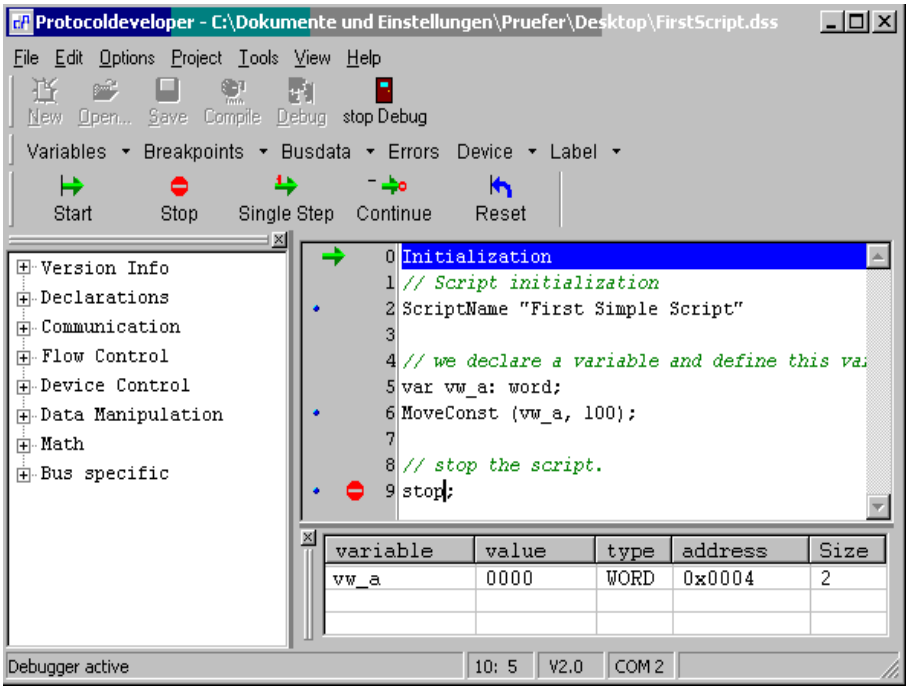
7.1.7 Step 7

Single Step

You may now click onto the single step button until the script is at the STOP command. After every step the Protocol Developer refreshes all variables used in the variable window, so after executing the line "MoveConst ..." our variable window looks like this:



Now please click on the reset button and wait until the device is reset. All display is refreshed, all memory is set to 0 again.
Select the line with the stop command in the debugger window.
Press the "F5" button or select from the context-sensitive menu (right-mouse-click) the "toggle breakpoint" item. Your screen should look like this:



After you click onto the "continue" button the script is executed until a breakpoint is reached. The Debugger stops right before the breakpoint, so the stop command itself is not yet executed. Even in this case the variable window is refreshed.

7.1.8 Step 8

Congratulations!

Now you have learnt how to use the Protocol Developer. Most functions are similar to the operations shown in the last steps. This short introduction does not show all debugging or script writing techniques, but it shows the principles of a script and how to create and debug it.

It is now up to you to write your own script. Feel free to have a look into our example files. Use this Online reference to learn about the script commands.

We hope that you are satisfied with our product.

8 Commands (selection of commands)

Please note that this chapter does not contain all commands.

Only the online help includes the descriptions of all commands!

8.1 BusStart

Syntax

BusStart ;

Description

The fieldbus will not start until the Script gateway executes this command.
It is always save to use this command even in busses not supporting a bus start. Such gateways will ignore this command and will not produce an error on execution.
You should use this command if you are writing one scripts for all busses.

Example

```
// Start of Script Set BusInputSize to 8; // operation should
be done before bus starts
Set ( BusOutputSize, 4 ) ;
BusStart;
...
// bus is may now became active
```

8.2 CalculateByte

Syntax

CalculateByte (Source1=v1, Source2=v2, Operator=v3, Destination=v4) ;

Description

This command is used for assigning the result of a mathematical or logical operation of exactly two operators to a given variable. The result of the operation is a word-value, parameters are byte-value only; if the parameter is a word only lower 8 bits are used!

Example

```
---------------- example code cut here -----------------
var a: word;
var b: word;
var c: word;

MoveConst( a, 0b0100) ;
MoveConst( b, 0b0101) ;
CalculateByte( a, b, and, c) ; // c ist now 0100
---------------- example code cut here -----------------
```

Execution Code

Possible execution codes are
DIV_ZERO: Operator v2 is zero; division by zero is not allowed.
UNKNOWN_FUNCTION: The selected operation is not part of this script device's firmware. Check script revision for support of this operator.

See also

Mathematical and logical operators

Note

You should know that some functions may return a word value. If you use a buffer variable as the result variable this is ok, if you use a buffer with an index the byte with index-1 is affected too.

Example:

```
Buffer1 contains some bytes in sequence ( 01  02  03  04).

var buffer1: buffer[4];
MoveConst( Buffer1, #1#2#3#4 );
CalculateByte( Buffer[1], Buffer[2], or, Buffer[3] );
The resulting buffer contains the following values ( 01  02  03  03).
```

8.3 CalculateWord

Syntax

CalculateWord (Source1=v1, Source2=v2, Operator=v3, Destination=v4) ;

Description

This command is used for assigning the result of a mathematical or logical operation of exactly two operators to a given variable. The result of the operation is a word-value. You will get an error if a result does not fit in a word.

Example

```
---------------- example code cut here -----------------
var a: word;
var b: word;
var c: word;

MoveConst( a, 0b0100) ;
MoveConst( b, 0b0101) ;
CalculateByte( a, b, and, c) ; // c ist now 0100
---------------- example code cut here -----------------
```

See also

Logical operators
Mathematical operators

8.4 Call

Syntax

Call subroutine ;

Description

The Call command calls a subroutine. The start of the subroutine is given by a label declaration, the routine's end is the return statement. It is necessary to complete a subroutine with a return statement, if not the gateway shows an error.

Errors

This command results in an error if return address is not available.

