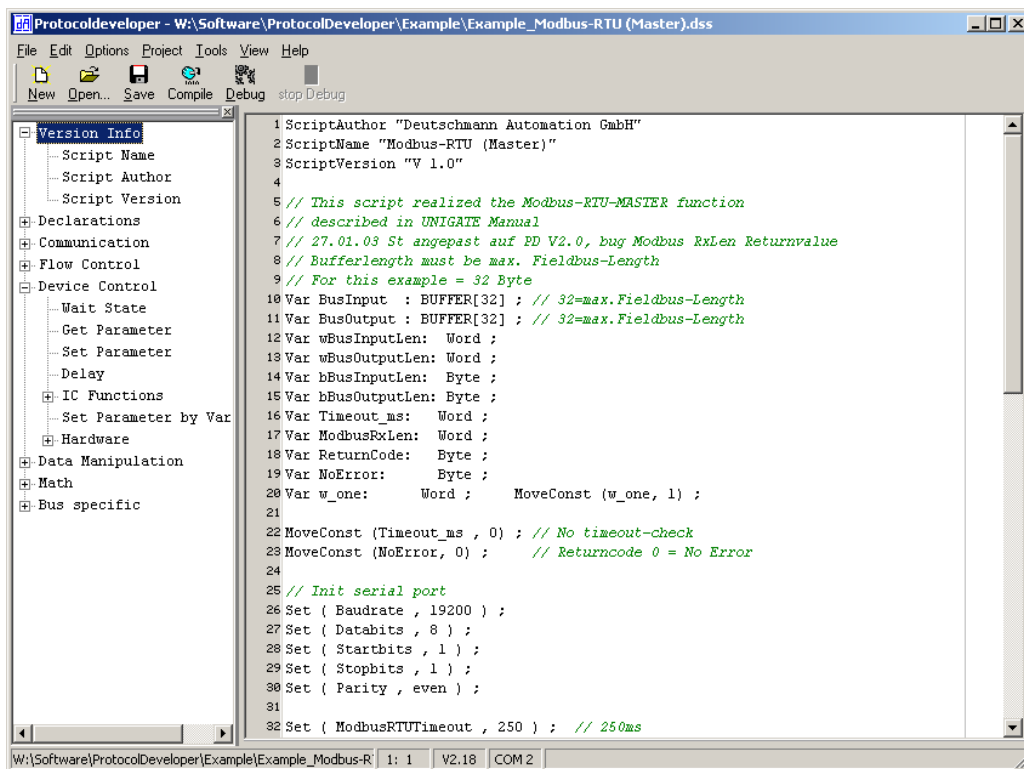




**Deutschmann Automation**

Cam Controls | Fieldbus Gateways | Industrial Ethernet Products

# Instruction manual



# Protocol Developer

Article No.: V3183E

Deutschmann Automation GmbH & Co. KG

Carl-Zeiss-Straße 8 D-65520 Bad Camberg ☎ +49-(0)6434 / 9433-0 📠 +49-(0)6434 / 9433-40

eMail: mail@deutschmann.de Internet: <http://www.deutschmann.de>



<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>What is a Script?.</b>	<b>8</b>
2.1	Decision for an own Script language	8
2.2	Memory efficiency of the programs	8
<b>3</b>	<b>Hardware</b>	<b>9</b>
3.1	UNIGATE SC in Debug-version	9
3.2	Debug mode	9
3.2.1	Setting the Debug mode	9
3.2.1.1	Switch-on message of the Gateway	9
3.3	UNIGATE IC	10
<b>4</b>	<b>What can you do with a Script device?</b>	<b>11</b>
4.1	Independence of buses	12
4.2	Further settings at the Gateway	12
4.3	The use of the Protocol Developer	12
4.3.1	Main window	13
4.4	Menu structure	13
4.5	The Debugger	15
4.6	Programming Scripts	18
4.6.1	Spelling of Script commands	18
4.6.1.1	Numbers	18
4.6.1.2	Texts	19
4.6.1.3	Comments	19
4.6.1.4	Label	19
4.7	About the Script development	19
4.8	Special rules for Scripts	20
4.9	Debugging	20
4.9.1	Proceeding	20
4.9.2	Debug commands.	20
<b>5</b>	<b>Description of the Script Program Tool</b>	<b>21</b>
5.1	Manual mode	21
5.2	Automatic mode	21
5.2.1	Setting up the automatic mode	21
<b>6</b>	<b>Appendix</b>	<b>22</b>
<b>7</b>	<b>Quick start</b>	<b>25</b>
7.1	Step-by-step	26
7.1.1	Step 1	26
7.1.2	Step 2	26
7.1.3	Step 3	26
7.1.4	Step 4	27
7.1.5	Step 5	28
7.1.6	Step 6	29

7.1.7 Step 7	29
7.1.8 Step 8	30
<b>8 Commands (selection of commands)</b>	<b>31</b>
8.1 BusStart	31
8.2 CalculateByte	32
8.3 CalculateWord	33
8.4 Call	34
8.5 Checksum	35
8.6 Convert	36
8.7 Copy	37
8.8 Delay	38
8.9 DIN19244DataExchange	39
8.10 ExchangeModbusRTUMaster	40
8.11 FillMemory	41
8.12 GetParameter	42
8.13 If - then - else	43
8.14 Init3964R	44
8.15 InitCommunicationChannel	45
8.16 Jump	46
8.17 Label	47
8.18 LONSelfDocString	48
8.19 MoveConst	49
8.20 ReadBus	50
8.21 ReadModbusSlave	51
8.22 Receive3964R	52
8.23 ReceiveSomeCharRS	53
8.24 ReceiveSpecialCharRS	54
8.25 Return	55
8.26 ScriptAuthor	56
8.27 ScriptName	57
8.28 ScriptRevision	58
8.29 Send3964R	59
8.30 SendRS	60
8.31 Set	61
8.32 SetByVar	62
8.33 SetLonMapping	63
8.34 Stop	64
8.35 VariableDeclaration	65
8.36 Wait	66
8.37 WaitBusChange	67
8.38 WriteBus	68

8.39	WriteModbusSlave . . . . .	69
<b>9</b>	<b>Parameters (selection of parameters) . . . . .</b>	<b>70</b>
9.1	3964RPriority . . . . .	70
9.2	AvailableBusData . . . . .	71
9.3	Baudrate . . . . .	72
9.4	BusBaudrate . . . . .	73
9.5	BusDataChanged . . . . .	74
9.6	BusInputsize . . . . .	75
9.7	BusOutputSize . . . . .	76
9.8	BusTimeout . . . . .	77
9.9	BusType . . . . .	78
9.10	ChecksumCalculationMethods . . . . .	79
9.11	CommunicationChannel . . . . .	80
9.12	DataBits . . . . .	81
9.13	ErrorCode . . . . .	82
9.14	ErrorProgramcounter . . . . .	83
9.15	EthernetDestinationPort . . . . .	84
9.16	EthernetSourcePort . . . . .	85
9.17	FieldbusID . . . . .	86
9.18	LonProgramID . . . . .	87
9.19	ModbusRTUTimeout . . . . .	88
9.20	ModbusSlaveAddress . . . . .	89
9.21	MPIDBFetch . . . . .	90
9.22	MPIDBSend . . . . .	91
9.23	MPIDWFetch . . . . .	92
9.24	MPIDWSend . . . . .	93
9.25	MPIFetchOn . . . . .	94
9.26	MPIFetchType . . . . .	95
9.27	MPIGapFactor . . . . .	96
9.28	MPIMax.Station . . . . .	97
9.29	MPIPartnerAddress . . . . .	98
9.30	MPISendType . . . . .	99
9.31	Parity . . . . .	100
9.32	ProductCode . . . . .	101
9.33	RSInCharacter . . . . .	102
9.34	RSOutFree . . . . .	103
9.35	RS_State_LED . . . . .	104
9.36	RSSwitch . . . . .	105
9.37	RSType . . . . .	106
9.38	SelectID . . . . .	107
9.39	ShiftRegisterInputBitLength . . . . .	108

---

9.40	ShiftRegisterInputType . . . . .	109
9.41	ShiftRegisterOutputBitLength . . . . .	110
9.42	ShiftRegisterOutputType . . . . .	111
9.43	StartBits . . . . .	112
9.44	StopBits . . . . .	113
9.45	Timer . . . . .	114
9.46	WarningTime . . . . .	115
<b>10</b>	<b>Miscellaneous . . . . .</b>	<b>116</b>
10.1	Return codes . . . . .	116
10.2	Script revisions . . . . .	116
10.3	Script execution . . . . .	116
10.4	Bus Types or Device Types . . . . .	117

## 1 Introduction

Our customers are looking for flexible solutions: Rightly. That was reason enough for us to also think about such a solution in the Gateway market.

We therefore took all known demands on a Gateway into consideration and from that superset we tried to find a simple solution for all problems. We realized very quickly that one single setting of the Gateway would not be sufficient to achieve the vast number of possible applications, that is already done with WINGATE®. However, an implementation that we offer to our customers is often too expensive - a solution would be to enable the customer to carry out the programming himself. If now a customer would try to write his own C-program, he would be faced with the problem to program bus-accesses. Knowledge of the single fieldbus controllers etc. is required. Therefore we offer an intermediate solution. The only thing the customer has to do is to process the data of the fieldbus. He does not have to bother about the specific features of the buses. Also the customer does not have to possess knowledge of programming languages, but he generates a Script by means of a Windows-tool.

A general basic knowledge in programming, however, is required. Examples that are given in this introduction are extracts from Scripts, that are not necessarily executable the way they are printed in this manual, as possible preconditions are not fulfilled. These examples are to be understood as basic statements.

## 2 What is a Script?

A Script is a sequence of commands, that are executed in that exact order. Because of the fact that also mechanisms are given that control the program flow in the Script it is also possible to assemble more complex processes from these simple commands.

The Script is memory-oriented. It means that all variables always refer to one memory area. While developing a Script you do not have to take care of the memory management though. The Protocol Developer takes on this responsibility for you.

### 2.1 Decision for an own Script language

Conditional on the hardware an existing Script language as for instance JavaScript, TCL, Perl, Python cannot be run on the Gateway. As the operating system is not a Microsoft® operating system, also Visual Basic and its variants are out of question. Another point also argues against these languages: None of the above languages is designed for the 'embedded' area.

All these points lead to one possible solution:

A language,

- that is designed exactly for the Gateway / fieldbus area
- that takes all characteristics of the Gateway into consideration
- that is simple
- that is low on memory
- that can be executed by the Gateway efficiently

These points altogether define our language and also explain the restrictions of the language at the same time.

### 2.2 Memory efficiency of the programs

A Script command can carry out e. g. a complex checksum like a CRC-16 calculation via data. For the coding of this command only 9 byte are required as memory space (for the command itself). This is only possible when these complex commands are contained in a library.

A further advantage of this library is, that the underlying functions have been in practical use for a couple of years and therefore can be described as 'void of errors'. As these commands are also present in the native code for the controller, at this point also the runtime performance of the Script is favorable.



## 3 Hardware

### 3.1 UNIGATE SC in Debug-version

On principle the Debug hardware does not differ from the one of the standard Gateway. In addition to the regular hardware a special variant is available which, however, is only required for the development of a Script. Due to technical reasons this extended hardware is not available for all buses, but a development on another than the target hardware can be made.

Compared to the standard Gateway, this Debug Gateway features an additional RS232-interface, which is available at the model with 9-pol. D-SUB connector. This Debug-interface itself is always operated with 9600 baud, no parity, 8 data bits and 1 stop bit. Apart from that there are no further differences neither in the software nor in the hardware.

### 3.2 Debug mode

The Debug mode is important for the development of a Script. In this mode the Script will only be executed as far and in the extent as indicated by the user. Furthermore, the currently running Script can be stopped by the user and be continued step by step or it can be continued at another position as well. Contents of variables can be observed, so that the processing of Script commands can be examined. Data coming in from the bus or the RS-interface can be displayed. The most important tool would probably be the single-step mode and the possibility to set Break-points as well as the possibility to read out the current memory (variables).

#### 3.2.1 Setting the Debug mode

When starting the Gateway the switch Interface (at UNIGATE SC) is supposed to be adjusted to RS232. After the start the device outputs a binary "0" (0x00) at the Debug interface. The device is in the Debug mode if this is answered with a O (0x4F) within 0.5 s.

The switches S4 and S5 as well as the RS-switch can be brought to any position if required.

For the user himself, all Debug commands are integrated into a convenient surface, that makes it easier to develop Script.

##### 3.2.1.1 Switch-on message of the Gateway

When switching on the device it issues a switch-on message on the RS232-interface (standard interface) provided the device is in the config mode (which means the config jumper is set at UNIGATE IC or S4 and S5 are set to position "FF" at CL + SC).

**RS-PB-SC D(232/485) V5.0[6] (c)dA Switch=0xFFFF Script=Empty SN=12345678**

Config mode...

**RS-PB-SC** means that the device is a Profibus Script Gateway.

**D** means that the connection is a 9-pol. DSUB.

**(232/485)** means that the device features the RS-interfaces RS232 and RS485. As an alternative the designation (232/485) could be mentioned here at SC. AT CL = (232/422/485), at IC the interface is not indicated.

**V5.0:** Software revision of the device; firmware is V5.0

**[6]:** Script revision 6

**(c)dA Switch=0xFFFF:** Copyright indication and switch position of all 4 rotary switches. At the different buses this message varies slightly. At IC the position of the switch is omitted.

**Script=Empty:** The device contains an empty Script. The designation of your Script will be here. The designations have to be at the beginning of the Script and they have a maximum length of 32 byte.

In addition an author as well as a version of the Script, that can also be displayed here can be added at this place.

### 3.3 UNIGATE IC

In addition to the UNIGATE SC Deutschmann Automation is also offering the UNIGATE IC.