Example

```
call :subroutine;
stop;

: subroutine ;
// do something
return;
```

See also

Command return
Label

8.5 Checksum

Syntax

Checksum (Source=v1, Destination=v2, NumberChar=v3, ChecksumMethod=v4) ;

Description

This command calculates a checksum with the given method for a number of bytes in the device's memory.
The variable source is a buffer containing the data the checksum should be calculated for. Destination is a word variable which holds the checksum after the routine finishes with return code OK. The number of characters is a variable which holds the number of bytes the checksum should be calculated for. Checksum method defines which method should be used for calculation.

Example

```
var num: byte;
var Data: buffer[10];
var Result: word;
MoveConst ( num, 5 ) ;
MoveConst ( Data, "Hello" ) ;
Checksum ( Data, Result, Num, CRC16 );
```

See also

Checksum methods

8.6 Convert

Syntax

Convert (Source=v1, SourceType=v2, Destination=v3, DestinationType=v4) ;

Description

Use this command if you like to change its data format. It is possible to convert 3 bytes ASCII to a binary word value or vice versa.
All known conversion types are usable either for source and for destination, so data conversion is possible from every data representation to another.

Errors

A conversion error may occur. This error is to be detected by checking the ErrorCode parameter.

Example

```
var Data: buffer[3];
var Result: word;
MoveConst ( Data, "123" );
// convert 3 Bytes ASCII-Dezimal into binary number
Convert ( Data, ASCII_DEZ_3, Result, byte );
```

8.7 Copy

Syntax

Copy (Source=v1, Destination=v2, NumberChar=v3);

Description

Copy a number of bytes from one variable to another. The number of bytes is a variable containing the number of bytes, Source and Destination are meant to be Buffer variables; it is also possible to use an ordinal variable as a byte, word or long instead.
If necessary it is possible to indicate an ordinal variable.

Example

```
var b1: buffer[10];
var b2: long;

var num: word;

MoveConst ( b1, "++++++++++" ) ;
MoveConst ( b2, 0x30313233 ) ; // means ASCII 0,1,2 and ASCII 3
MoveConst ( num, 3 );
Copy ( b2[1], b1[1] , num ) ;   // copy 4 bytes

// b1 is now "+123++++++"
```

.

See also

FillMemory
CopyIndexed

8.8 Delay

Syntax

Delay (ConstTime=v1) ;

Description

Execution of the actual script is delayed by the given time. The time is in milliseconds and can take directly values from 0 to 65535. It is not possible to hold the time in a variable.

Errors

This command will not produce any errors.

Example

```
// assume time is 12:00:00
delay ( 10000 );
// time is 12:00:10
```

8.9 DIN19244DataExchange

Syntax

DIN19244 (InBuffer, OutBuffer, Insize, Outsize);

Description

This command exchanges data with DIN 19244 routine.
Send buffer and Receive buffer must have a special data format. See DIN 19244 specification for details.
After data was sent by the gateway the routine waits for a response.

Example

www.deutschmann.de

8.10 ExchangeModbusRTUMaster

Syntax

ExchangeModbusRTUMaster (Source=v1, Destination=v2, NumberCharReceived=v3) ;

Description

This command takes two buffers and sends the data given in the variable Source. Source contains all data necessary for a Modbus RTU record except the CRC checksum. This checksum is generated by the function itself.
The function returns the number of bytes received by the Modbus Slave. If a timeout occurs this value is 0. If no timeout occurs the function copies the slave's response data into the variable Destination.

Example

An example is available from our website
www.deutschmann.de.

Errors

A timeout may occur.

See also

Errorcodes

8.11 FillMemory

Syntax

Fill (Destination=v1, Char=v2, NumberChar=v3) ;

Description

This command fills a given number of bytes of the desired variable with a specified character. The number of characters to be filled is a variable containing the number. The variable source is filled with the character contained in the variable fillchar.

Example

```
// Script Start
var fillchar: byte;
var number: byte;
var data: buffer[10]; // Variable data contains 0x00 in all
bytes
MoveConst ( number, 10 ) ;
Moveconst ( fillchar, "?") ;
Fill ( data, fillchar , number ) ;
//data now is "??????????"
```

8.12 GetParameter

Syntax

Get (Parameter=v1, Returnvalue=v2) ;

Description

With this command it is possible to read most of the settings of the gateway.
Most important is the capability of reading the RS-switches and the type of RS-interface. It is also possible to determine the baudrate, input-/outputsize for the bus and some other parameters.

Errors

This command does not produce any errors.

Example

```
var RSSwitchInfo: byte;
Get ( RS_Switch , RSSwitchInfo );
// variable RSSwitchInfo now contains a number. See parameter
RS_Switch for more information.
```

See also

Parameters

8.13 If - then - else

Syntax

if variable1 operator variable2 then :thenpath else :elsepath;

Description

Variable1 and variable2 are compared.
If the result of operation is true the script is continued at :thenpath, otherwise it is continued at :elsepath. Comparison of the variables is type-depending. The type of relation is determined by the smaller variable types (types with less bytes). A buffer variable is treated as a byte-variable.

Example

```
var a: word;
var b: byte;
MoveConst ( a, 257 ) ; // binary of a is 0x01 0x01
MoveConst ( b, 2 ) ; // binary of b is 0x02;
if a greater b then :isgreater else :isnotgreater ;//
// only the lower byte of variable a is used for compare
function,
// therefore variable a (0x01) is not greater than b (0x02)
 :isnotgreater;
MoveConst ( a, 0 ) ;
if a greater b then :isgreater else :DoError ;
 :isGreater;
// a is really greater than b
...
:DoError;
```

See also

Operators

8.14 Init3964R

Syntax

Init3964R (Priority, CONST MaxReceiveSize) ;

Description

Communication with 3964R needs an initialization before data exchange is possible. You need to set the ReceiveBuffer and the maximum number of bytes to receive.

Example

```
...
Init3964R( low, 10 );
// The communication is initialized with low priority
// maximum receive size is 10 byte user data
...
```

Notes

A character DLE, which is duplicated by the communication is not counted twice by the maximum receive size. Handshaking characters and framing characters are not counted either.
A number of protocol errors may occur. Those errors are handled by the operating system of the device and can be caught by the OnError function.

8.15 InitCommunicationChannel

Syntax

InitCommunicationChannel (CONST channel, Value);

Description

A communication channel must be initialized exactly once before used. You will get an error if you try to initialize the channel two times.
Up to 16 channels may be opened at one time; the number of channels depends on the device type.

Example

```
...
Var ArcnetID: long; // declare variable
MoveConst ( ArcnetID, 4 );   // assign value
 InitCommunicationChannel( 1, ArcnetID ) ;
// The communication channel 1 is initialized with
// ID 4. This means all communication send to
// this channel is going to Arcnet Partner #4.
...
```

Notes

Fieldbus type	Meaning
ARCNET	Send ID of ARCNET message.
CANopen®	Channel 0 is PDO1. No others are defined yet.
DeviceNet	Channel 0 is poll connection. No other connections are defined yet.
Ethernet	TCP-IP destination address for sending packets. Destination is sender if the value is 0.
Interbus	Only standard communication channel (process data) available.
LON	Not available.
MPI	MPI partner is adjusted by parameters.
ProfibusDP	Only standard communication channel (cyclic process data) available.

8.16 Jump

Syntax

jump :address;

Description

A script execution is continued at the given address. The address itself is a label. The compiler stops if a destination label is not defined by the script. Resolution of labels to addresses is done by the compiler.

Errors

An error 5 (unknown command) is produced if the address points to an invalid command.

Example

```
jump :GoOn; // continue script execution at this label
stop ; // This code is never executed
: GoOn; // Next command after jump
...
```

See also

Label declaration

8.17 Label

Syntax

:Identifier ;

Description

A Label is defined by the first character ":" and the following identifier. A Label is a mark in the script which could be used by a jump, a call or an if statement.

Example

```
jump :GoOn; // continue script execution at this label
stop ; // This code is never executed
: GoOn; // Next command after jump
...
```

See also

Jump
Call
If
SetErrorHandler

8.18 LONSelfDocString

Syntax

LonSelfDocString (source=v1 , Numberchar=v2) ;

Description

Device self-documentation string. If the documentation string is not supplied, there is a single line containing a single asterisk. If supplied, the documentation lines begin with a double-quote character (not part of the documentation string) each. Multiple lines must be concatenated without any intervening characters. There is no end double-quote, instead the line is terminated by a new line. The characters of the string must all be printable ASCII characters (this includes spaces, but not tabs). Trailing spaces are included. The line may be up to 60 characters long, not including the starting double-quote character or the new line. Any non-printable characters must be encoded using an ANSI C hex character escape sequence of "\xHH"' where H represents a single hexadecimal digit. The values A-F within a hex character escape sequence must be specified with upper case letters exclusively.
If the static interface contains LONMARK objects, the device self-documentation string should be formatted as described in The LONMARK Interoperability Guidelines.

Example

```
...
// variable declarations
var SelfDocsize : word ;
var SelfDocBuffer: buffer[60];
 moveConst( SelfDocsize, 60);
 // define static docstring
moveConst( SelfDocBuffer[0], "Hallo World
345678901234567890123456789 0abcdefghijklmnopqrst" );
 // activate docstring
LonSelfDocString( SelfDocBuffer[0], SelfDocsize);
.
..
```

Values

The length of Numberchar may be 0 .. 60

8.19 MoveConst

Syntax

MoveConst (Destination=v1, ConstantValue = v2) ;

Description

A constant value is transferred into the memory used by the given variable. This command is used to predefine a variable with a constant value.

Example

```
var a: byte;
var b: buffer[10];
 MoveConst ( a, 2 ); // a contains now 2
MoveConst ( a, 0b10101 ); // a contains now 21, which is 10101
binary
MoveConst ( a, 0x0A ); // a contains now 10 which is
hexadecimal 0A
MoveConst ( a, "A" ); // a contains now 65, which is ASCII-
code for character "A"
MoveConst ( b, "Hello"#0x0D ) ;
// b contains now text "Hello" followed by the character 13,
which is 0x0D(hex)
```

Memory organization

Every variable in the script is meant to be a cell in the device's memory. The address of the variable is determined by the Protocol Developer software. A variable of type byte needs 2 bytes memory of the device, a variable of type long needs 4 bytes of memory. A buffer variable needs the number of data bytes given in its declaration.
A byte variable can keep values from 0 to 255 (0x00 to 0xFF).
A word variable can keep values from 0 to 65535 (0x0000 to 0xFFFF).
A long variable can keep values from 0 to 4294967295 (0x00000000 to 0xFFFFFFFF).
The address of a byte variable can be 0x0003. If this variable is used as a word variable, the Protocol Developer changes the address from 0x0003 to 0x0002, so the value can be used as a word value.
Normally it is safe to use a byte variable as a word value. If you assign a word value to a byte variable, the higher byte of the variable may be lost.
If you like to use a buffer element as a byte variable you must convert the element to a byte variable and use the resulting variable.

8.20 ReadBus

Syntax

ReadBus (Destination=v1, Numberchar=v2) ;

Description

The number of bytes given in the variable number is read from the bus input buffer. The number of bytes available should be read from the parameter BusInputSize. The destination buffer must be big enough to hold all bytes from the bus.

Example

```
var BusSize: byte;
var data: buffer[10];
 // assume bus contains new data "Hello"
Get ( BusInputSize , BusInsize ); // assume a Bus input size
of 5 bytes
ReadBus ( Buffer , BusSize ) ; // Buffer Data contains now
"Hello"
```

See also

Get BusInputSize
Set BusInputSize
WriteBus

8.21 ReadModbusSlave

Syntax

ReadModbusSlave (Destination, Length) ;

Description

This command reads a request with Modbus RTU. If the value for length remains 0 after the function was called the master did not tried to read the slave, otherwise the value contains the number of valid bytes in the Destination buffer.

Example

An example is available from our website
www.deutschmann.de.

Return codes

MODBUS_ERROR
RX_OVERRUN
RX_DIN19244_MODBUS_ERROR
CHECKSUM_ERROR

See also

Parameter ModbusSlaveAddress

8.22 Receive3964R

Syntax

Receive3964R (Source, Size, Timeout) ;

Description

It is possible to wait for an incoming data record in the 3964R format. This means, the routine handles the complete STX, ETX, BCC and DLE handling. You must give 3 variables to the function: source: a buffer variable which holds the received data after the function returns with function code OK. The variable size is overridden by the function and holds the number of received characters after the function completes with result code OK. The variable Timeout defines how long the routine should wait for the partner to send.