The UNIGATE series IC includes all analogue and digital components that are required for a fieldbus implementation. Processor, flash memory, RAM, Fieldbus ASIC and all analogue components as well as the opto-coupler and voltage supply are being combined on a small face. The IC-Gateway also carries out the complete fieldbus communication. The difference between the serial interface and the interface of the UNIGATE SC is that at the UNIGATE IC no drivers are connected and the levels are TTL-levels.

**Please note that the Gateways UNIGATE CL, IC and SC feature the same Script functions, unless differences are explicitly pointed out. The expression „Gateway“ used in this instruction manual stands for UNIGATE CL, UNIGATE IC and for UNIGATE SC as well.**

Notes on the UNIGATE IC can be found in the instruction manual UNIGATE IC and on our website at <http://www.deutschmann.de>.

## 4 What can you do with a Script device?

Our Script devices are in the position to process a lot of commands. In this case a command is always a small firmly outlined task. All commands can be put into classes or groups. A group of commands deals with the communication in general. This group's commands enable the Gateway to send and receive data on the serial side as well as on the bus-side.

The command groups are:

<b>Declarations</b>	Variable declaration
<b>Flow Control</b>	Subfunction calls, jumps, branches
<b>Math</b>	Mathematical functions Data conversions
<b>Communication</b>	Send and receive data
<b>Device Control</b>	Set and read parameters. Exemplary the baud rate for the serial interface is mentioned here.
<b>Bus specific</b>	Here the commands are placed that enter bus-specific values. We consciously used these commands as rarely as possible, so that the Scripts remain compatible.
<b>Version Info</b>	Text, issued by the Gateway in its switch-on message. These commands do not have a direct influence on the Script itself and also at the runtime they are of no account.

### Data Manipulation



**Please note that no detailed description of commands is given here; the commands are described in the Online Help.**

Please note that at this point no detailed description of the commands is given; the commands are described in the On-line help.

The amount of tasks, that can be processed with it is virtually endless.

Scripts,

- that over and over automatically acquire data from one participant at the serial interface, edit the data and then present the edited data in the bus
- that carry out actions only in case the bus data changes
- that carry out time-controlled actions
- that inform of the communication states
- that exchange data between 2 serial participants (RS485) and present the state in the bus are conceivable.

By means of this short enumeration it becomes clear that the Scripts are a flexible solution to your problems. Data can be processed, converted and arranged on both sides (on the RS-side and the bus-side as well). That way the Script basically offers the chance to cope with all requirements.

Problems are only to be expected in few cases:

- Is your requirement extremely time-critical? Because the Script is interpreted, the runtime performance is not as favorable as a direct implementation.
- Depending on the protocol that is to be handled, however, a reaction time of few milliseconds can also be achieved with the Script, that will absolutely do for most applications.

## 4.1 Independence of buses

Basically the Scripts do not depend on the bus, they are supposed to operate on. It means that a Script which was developed on a Profibus Gateway can also be operated on an Interbus without changes, since the functioning of these buses is very similar. In order to also process this Script on an Ethernet Gateway, perhaps further adjustments have to be made in the Script, so that the Script can be executed reasonably.

There are no fixed rules how which Scripts have to operate properly. When writing a Script you should take into account on which target hardware the Script is to be executed, so the necessary settings for the respective buses can be made.

## 4.2 Further settings at the Gateway

Most devices require no further adjustments, except for those made in the Script itself. However, there are also exceptions to it. These settings are made by means of the software WINGATE. If you know our UNIGATE-series, you are already familiar with the proceeding with it. An example is the adjustment of the IP-address and the net-mask of an Ethernet-Gateway. These values have to be known as fixed values and are not available for the runtime. Another reason for the configuration of the values in WINGATE is the following: After an update of the Script these values remain untouched, i. e. the settings that were made once are still available after a change of the Script.

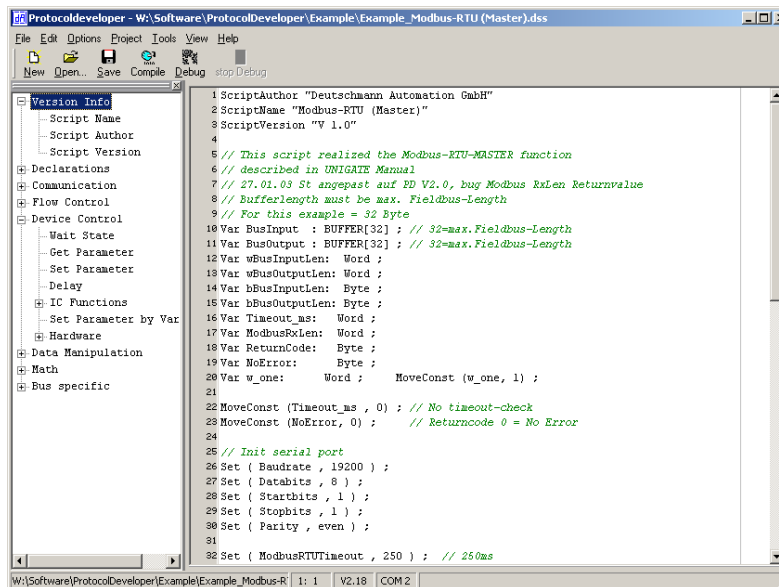
Only this way it is also possible that the same Script operates on different Ethernet-Gateways, that feature different IP-addresses.

## 4.3 The use of the Protocol Developer

The Protocol Developer is a tool for an easy generation of a Script for our Script Gateways. Its operation is exactly aimed at this use. After starting the program the Script that was loaded the last time is loaded again, provided that it is not the first start.

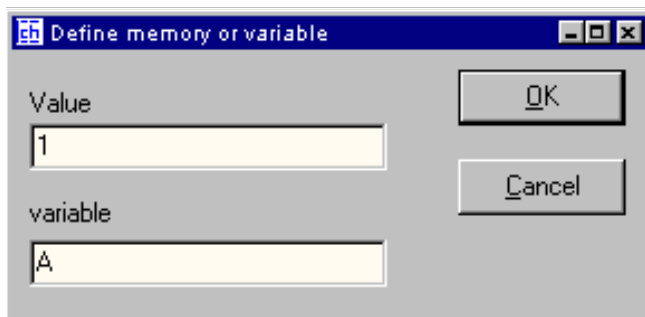
Typical for Windows Script commands can be added by means of the mouse or the keyboard. As far as defined and required for the corresponding command, the dialog to the corresponding command is displayed, and after entering the values the right text is automatically added to the Script. The insertion of new commands by the Protocol Developer is carried out in a way that existing commands will not be overwritten. Generally a new command is inserted in front of the one where the cursor is positioned. Of course the commands can also be written by means of the keyboard or already written commands can also be modified.

### 4.3.1 Main window



The actual writing of a Script is made in the main window. From the list in the left part of the window you can “take” a command by means of the mouse and insert it into the Script on the right side. If a dialog is available for the respective command, it is called before the text is inserted. Now variable names etc. can be entered, that will be available in the Script then. At this point you do not have to worry about the spelling of Script commands, the Protocol Developer will support you here.