Example

```
..
var RcvBuffer: Buffer[10];
var Size: Word;
var Timeout: Word;
MoveConst ( Timeout, 1000 ) ;
Receive3964R ( RcvBuffer, Size, Timeout );
...
```

Return codes

```
OK
TIMOUT
3964R_ERROR
CHECKSUM_ERROR
3964R_WRONG_CHAR
```

8.23 ReceiveSomeCharRS

Syntax

ReceiveSomeCharRS (Timeout=v1, ReceiveDataBuffer=v2, NumberCharToReceive=v3) ;

Description

This command waits for the receipt of the given number of characters. The time between any characters must not exceed Timeout milliseconds. If variable Timeout contains 0 no timeout control is used. All characters received are stored in variable Destination. This variable must be big enough to hold all incoming characters, for example a buffer variable.

Example

```
var timeout: word;
var number: word;
var data: buffer[10];
 MoveConst ( number, 3 ) ;
MoveConst ( timeout, 1000 ) ;
// Assume data string "Hello" not yet read in the gateways RS-
Input Buffer
ReceiveSomeCharRS ( timeout, data , number ) ;
// 3 bytes are read, data now contains "Hel",
// RS input Buffer still contains data "lo"
// which is the rest of the data "Hello"
```

Note

If you are using 9 databits every character received is stored as 2 bytes in the receive buffer. If you like to receive 4 characters consisting of 9 bit each character you should set your NumberToReceive to 8.

8.24 ReceiveSpecialCharRS

Syntax

ReceiveSpecialCharRS (Char=v1, Timeout=v2, ReceiveDataBuffer=v3, Size=v4) ;

Description

This command waits the time given in variable Timeout for the character variable Char points to. All characters received until the specified Char is recognized are stored in the buffer ReceiveDataBuffer. After the command is finished the variable Size contains the total number of bytes received.

Example

No example is available at the moment.

Note

If using 9 databits (= CharToReceive) it is only possible to trigger for the higher byte of a character. The higher byte can only be 0 or 1.

8.25 Return

Syntax

return;

Description

Return completes execution of a subroutine and continues the script execution with the command after the call statement. You should never use a return command in a script if you do not use a call. Using the return without a prior call produces an error.

Example

```
call :subroutine; // subroutine is executed and finishes
stop;
 :subroutine ;
// DoSomething
return;
```

See also

Jump

8.26 ScriptAuthor

Syntax

ScriptAuthor ("Author") ;

Description

Every script may have an author. An author is a text containing characters up to a size of 32 bytes.
This text is displayed in the device's boot message.
The boot message is sent out by the device in Configmode on startup.
The Script author item must be one of the first commands in the script; otherwise it is without a function.

See also

Script Name
Script Revision

8.27 ScriptName

Syntax

ScriptName "NameString" ;

Description

Every script may have an Name. A name is a text containing characters up to a size of 32 bytes. This text is displayed in the devices boot message.
The boot message is sent out by the device in Configmode on startup.
The ScriptName item must be one of the first commands in the script; otherwise it is without a function.

See also

Script Author
Script Revision

8.28 ScriptRevision

Description

Every script may have a revision. A revision is a text containing every character up to a size of 31 byte. The size may differ for some devices in the future; a typical revision is an upcounting number or a short text.
This text is displayed in the device's boot message.
The boot message is sent out by the device in Configmode on startup.
The Script revision item must be one of the first commands in the script; otherwise it is without a function.

Command

ScriptRevision (RevisionText) ;

Example

```
-----
ScriptRevision ( "V 1.0" );
----

e.g. message from a device
RS-EN10-SC D(232/485) V2.1t[13] (c)dA Switch=0xFFF
Script="sc" Author="DA" Version="V 1.0" Date=16.11.2001 SN=0
Konfigmode...
```

Defaults

There is no default value for the script revision.

See also

ScriptAuthor
ScriptName

8.29 Send3964R

Syntax

Send3964R (Source, Size) ;

Description

It is possible to send a number of bytes (a data record) with the procedure of 3964R. The routine itself only requires the data and the number of bytes to be sent; the protocol handling (STX, ETX...) is done by the routine. If the partner does not respond or a conflict occurs an error can be detected with the "Get (ErrorCode, variable)" command.

Example

```
...
var SendBuffer: Buffer[10];
var Size: Word;
MoveConst ( SendBuffer, "Hallo"#13#10 );
MoveConst ( Size, 7 );
Send3964R ( SendBuffer, Size );
...
```

Notes

```
OK
TIMOUT
3964R_ERROR
CHECKSUM_ERROR
3964R_WRONG_CHAR
```

8.30 SendRS

Syntax

SendRS (Source=v1, NumberChar=v2) ;

Description

SendRS writes the given number of characters to the output buffer of the gateway. As the buffer is filled with data the hardware (no longer the software) is responsible for accessing serial lines. With the action SendRS the script has no more control over the data, it is not possible to clear output string etc.
Number is a variable containing the number of bytes to be sent, source is a variable containing the data itself.

Example

```
var src: buffer[7];
var size: word;
 MoveConst ( src, "Hello"#0x0A#0x0D ) ;
MoveConst ( size, 7 ) ;;
SendRS ( src [0], size ); // writes "Hello" followed by CR-LF> to
serial port
```

See also

ReceiveSomeCharRS
ReceiveSpecialCharRS

Note

The number of bytes sent by the device is always given in byte. If you are using 9 databits every character must consists of 2 bytes in the send buffer, the higher byte contains the MSB only (0 or 1) and the lower byte contains the data byte.
The number of bytes to be sent must be even, and the higher byte must always be 0 or 1. An error occurs if one of those conditions is violated.

8.31 Set

Syntax

Set (parameter=v1, Value=v2) ;

Description

Sets the given parameter to the value. The parameter is one of the parameters defined for this command, the value must be one of the valid values for this parameter.

Errors

If a parameter or a value is not supported by the gateway it will show an error and stop all actions. Script execution with such a problem makes no sense.