Example for a dialog to a command.



After confirming the dialog with OK, the Protocol Developer generates the code for this command. In the example this would be the code "MoveConst (A, 1);". This code can also be modified manually.

## 4.4 Menu structure

### Menu File

In the File Menu all menu options that are necessary for the files can be found.

### New

With File New a new editor file is generated. The new file does not contain a Script and is completely empty. When such a file is compiled, only the basic code for a Script is generated.

**Open**

With File Open an existing file is opened. The file already has to exist. A new file is to be generated with File New.

**Save**

With File Save a new file is saved on a data carrier. In case a file does not have a name so far, as it was generated with File New, then the file Save as Dialog is automatically opened.

**Save as**

With this menu option a file can be saved under a different name than the one that was used before.

**Save compiled file**

This file is available as source code. With Save compiled File the source code is compiled once again. When the file was successfully compiled it is saved as binary file. This file can be transferred to a Script Gateway with other tools (WINGATE or SPT- ScriptProgramTool).

**Print**

With Print the complete source code of a program can be printed. At present the print cannot be restricted to certain pages.

**Exit**

With Exit the Protocol Developer is quit. In case the current file is not stored, then you are prompted to save the file.

**Menu Options  
Settings**

With Settings the underlying settings of the Protocol Developer can be adjusted. For instance the adjustment of the serial interface belongs to it.

In addition, all files that contain dialogs, commands etc. are created here. However, you should never process this list without being asked.

**Menu Project  
Compile**

The current Script, that is in the Protocol Developer is translated. In case of an error the compiler will indicate, where the error was detected and what kind of error it is. Then you can specifically process this point in the editor.

**Debug**

The Debugger is started with this. However, the Script has to be syntactically alright for it and a Gateway in Debug mode has to be connected to the interface. See also chapter Debugging.

**Menu View****Editor**

The editor is displayed.

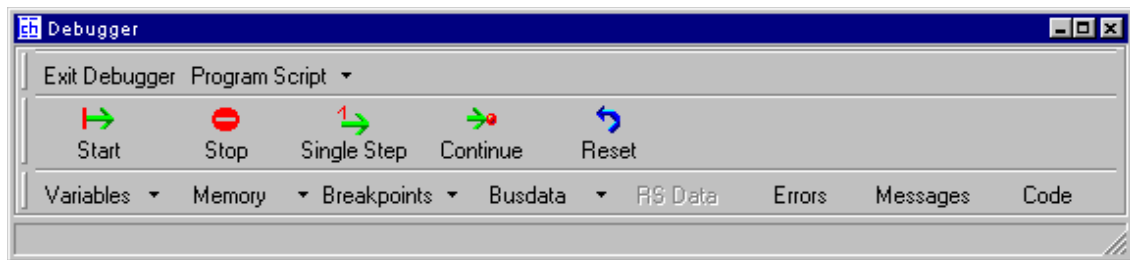
**Internals**

With Internals the internal values of the compilers can be looked at.

**Help**

When the Help File is in the same directory as the Protocol Developer, then the start page of the Help is displayed.

## 4.5 The Debugger



The main window of the Debugger enables to control a Gateway that is in the Debug Mode. The window offers the interface for operating and controlling the Debugger or the Debug Gateways.

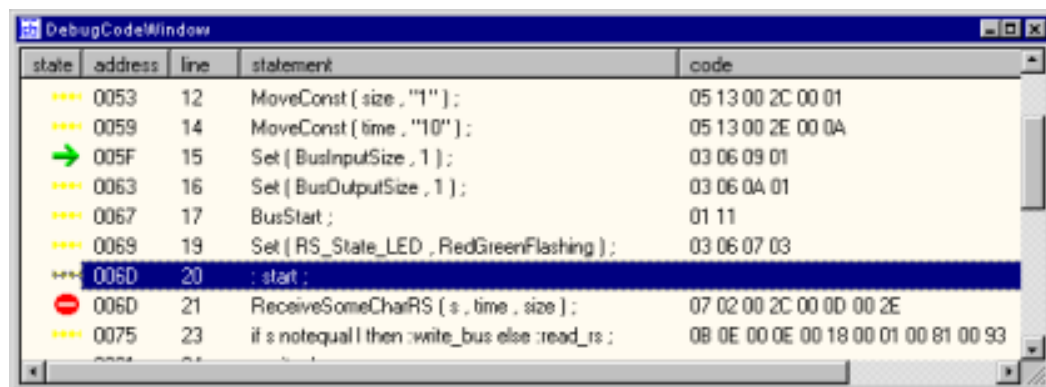
### Exit Debugger

The Debugger window is closed.

### Program Script

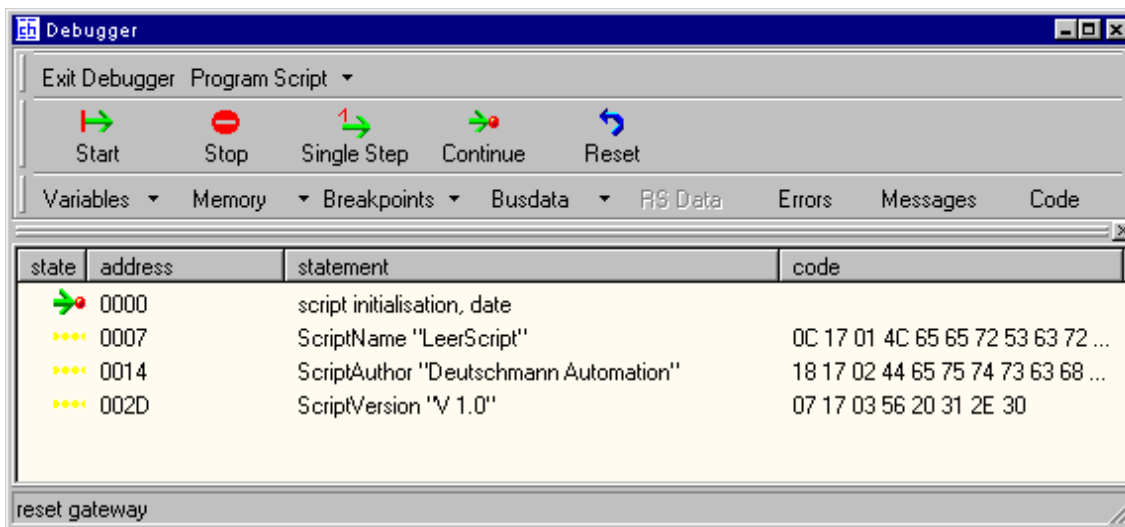
The Script that is in the Debugger at this time is permanently programmed into the Gateway (UNIGATE IC, UNIGATE-SC), so that it will be still available after a new start of the Gateway. The Debugger is now in the position to find out if a Script is already programmed and so every time when it detects a new start of the Debug Gateway the Script that was processed the last time will be sent to the device. With this proceeding it is guaranteed that the Debug Gateway always contains the Script from the Code window.

### The Code window



In the Code window the Script itself is displayed as a version that was reviewed by the Compiler. In the Code window you can set Breakpoints, start Scripts etc. Here you follow the course of your Script and watch the behavior of the Script and your application in order to detect and repair potential logic errors in the Script.

Usually the window is docked to the main window, however it can also be disconnected from it.



### Starting a Script

After the Debugger is started, usually the Script has the execution address 0000, it stands at the very beginning. A green arrow in the column State indicates the current position. With "Start" you are now in the position to execute the Script yourself. As soon as the Gateway is into operation and the processing of the Script is started, then the switches for "Single Step" and "Continue" are hidden.

### Stopping a Script

In order to stop the Script you have to wait until the Script comes across a Breakpoint and stops the further processing of Script commands with it or you can also stop the Script with the „Stop“ key. In case a command which takes quite a while is executed at the time, the Gateway is not stopped. The current command is completely executed. It seems the Debugger does not work any more or the Gateway does not react.

Example: the command "delay(10000);" waits for 10 seconds. In case you interrupt this command by "Stop", the command will be finished anyway, i. e. the control is passed to the debugger after a period of the 10 seconds. Please note that there are also commands, that do not have a fixed runtime. These commands cannot be interrupted by Stop.

### Reset of a Gateway

With Reset the Gateway is brought to its original condition, it is in after switching on. The device's whole memory is preset with 0, the execution position for the Script command is address 0000.

### Setting a Breakpoint

In the Code window please select that line, in front of which the execution of the Script is supposed to be stopped. Now a Breakpoint can be switched on or off through the context sensitive menu (right mouse button). Please note that all Breakpoints are deleted after a reset.

### Changing the execution position

In the Code window please select that line, where you want to continue the execution of the code. By means of the context sensitive menu you can place the "ProgramCounter" on this position now. You should only do so if you are able to keep track of all side effects that might occur because of this. This command is useful in case you are for instance developing without bus and you want to skip the command "wait(Bus\_Active)".



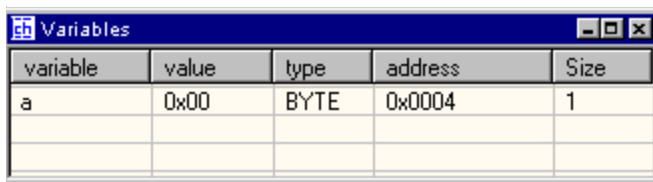
### Variable display

From the Debugger window the display of the window itself can be controlled. Variables can be displayed and this variable's type of representation can be set. A variable can be deleted again. Also all other variables can be deleted or all variables of a Script can be displayed.

Besides there is the possibility to update all values of the variables manually. Usually after the Gateway is stopped, all variables are updated automatically.

In order to add variables you can use the key in the Debugger or the context sensitive menu of the variable window. Only those variables can be added, that are also included in the current Script. In case variables from earlier Debug sessions are included only the name without value is displayed. These variables should be deleted.

With a double click variables can be edited. Now you can change the variable's type of representation.



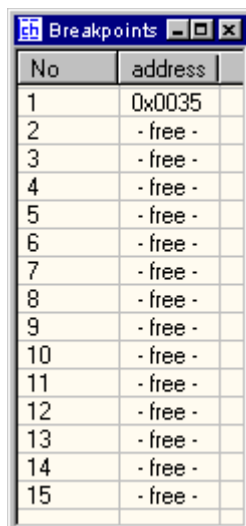
variable	value	type	address	Size
a	0x00	BYTE	0x0004	1

### Breakpoints

For the development of a Script, the Script Gateway is able to manage up to 15 Breakpoints. A Breakpoint is set on a line in a code by moving the marking to the corresponding line (or by selecting that line with the mouse) and a switchover is made either through the menu of the right mouse button or with the F5-key. A Breakpoint can only be set in case the Gateway is in the Stop state. If the Gateway is executing a Script that time, then the setting of a Breakpoint is without function.

A list of active Breakpoints can be called through the Debugger menu.

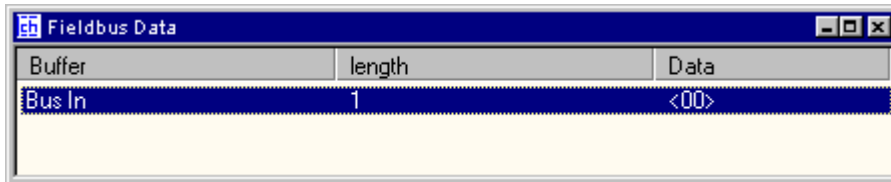
If a Breakpoint is set in the Script and the execution of the Script meets with a Breakpoint, then it will be stopped BEFORE the execution of the corresponding command. This line can be executed with Single Step, with Go the Script can be executed further from the stopping position on.



No	address
1	0x0035
2	- free -
3	- free -
4	- free -
5	- free -
6	- free -
7	- free -
8	- free -
9	- free -
10	- free -
11	- free -
12	- free -
13	- free -
14	- free -
15	- free -

### Bus Data Window

Here the current bus data can be monitored. They are updated after every Stop command. They can also be updated manually. However, this data is only available AFTER the execution of the first Script command. For a useful utilization of this window it is absolutely necessary that the bus is in same kind of operation as it is supposed to be used later.



### Error Window

The Gateway executes each Script command individually. On the execution itself a constellation might occur, that the command was executed, but still an error condition was reached.

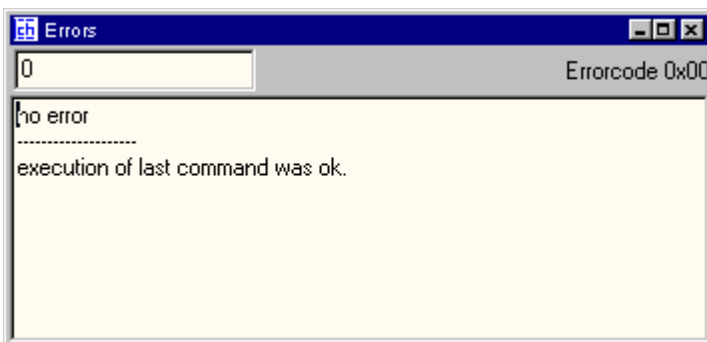
Example: Receipt of serial data with Timeout, then Timeout initiates an error, which, however, does not result in an abort of the Script. The Script has to continue evaluating this condition.

Each time the Gateway stops in the Debug mode, the most recent error state of the device is transferred to the PC. The Gateway executes a stop:

- after a Single Step.
- after a restart.
- when a break is reached.

An error code as well as a short description to this error is displayed in the window.

The Online help also contains the error codes.



## 4.6 Programming Scripts

### 4.6.1 Spelling of Script commands

No distinction is made between upper-case and lower-case letters.

#### 4.6.1.1 Numbers

There are different possibilities to write numbers; in decimal, binary and hexadecimal form. Besides, an explicit type conversion can be made, when a constant factor has to be converted into a specific format.