Example

```
Set ( Baudrate, 9600 ) ;
Set ( Parity, none ) ;
Set ( RS_State_LED, RedGreenFlashing );
```

See also

GetParameter
SetByVar

8.32 SetByVar

Syntax

SetByVar (parameter=v1, Value=v2) ;

Description

Sets the given parameter to the value. The parameter is one of the parameters defined for this command, the value must be a variable containing a valid value for this parameter.
You have to make sure that the variable type MUST be correct for the parameter. The Protocol Developer will not make any type conversion nor check the correct type of the parameter.
If you use a wrong variable type it will result in runtime errors.

Errors

If a parameter or a value is not supported by the gateway it will show an error and stop all actions. Script execution with such a problem makes no sense. All those errors could be trapped with OnError function.

Example

```
var iBaudrate: long;
MoveConst(iBaudrate, 9600 );
SetByVar ( Baudrate, iBaudrate ) ;
```

8.33 SetLonMapping

Syntax

LonInMapping (source=v1, Numberchar=v2) ;
LonOutMapping (source=v1, Numberchar=v2) ;

Description

LON defines a number of SNVT In and SNVT Out variables. A variable is always seen from the LON device, this means an In-variable is received by a LON device.
It is necessary to tell the gateway which Input and Output variables are to be used. This is done with the commands LonInMapping and LonOutMapping. All SNVT's which should be available on LON-side are declared by those two commands.
All SNVT-types available are given in the map-table, which is a normal script buffer variable. This variable consists of a number of bytes and each byte defines an SNVT-type. The commands LonInMapping / LonOutMapping create all requested SNVT's.
LON variables of type 0 are defined by the commands Set(BusInputSize,x) and Set (BusOutput-Size, x). By setting BusInputsize or BusOutputSize to a value greater 0 a variable of type with the given size is created. If the given size is 0 or the command is never called no SNVT 0 is created. The similar behaviour is available for output data.

Values

The length of an SNVT for SNVT's type 1 .. 145 is defined by the SNVT itself. The length of a SNVT type 0 is defined by the commands Set BusInputSize/BusOutputSize.

Example

```
// variable declarations
var InMapsize : word;
var InMapTable: buffer[4];
var BusInSize: word; var ReceiveBuffer: buffer[5];
// define 2 bytes for mapping
// Maptable consists of 4 bytes, but only 2 bytes are used
moveConst ( InMapsize , 2) ;

// define static mapping
MoveConst ( InMapTable[0] , #8#7 );
// type 8 = count= 2 byte
// type 7 = char_ascii = 1 byte, is a total of 3 byte

// activate mapping
LonInMapping ( InMapTable, InMapsize ) ;

...
Get ( BusInsize , BusInputSize ) ; // var has now value 3, 3 bytes
ReadBus ( ReceiveBuffer , BusInsize ) ;

// ReceiveBuffer is now updated,
// Bytes 0 and 1 contain data from SNVT 8
// Byte 2 contain data from SNVT 7
```

8.34 Stop

Syntax

stop ;

Description

This command stops the execution of the script. This is useful if the script comes to a point a further script execution makes no sense, e. g. if BusInputSize does not fit the required values.
If the gateway shows an error (indicated by a flashing LED) the error still remains (LED still flashes).

Example

```
...
// stop condition reached
Set ( Error , 8) ; // Errorcode 8 is shown
stop ;
...
```

8.35 VariableDeclaration

Syntax

var VariableName: Type ;

Description

Declare a variable with a type. The size of the variable is depending on the type.
A variable name can be every valid identifier, beginning with a character or an underline, followed by an alphanumerical value or an underline. The length of a variable name is not limited and does not reflect usage of memory in the gateway. You should choose names which describe their content instead of abbreviations.
The compiler automatically arranges variables in the order of declaration in memory, leaving no memory holes..

Types

byte
A byte is a variable with 8 bit datawidth. Such a variable can hold binary values from 0 to 255 or a single character.

word
A word is a variable with 16 bit datawidth. A word variable can hold binary values from 0 to 65535 or 2 characters.

long
A long value can hold binary values from 0 to 4294967295 or 4 characters.

buffer
A buffer is additionally defined by the size. Its size is from 1 character to 255 characters.
Every single char of a buffer is accessible by the index.

Example

```
var Size: byte;
var Destination: word;
var Source: buffer[5];
MoveConst ( Size, 1 );
Copy ( Source , Destination, size );
```

8.36 Wait

Syntax

Wait (condition) ;

Description

The gateway waits until the condition is completed. The condition is only one of the predefined conditions, not every possible expression.

Example

```
...
wait ( Bus_Active );
// Bus is now active, e.g. data exchanging
...
```

8.37 WaitBusChange

Syntax

WaitBusChange (Timeout , WatchSize) ;

Description

The command waits the given time for changing busdata. The number of bytes set in the variable Watchsize are observed. If no byte of the observed changes a timeout (return code 2) occurs. The command immediately returns on changing data.
If the value for timeout is 0 then the commands wait infinite.
The value for the Watchsize should never be 0.

Execution Code

PARA_NUMBER_ERROR	The number of parameters when calling the command is not ok. Please check all parameters.
PARA_RANGE_ERROR	The number of bytes to be observed is zero; this is not allowed.
TIMEOUT	The number of busdata has not changed within the given time.

Example

```
var timeout: word;
var watchsize: word;

MoveConst ( timeout, 1000 );
MoveConst ( watchsize, 5 ); // observe 5 characters
WaitBuschange ( timeout, Watchsize );
// command waits max 1000 ms for changing busdata, 5 bytes are
observed.
```

Note

This command requires a word variable for the parameter Timeout and a word variable for the parameter Watchsize. If those parameters are not word-variables you will get wrong results. If your variables for Watchsize or Timeout have other types use a type conversion prior to using this command.

8.38 WriteBus

Syntax

WriteBus (source=v1, NumberBytes=v2) ;

Description

This command refreshes the busdata. The given number of bytes are written to the fieldbus controller hardware, software is no longer responsible for the data.

Errors

If you send more data than the bus is capable to send an error occurs.

Example

```
...
var size: byte;
var data: Buffer[5];
MoveConst ( size, 5 ) ;
Moveconst ( data, "Hello" ) ;
WriteBus ( data, size ) ;
...
```

8.39 WriteModbusSlave

Syntax

WriteModbusSlave (Source) ;

Description

This command writes response with Modbus RTU.

Example

An example is available from our website www.deutschmann.de.