Example:

```
var b: word;                // B in use 2 byte in memory
MoveConst ( B, 257 ) ;      // B has the value 0101
MoveConst ( B, 0x110 ) ;    // B has the value 0110
MoveConst ( B, "AA" ) ;     // B has the value 4040
MoveConst ( B, #0x01#13 ) ; // B has the value 010D
MoveConst ( B, "A"#13 ) ;   // B has the value 400D
MoveConst ( B, 0b1111 ) ;   // B has the value 000F
```

#### 4.6.1.2 Texts

Texts are generally stated in double quotes. It is also possible to integrate special characters into the text. At the declaration time length checks are made, i. e. it is not possible to assign a 30-digit text to a variable with a length of 10 characters only. Binary characters can be attached to a string constant by indicating the decimal code.

Unlike e. g. in Pascal no lengths are stored in the texts, as the length is to be calculated by the Script itself.

#### 4.6.1.3 Comments

Comments can be added at any position in the Script. Comments are always introduced with a double slash and stretch down to the end of a row on principle. There is no other possibility, that comments extend over several rows, than to introduce each row as a new comment.

#### 4.6.1.4 Label

Basically all identifiers, that are structured correctly can be a label. There is no length restriction, however, only labels with 255 digits are treated with significance, which means that labels with more than 255 characters and whose first 255 characters are equal, are considered to be identical. The first character of a label is the colon, followed by any alphanumeric characters and the underscore.

### 4.7 About the Script development

Think about some initial considerations:

Is your Script meant for one bus only or is it supposed to be working for several buses?

Are the buses similar on principle or contrary?

If your Script is meant for one bus only, then you can take advantage of all properties of this bus.

If your Script is supposed to support several buses, a few things have to be considered:

Which bus, that is supposed to be supported has the minimum data capacity? Take this data capacity as a basis and structure the bus data so that all buses can work with it. That way you do not have to make a distinction between the Scripts in bus-specific parts.

Are the buses similar on principle or contrary?

Here those buses would be considered as similar buses, that are for instance Single-master buses where the master transfers data cyclically. Profibus, Interbus, DeviceNet in a restricted extent (in poll operation) rank among these buses.

Ethernet and CANopen® can be mentioned here as dissimilar buses.

Please describe the Script's duty:

A Script can only be converted successfully with a clearly outlined duty.

Please generate a structure in any familiar form, no matter if it is a graphical structure or a verbal description. Those procedures make sense in order to find logical errors in Scripts.

Please start with the coding of the single duties only now. If possible divide some duties into smaller unities in order to keep them clear and also to be able to check them correspondingly.

Always run the test of the Script piece by piece, remember or take down which properties have subfunctions.

## 4.8 Special rules for Scripts

The commands "ScriptName", "ScriptAuthor" and "ScriptVersion" have to be entered in the Script as first commands, so that they appear in the device's switch-on information. The Script dat is the Script's translation date and is entered automatically. If these commands are in any other position, then they are not recognized when the Gateway is initialized. They do not lead to a malfunction of the Script itself, though.

The order of commands does not matter; also only one command from this group can be used.

## 4.9 Debugging

In the programming the general error location in a project is described as Debugging. The Debugging can be seen on the Gateway in a restricted way because of the hardware, where the Script is executed. However, it still offers all necessary properties in order to see errors of logical structure. Indeed you should realize that the Protocol Developer is not in the position to detect errors automatically. It only offers the tools to do so. You still have to do the exact analysis of the things, the Script is doing.

The Protocol Developer can support you in some points.

### 4.9.1 Proceeding

First of all you have to develop a Script or, provided that you want to make yourself familiar with the functions of the Protocol Developer, you have to load an existing Script into the development environment. Syntactic errors must not be existing in the Script any more.

A Gateway capable for development has to be connected and it has to be in the Debug mode (for this see chapter 'Hardware').

The Script has to be stored.

Please start the Debugger through the menu or the button. Now the Protocol Developer will try to recognize the Gateway, translate the Script and transfer it to the Gateway. Not until these steps have been carried out successfully, the Debugger itself is started.

### 4.9.2 Debug commands

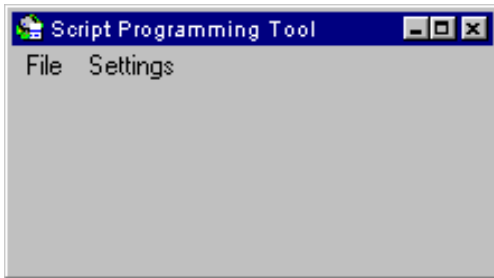
A lot of Debug commands are available, that can be carried out via the Gateway's diagnosis-interface. The commands are designed in a way that they can be executed from any terminal. Usually a "manual" operation is not necessary, as these commands are applied by the Debugger of the Protocol Developer. Besides, the Protocol Developer also carries out further actions, that make an error location in a program much easier, as for instance the direct observation of variables in different display formats, the easy change of the present execution position.

Additionally the environment also guarantees that always the currently valid row Code is displayed (Comments are not visible here and commands that stretch over several rows in the original code are reduced to one row here. Branch destinations are indicated.) The code that was generated by the Compiler can be seen, so that memory limits etc. can be revealed.

## 5 Description of the Script Program Tool

This tool was developed to program a Script that is there in translated form into a Gateway capable to execute a Script. A Script in the translated form can be generated by means of the Protocol Developer.

Basically 2 equipment options are available.



### 5.1 Manual mode

The program is called.

From the menu a file (a translated Script) is loaded and sent to the device (Per Dialog File...).

The program is closed (File Exit).

Settings can be made (COM 1.. COM 4).

### 5.2 Automatic mode

In the automatic mode the program is directly supplied with the file name of the translated Script, that is to be programmed.

An attempt to start the program is made. At the first start the serial interface the Scriptgateway is connected to is additionally determined interactively.

With the program start itself an attempt is made to load this file.

Then the program is trying to send the Script to the Gateway. When this operation can be carried out a progress display is presented, otherwise an error.

The program waits until the Gateway is reset after the Script was written and then terminates itself independently. For the programming only a click is required.

#### 5.2.1 Setting up the automatic mode

To set up the automatic mode a 'link' has to be generated in the Explorer (right mouse button, New -> Link). Then the Link itself has to be marked (click with the mouse).

The properties of the Link have to be adjusted (right mouse button, Edit -> Properties).

The name of the file is to be installed as parameter in the command line.

Example: ( SPT.EXE "FileName.gws.gwc")

Confirm the dialog, and that's that.

## 6 Appendix

### Listing of all Debug commands in alphabetic order

#### Break

With Break a Breakpoint is deleted or set. Setting a Breakpoint to one address:

B030020: the Breakpoint 03 is set to the address 0020, no matter if the Breakpoint existed before or not.

B030000: the Breakpoint 03 is deleted. Deleting a nonexisting Breakpoint does not have any consequences.