Return codes

MODBUS_ERROR
TX_19244_MODBUS_ERROR
SEND_LEN_ERROR
PARAM_RANGE_ERROR

See also

ReadModbusSlave
Parameter ModbusSlaveAddress

9 Parameters (selection of parameters)

Only the online help includes the descriptions of all parametrs!

9.1 3964RPriority

Description

Both partners of a 3964R connection need a Priority. One partner MUST have Low, the other one MUST have High.

Commands

Init 3964R

Defaults

A default value for this parameter does not exist.

Example

```
...
Set ( FieldbusID, 5); // Sets Id to 5
...
```

9.2 AvailableBusData

Description

For Ethernet, ARCNET and other busses which support different sizes it is necessary to determine how many data bytes are available by the bus

Commands

This parameter is available for the command Get.

Defaults

A default value is not available for this parameter.

Example

See file:

Note

This parameter is available from script revision 16 on.

9.3 Baudrate

Description

The baudrate of the RS interface is a read/write parameter. It is possible to change the baudrate at any time. For all busses the value for the baudrate is a long-value.

Commands

Set baudrate
SetByVar
Get baudrate

Defaults

The default value for baudrate is 9600 baud.

Values

RS232 baudrates are allowed from 110 baud to 57600 baud. RS485 and RS422 baudrates are allowed from 110 to 625000 Baud.

Example

```
...
Set ( Baudrate 9600); // baudrate is now 9600 baud
...
```

9.4 BusBaudrate

Description

For some busses it is possible or necessary to set the bus baudrate.
The behaviour is bus-depending.
PROFIBUS will always ignore this parameter because the PROFIBUS baudrate is adjusted only by the master.
You will get an error if the device does not support this parameter.

Commands

Defaults

Example

```
...
Set ( BusBaudrate, 2500000 );
// This command defines the bus baudrate to be 2.5 MBaud.
// This baudrate is available for ARCNET.
...
```

Note

If a bus does not support the parameter BusBaudrate the device will ignore the command.
This parameter is available from script revision 11 on.

9.5 BusDataChanged

Description

Reading this parameter returns 0 if busdata did not change since starting the gateway or last call to ReadBus. If busdata changed since then the result of this parameter is 1.
Use this parameter to read and process busdata only if data has changed. This makes the script more efficient.

Command

This parameter is available for the Get command. It is not possible to set this parameter. You will get an error if you try to set this parameter.

Defaults

This parameter always returns an actual value and never a default value.

9.6 BusInputsize

Description

Read or write the actual size of the bus input side, input is seen from the gateways view.

Commands

set BusInputSize
get BusInputSize

Defaults

The default value for BusInputSize is 8, which means a default size of 8 bytes is transferred from the bus master to the gateway.

Example

```
...
Set ( BusInputsize, 12) ; // Now the BusInputSize is 12 bytes
...
```

Comments

This command is bus dependent, e.g. it is not possible to use more than 8 bytes Input size for a small Interbus slave,
You should be aware of this problem if you design a script for more than one bus.
PROFIBUS DP ignores this command because bus input size is defined by the module from the device's GSD file.

9.7 BusOutputSize

Description

It is possible to set the Bus Output size of a device. For some busses it must be set prior to Busstart.

Command

SetParameter

Defaults

The default value for the BusOutputSize is 8 byte.

See also

Parameter BusInput Size

9.8 BusTimeout

Description

Some busses need a value for a bus timeout.

For ARCNET this is the reconfiguration timeout.

Commands

Set (BusTimeout, ...);
SetByVar (BusTimeout, ...);

Defaults

For ARCNET a default value is defined as a named constant ARCNET_DEFAULT.
This default value is 41 microseconds.

Example

```
Set ( BusTimeout, ARNET_DEFAULT );
```

Note

If a bus does not support Set BusBaudrate the device will ignore this command.
This parameter is available from script revision 16 on.

9.9 BusType

Description

This parameter returns the bustype of the gateway running the script. Normally it is not necessary to detect the bustype because all Script commands are executed by all Script Gateways. It may be useful if you like to show on which device a script is running.

Return codes

See Bustypes for more information.

9.10 ChecksumCalculationMethods

Description

Following methods are defined for checksum calculation:

Xor: bytewise xor operation, result is a byte.

XorNot: bytewise xor operation, result is inverted and a byte.

Sum: bytewise add operation, result is a word.

SumNot: bytewise add operation, result is inverted and a word.

CRC_16: e. g. like Modbus RTU.

Command

Checksum

Defaults

A default value does not exist for checksum.

9.11 CommunicationChannel

Description

This parameter is used to select a communication channel.

Commands

Set

9.12 DataBits

Description

It is possible to set the number of data bits for the serial communication.

Defaults

Default value for data bits is 8.

Values

Possible values are:
7
8
9

Note

9 databits are available in script revision 13 and higher. You should know that the behavior of the receive and send functions is different in 9 bit mode.

9.13 ErrorCode

Description

It is possible to display a user defined code. If an errorcode is set the RS-State-Led flashes slowly red (1 per sec) in difference to fast red flashing if the gateway itself shows an error.

Defaults

No error is shown per default.

Values

Every value from 0 to 15 takes effect.
By setting the value to 0 all previously set errorcodes are deleted.

Bus and device specific behavior

Devices of our UNIGATE-SC series are capable of displaying an error with LED's directly.
UNIGATE® IC's only display the errorcode in the fieldbus diagnosis, if the bus has such a feature (like PROFIBUS DP).

9.14 ErrorProgramcounter

Description

Every script command executed results in an error code. If an error occurs the position of the last error is to be determined by this parameter.
Use this command only for debugging purposes.

9.15 EthernetDestinationPort

Description

An Ethernet communication always needs a port number; some port numbers are defined by RFC's like Port 80 for http, but some numbers are free for custom usage.

Command

Set (EthernetDestinationPort, 2000) ;

Values

The port number may be in a range from 1 to 655535. Valid free port numbers are from 2000 upwards.

Defaults

No port number is the default number. If a UDP Message is received, the answer is at the same Ethernet port.

9.16 EthernetSourcePort

Description

A Ethernet communication always needs an port number; some port numbers are defined by RFC's like Port 80 for http, but some numbers are free for custom usage.
If the source port is different from 0 the gateway only reacts to messages for this port.

Command

Set (EthernetSourcePort, 2000) ;

Values

The port number may be in a range from 1 to 655535. Valid free port numbers are from 2000 upwards.

Defaults

No port number is the default number.
If a UDP Message is received, the port number is to be determined by this parameter.

9.17 FieldbusID

Description

It is possible to override the fieldbus ID determined by the value programmed with WINGATE or selected by switches.

Commands

Set
Get
SetByVar

Defaults

A default value for this parameter does not exist.

Example

```
...
Set ( FieldbusID, 5); // Sets Id to 5
...
```

Type

The type for the FieldbusID is long. If you use this parameter with the SetByVar command it is absolutely necessary to use a long variable for the FieldbusID.

9.18 LonProgramID

Syntax

Set (LonProgramID =v1, Value=v2) ;

Description

Sets the LonProgramID to the value.

Program ID: it consists of eight 2-digit hex values, separated by colons (no spaces). The first hex digit identifies the program ID format. If the first digit is 7 or less, the format is an ASCII string, typically with the name of the program.

The first two ASCII char for this Lon Script UNIGATE is fixed to "SC" followed by the value v2.
SC = 53h, 43h
The first digit is 5.

Example

```
...
Set ( ProgramID, 123456 ) ;

// You see the following string in the XIF file
// 53:43:31:32:33:34:35:36
// it means: SC123456
...
```

Values

The length of value 0 .. 999999

Errors

If a parameter or a value is not supported by the gateway it will show an error and stop all actions. Script execution with such a problem makes no sense.

9.19 ModbusRTUTimeout

Description

When a Modbus frame is sent (with ModbusDataExchange) the gateway expects a response. If this response is not received within the time specified by this parameter a script timeout occurs. The script should check the result of a ModbusDataExchange before evaluating the response data.

Command

Defaults

9.20 ModbusSlaveAddress

Description

For a Modbus RTU Slave protocol it is necessary to define a Modbus slave address. This parameter is used to set the address. A valid Modbus slave address is in range 1-247.

Command

Set
SetByVar
Get

Return codes

The return code is Parameter_Range_Error if the Modbus Slave Address is not in the valid range.

Defaults

There is no default value for the ModbusSlaveAddress.

9.21 MPIDBFetch

Description

This parameter describes the number of the data component within the PLC to be read out by the gateway.

Defaults

Default value for this parameter is 0. If the data component 0 does not exist in the PLC the behavior is dependent on the PLC.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.22 MPIDBSend

Description

This parameter describes the number of the data component within the PLC to be written.

Defaults

Default value for this parameter is 0. If the data component 0 does not exist in the PLC the behavior is dependent on the PLC.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.23 MPIDWFetch

Description

This parameter is the address of the PLC's data component to be written. The number of the PLC's data component is given by the DB Fetch Parameter.

Defaults

Default value is 0.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.24 MPIDWSend

Description

This parameter is the address of the PLC's data component to be written.

Defaults

The default value for DW Send is 0. Normally counting starts with 0 so no problem should occur.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.25 MPIFetchOn

Description

This parameter describes the time in milliseconds between to fetch commands automatically performed by the gateway. If this parameter is 0 no fetch is performed by the gateway.

Defaults

Default value for this parameter is 0.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.26 MPIFetchType

Description

This parameter describes the kind of data to be fetched by the gateway. If this parameter is 0 no fetch is performed by the gateway.

Defaults

Default value is 0.

Values

No send type	= 0
Data module	= 68
Input bytes	= 69
Marker bytes	= 77
Meter cells	= 90
Absolute addresses	= 83
Extended data module	= 88
Output bytes	= 65
Periphery bytes	= 80
Time cells	= 84
System addresses	= 66

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.27 MPIGapFactor

Description

No description available at the moment.

Defaults

Default value is 5. Normally it should not be necessary to change this value.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.28 MPIMax.Station

Description

No description available at the moment.

Defaults

Default value is 31. Normally it is not necessary to change this value.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.29 MPIPartnerAddress

Description

Command

Defaults

No default value for this parameter.

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.30 MPISendType

Description

Command

Defaults

Note

This command affects MPI only. All other gateways simply ignore this parameter and do not produce an error.

9.31 Parity

Description

Every character sent and received at the RS-interface has a parity bit. It is possible to select the kind of generation for the parity.

Defaults

The default value for parity is none.

Values

Possible values are:
none
even
odd
mark
space

Note

Parity mark and space are available from script revision 13 on.

9.32 ProductCode

Description

It is possible to set a fixed product code for a script gateway.
If this value is set to 0 the gateway calculates its product code by 256 * consumed size + produced size.

Command

Set

Defaults

Default value is zero; the gateway calculates its product code.

Note

This parameter and its value affect DeviceNet only.
All other busses simply ignore this command and will not produce an error on executing this statement.

9.33 RSInCharacter

Description

Determines the number of characters in the RS input buffer.The result value is a word value.
The number of bytes in a serial input buffer may not exceed 255 in a standard device; some devices may have smaller or larger input buffers. Please refer to the device's specification for details.

Command

Get

Note

This parameter is read only.

9.34 RSOutFree

Description

The RS output buffer may only get hold of 255 bytes. If you like to send more data, you need to wait until the number of bytes available in the device's output buffer is big enough to keep the new message.
Use this parameter to check if the output buffer has enough space for your data.

Command

Get

Note

Data in the output buffer is automatically sent by the device's operating system.
Some devices will have a larger input buffer (>1kByte). For those devices use the parameters RSOutFree16 instead.

9.35 RS_State_LED

Description

All UNIGATE® SC devices - not depending of the bus type - have an LED called RS_State_LED. It is possible to set the function of this LED by this script parameter.

Command

Set

Defaults

By default the LED is off.

Values

The parameter take one of the following values:
- off
- staticgreen
- staticred
- greenflashing
- redflashing
- redgreenflashing

Note

The function of the LED may be overridden by a system error or a user error.
This parameter does not affect devices of our UNIGATE® IC series.

9.36 RSSwitch

Description

Determines the position of the switches S4 and S5 in a device which supports those switches (this excludes all IC's).
S4 is in the high nibble, S5 is in the low nibble of the byte.