#### ChangeProgramCounter

The Program Counter (the current position of the Script) can be changed manually at a Gateway in Stop state. The indication of the new execution position is made in 4 Byte ASCII Hex:

P0020: the Program Counter is set to address 0020 (typically the beginning of the Script). With the next Go or Single Step the Script will be executed from this position on.

#### Download

The download of a program in the RAM is initiated with D, followed by a word Data (binary), that states the length of the following bytes, the data themselves in binary format and then the checksum (1 word binary). In case the download was successful the Gateway confirms with O, otherwise in case an error occurred, then an E is issued.

The checksum is the total of all data bytes.

#### Go

A Script can be executed from its beginning by executing a Go command from the address O on. Also any other address can be specified as starting address. If an address where no Script command starts is specified, an error is probably output. The Script address is indicated in 4-digit spelling.

Examples: G0000 or also G0043.

#### MemoryDump

The Gateway's user memory can be read out. Each declared variable has a valid address in the user memory. Valid addresses occur when the area ranges between 0 and 8000 (hex). All other addresses are invalid memory addresses. The returned values are probably invalid.

The requirement of a Memory Dump occurs by the command Mxxaaaa, where xx is the amount of the data bytes that are to be requested and aaaa the start address, in HEX-ASCII notation in each case: the return consists of 2\*xx+2Byte. The first byte specifies the length of the returned data, followed by the data in HEX-ASCII format. A maximum of 128 byte may be read per call; a higher number might result in corrupt data during the execution.

Two special cases exist:

- Reading out the fieldbus input buffer: the start address is FFF0  
Here the length reflects the actual number of used data in the bus and not the number of read bytes.
- Reading out the RS- input buffer: the start address is FFE0  
Here the length equals the number of data in the RS-input buffer.

#### Programming

The current Script is only in the RAM after the download, and it can only be executed as long as the Gateway is supplied with voltage. The current Script has to be programmed in order to load it into the non-volatile memory. This happens by sending the character E to the Gateway. After a successful transfer of the Script the Gateway responds with "O", in case of an error with "E".

## Reset

The Reset of the Gateway is initiated with R. After the Reset, however, the Script that is currently in the RAM will be rejected, and the Script that was recently included in the EEROM is activated on a restart. For this reason the current Script should either be transferred to the EEROM (Programming) or it should be loaded once again into the RAM after a restart. A download is always carried out after a restart since the Debugger is not in the position to decide whether the EEROM also contains the current Script.

After a Reset the Gateway carries out a Restart and sends a Start-message. Subsequently it will output its state.

## SingleStep

The Gateway can be given instructions to process a single Script command or a sequence of commands. The number of commands has also to be mentioned. The command is "Sxx", where x is the number of commands. Exception: If "00" is specified as x, 256 commands are executed. After the execution of the command (the commands) the Gateway outputs a status.

## StartMessage

In case a Gateway is restarted in the Debug mode (cold start through voltage or warm start through Reset) it outputs the character string CR LF (0x0D 0x0A). After receipt of this sequence a connected program is able to detect that the device was restarted.

## State

After different conditions the Gateway will output its State Message.

1. after a Reset Message.
2. when reaching a Breakpoint
3. after a Single Step execution
4. after the execution of a Stop Script command
5. after a stop through the Debugger

The structure of the message itself is always the same:

lbbppppeeaa:ss:

**l:** Constant key character

**bb:** Breakpoint

**pppp:** ProgramCounter, the currently reached position of the Script

**ee:** ErrorCode, error state at the time of the Break 00, no error

**aa:** The address where the error occurred

**ss:** The Stack pointer

## Stop

By sending the character X to the Gateway the current Script is stopped on the next command. The Gateway sends out a status message, the break number is FF.

## Upload

With sending the command U to the Gateway the device outputs its memory. At present this content is limited to 32 byte.

Format of the upload:

1 word binary (data bytes), data in binary format, 1 word checksum (binary);

The checksum is generated as 16 bit sum with all data byte.

**WriteMemory**

Equivalent to reading out the user memory you can also write in the memory. Per call a maximum of 128 byte can be overwritten in the memory. The format for the call is Waaaaxd1d2..dn, where

**aaaa** is the start address,

**xx** is the length of the data and

**d1..dn** are the data themselves.

The data have to be in Hex-ASCII format. The execution of the command is carried out immediately and does not come up with a confirmation after the execution.



The commands listed here as well as the corresponding definitions are available in English only. Since all commands are in English a programmer who understands these commands should also be in the position to cope with the English definitions of the commands. On account of the topicality it is quite common in this field that the translations are not updated.

The definitions of the commands cannot be kept up to date in this document. Therefore it is possible that newer commands or corrections are not included here. They are only available in the Online Help.

Here also the examples are only to be seen as short excerpts. Further examples as well as complete scripts can be found on our website.

## 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

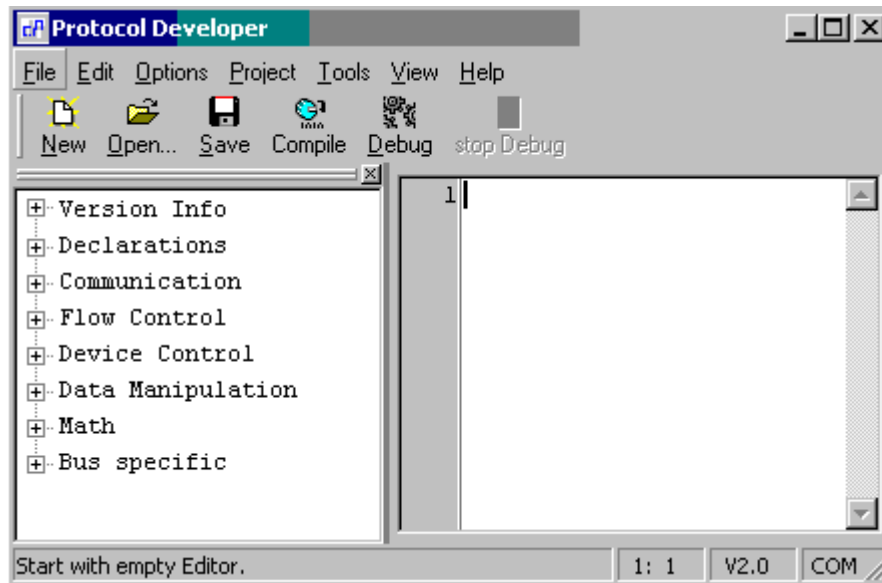
#### Connecting the device (Power and Debug)

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"
```

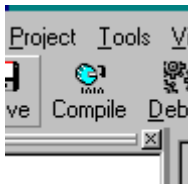
```
// we declare a variable and define this variable with a fix value
var vw_a: word;
MoveConst (vw_a, 100);
```