Command

Get

Example

```
-----
// assume s4 = "1" and S5 = "5"
var v1: byte;
Get ( RSSwitch, v1 ) ;
// the value v1 is now 0x15, which means the high nibble is1 (=S4) and
// the low nibble is 5 (= S5). Use logical AND and SHIFT operations to
// extract the single values.
-----
```

Note

This parameter is read only.
You get an error when attempting to write this value.

9.37 RSType

Description

Determine the kind of the RS interface.
Possible values range from 0 to 2, reflecting RS232, RS422 and RS485. If you like to use a UNIGATE® SC in RS485 mode or if you like to use a UNIGATE® IC's TE pin set the corresponding RS-type.

Command

This command is read only.

Note

A default value does not exist; the value always reflects the real hardware.
An error occurs if a device can not detect the hardware (e. G. UNIGATE® IC's).

Values

0 = RS232
1 = RS485
2 = RS422

9.38 SelectID

Description

The gateway consists of 4 LED's. Those LED's signed with SelectID / Error No are accessible by this parameter. Every binary number from 0 (all LED's off) to 15 (all LED's on) is possible.
Best is to set the value for this parameter as binary value.

Defaults

By default all LED's are off.

Example

```
...
Set SelectID to 0b1010;
// SelectID LED's No 8 and 2 are on. similar to :
// Set SelectID to 10;
// Set SelectID to 0x0A;
...
```

9.39 ShiftRegisterInputBitLength

Description

Specifies the exact bitlength of the connected hardware. The given value must fit the real hardware; otherwise you will get an error.

Commands

Set
Get

Defaults

The default value is 8 bit.

Example

```
...
Set (ShiftRegisterOutputbitLength, 12); // Now the size is 12 bits
...
```

9.40 ShiftRegisterInputType

Description

Specifies the type of hardware connected to the device.

Commands

Set
Get

Defaults

Example

. . .
. . .

9.41 ShiftRegisterOutputBitLength

Description

Specifies the exact bitlength of the connected hardware. The given value must fit the real hardware; otherwise you will get an error.

Commands

Set
Get

Defaults

The default value is 8 bit.

Example

```
...
Set ( ShiftRegisterOutputbitLength, 12) ; // Now the Bit size
is 12 bytes
...
```

9.42 ShiftRegisterOutputType

Description

Specifies the type of hardware connected to the device.

Commands

Set
Get

Defaults

Example

. . .
. . .

9.43 StartBits

Description

Number of startbits for serial communication.

Defaults

The default value for the startbit is 1.

Values

At the moment only 1 startbit is allowed.

9.44 StopBits

Description

Every character sent over a serial connection is framed in start and stop bits. All devices support 1 or 2 stopbits.
If you are using 1 stopbit a character sent by the partner with 2 stopbit is received without an error.

Command

Set

Defaults

The default is 1 stopbit.

9.45 Timer

Description

An internal timer runs with a frequency of 1 millisecond. It is possible to read and to write the timer.

Commands

Set baudrate
Get baudrate

Defaults

The timer value is 0 at gateway startup. No other value is available as default.

Values

A timer can have values from 0 to 2^32-1. It is a long value (4 bytes). Assign this value only to long variables.

Example

. . .
. . .

Note

A timer value is a long value. This means a timer starting by 0 runs to 2^32-1 and then runs over to 0 again. The timer value increases every millisecond, so a runover occurs every 49 days. To preserve such a behavior the timer value should be set to 0 before use. If you use a timer value the condition always should be a greaterEqual condition.

9.46 WarningTime

Description

Under some circumstances the gateway may show an error. This indication may be a real hardware error, a script error or a user defined state (a visual note). Normally it is not necessary to change this time.
Changing this parameter does not have any effect on active errors.

Defaults

Default is 60000 = 60 * 1000 ms = 1 minute.

10 Miscellaneous

10.1 Return codes

This may be an incomplete list of all errors generated. After a script line is executed the result is available by the parameter Error code. The Protocol Developer software shows an error description for every error.

Only return codes from 0x10 to 0x7F are recognized by the OnError command. All other errors are treated as "hints" instead of a real error.

A list that contains all Errors with description can be found in the online help Protocol Developer under ”Return Codes”

10.2 Script revisions

This is a informative list of script revisions and implemented features. Functions which are not listed here are implemented since initial script revision. This list may have errors, or may be incomplete, and function names may not directly correspond to function names available in the Protocol Developer software.
A topical list of Script revisions can be found in the online help Protocol Developer ”Script Revisions”.

10.3 Script execution

Every script command is executed by the device’s script interpreter. After a command is executed a return code (= errorcode) is available and can be read by the Get (ErrorCode, x) command.

10.4 Bus Types or Device Types

Device	Decimal type value	Hexadecimal value
UNIGATE SC PROFIBUS DP	103	0x67
UNIGATE IC PROFIBUS DP	173	0xAD
UNIGATE SC CANopen®	105	0x69
UNIGATE IC CANopen®	175	0xAF
UNIGATE SC DeviceNet	104	0x68
UNIGATE IC DeviceNet	174	0xAE
UNIGATE SC Interbus	106	0x6A
UNIGATE IC Interbus	176	0xB0
UNIGATE SC Ethernet 10	151	0x97
UNIGATE IC Ethernet 10	177	0xB1
UNIGATE SC LONWorks (512)	157	0x9D
UNIGATE SC MPI	152	0x98
UNIGATE SC Arcnet	150	0x96
UNIGATE IC Fast Ethernet	178	0xB2
UNIGATE SC Fast Ethernet	159	0x9F
UNIGATE IC LONWorks	179	0xB3
UNIGATE IC RS	180	0xB4
UNIGATE SC LONWorks S (62)	162	0xA2
UNIGATE CL PROFINET	167	0xA7
UNIGATE IC PROFINET	189	0xBD
UNIGATE CL EtherNet/IP	165	0xA5
UNIGATE IC EtherNet/IP	187	0xBB
UNIGATE CL Powerlink	188	0xA6
UNIGATE IC Powerlink	166	0xBC
UNIGATE CL EtherCAT	168	0xA8
UNIGATE IC EtherCAT	190	0xBE

There is no difference in bus type between a UNIGATE® SC and a UNIGATE® SC (Debugversion).