```
// stop the script.
stop;
```

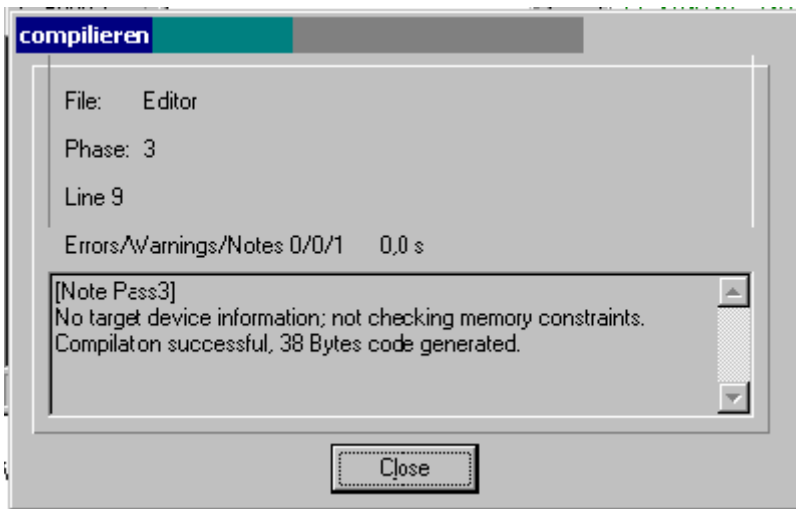
#### 7.1.4 Step 4

##### 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:

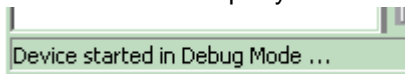


The script is now checked for syntax and a compressed code is generated for the code.

#### 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.

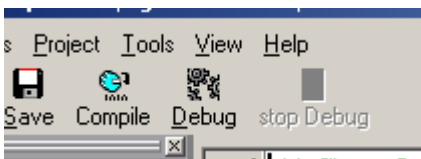


From now on it is possible to debug the script.

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

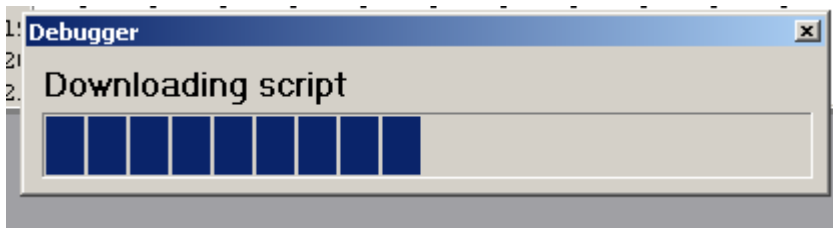
Every time the device is switched off and on again you will see this message.

**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.



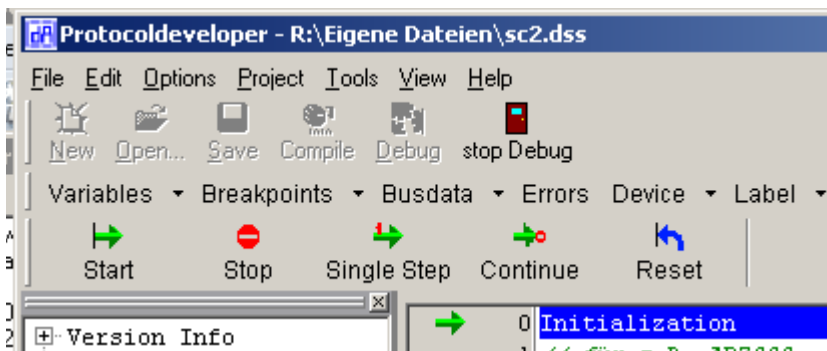
When you start a debug session the script is automatically compiled and downloaded to the script device. This transmission takes a short while, depending on the size of a script. For this time you see a progress bar.

### 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:

variable	value	type	address	Size
vw_a	0000	WORD	0x0004	2

### 7.1.7 Step 7

#### Single Step

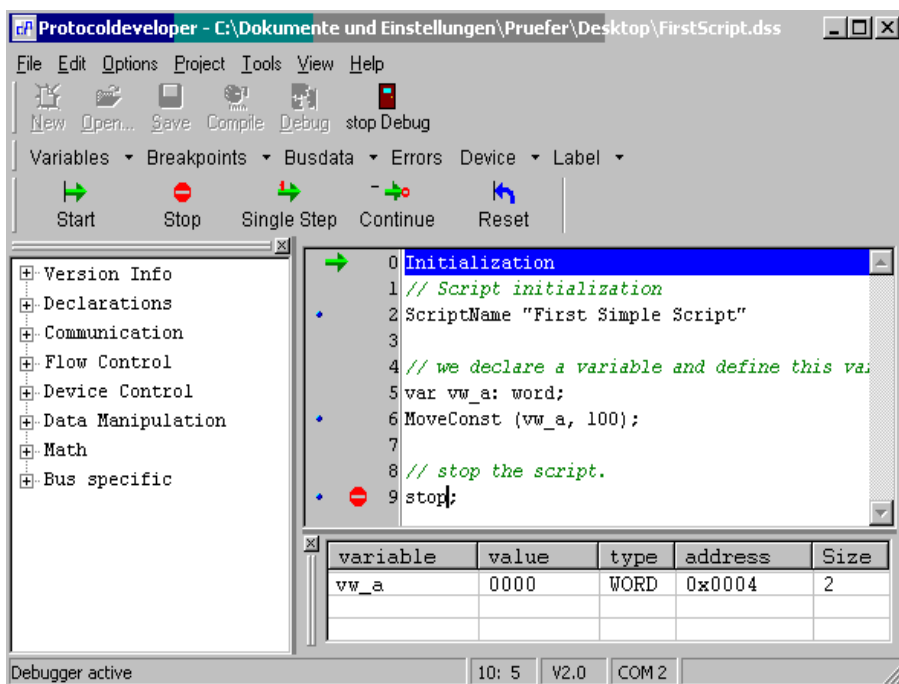
You may now click onto the single step button until the script is at the STOP command. After every step the Protocol Developer refreshes all variables used in the variable window, so after executing the line "MoveConst ..." our variable window looks like this:

variable	value	type	address	Size
vw_a	0100	WORD	0x0004	2

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!

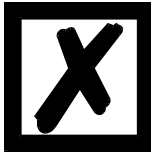
Now you have learnt how to use the Protocol Developer. Most functions are similar to the operations shown in the last steps. This short introduction does not show all debugging or script writing techniques, but it shows the principles of a script and how to create and debug it.

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](http://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](http://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 ( RSSwitchInfo , RS_Switch );
// 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-dependent. 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
3456789012345678901234567890abcdefghijklmnopqrst" );
  // 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 try 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](http://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 Number-ToReceive 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=0xFF  
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 (BusOutputSize, 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](http://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 parameters!

### 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-dependent.

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.

ProfibusDP 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 ARNET\_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:            bitwise xor operation, result is a byte.

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

Sum:            bitwise add operation, result is a word.

SumNot:        bitwise 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 ProfibusDP).

## 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 65535. 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 65535. 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  $256 * \text{consumed size} + \text{produced size}$ .

### 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 is 1 (=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  $60000 = 60 * 1000 \text{ ms} = 1 \text{ minute}$ .

## 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).

